University of California

Los Angeles

Iterative Information Processing on Unreliable Hardware: An Information Theoretic Approach

A dissertation submitted in partial satisfaction

of the requirements for the degree

Doctor of Philosophy in Electrical Engineering

by

Chu-Hsiang Huang

2015

Abstract of the Dissertation

Iterative Information Processing on Unreliable Hardware: An Information Theoretic Approach

by

Chu-Hsiang Huang

Doctor of Philosophy in Electrical Engineering

University of California, Los Angeles, 2015

Professor Lara Dolecek, Chair

In traditional information processing systems, inference algorithms are designed to collect and process information subject to noisy transmission. It is saliently assumed that the inference algorithms themselves have error-free implementations. However, with the scaling of process technologies and the increase in process variations, nano-devices will be inherently unreliable. Producing reliable decisions in systems with unreliable components thus becomes an important and challenging problem. In this dissertation, we provide a novel information theoretic approach to analyze and develop robust system design for iterative information processing algorithms running on noisy hardware. We characterize the fundamental performance limits of the systems under the joint effect of communication/environment noise and hardware noise. Based on this analysis, we then propose new theory-guided methods that guarantee reliable performance under hardware errors of varied characteristics. The proposed methods successfully explore the inherent robustness of the information processing algorithms and leverage the error-tolerance of the considered applications to minimize the overhead introduced in robust system design.

We investigate a wide range of iterative information processing systems implemented on noisy hardware via the proposed information theoretic approach.

Starting from iterative message passing decoders, we study different decoder implementations including finite-precision and infinite precision decoders subject to various types of hardware errors. We identify the performance-critical components in the iterative decoders via a theoretical analysis and develop robust system designs to assign computation units with different error characteristics to different components in the decoder. Then, we apply the proposed analysis and design methodology to general inference problems on probabilistic graphical models and develop robust implementations of the general belief propagation algorithms by noise cancellation based on averaging. For certain applications with error-tolerance, e.g. image processing and classification based on machine learning, we propose theory-guided adaptive coding schemes inspired by approximate computing to correct errors without additional hardware redundancy. The redundant free codes have the same performance as the traditional codes. Our algorithm-guided approach offers up to 100x reduction in the error rates relative to the nominal system designs.

Table of Contents

List of Figures

List of Tables

Acknowledgments

This dissertation would not have been possible without the support of my family, colleagues, teachers, and friends. First and foremost, I want to thank my parents and my brother, whose love, patience, and support were ever-present. In the UCLA Department of Electrical Engineering, I want to thank my advisor Professor Lara Dolecek, whose kindness and support enabled me to pursue a variety of interesting topics during my time as a graduate student. Her enthusiasm for research and insight into problem solving were a great source of inspiration during the time I spent in the Laboratory for Robust Information Systems. I also want to thank Professors Greg Pottie, Danijela Cabric, Mani Srivastava, Puneet Gupta, and Mario Gerla for the many positive interactions I've shared with them. While at UCLA, I have had the great fortune of knowing many other professors in the department; to all of you, I am grateful. I want to thank the support from NSF Variability Expedition and IARPA RiM4S, they are very interesting projects, I did many good research works and learned a lot from participating in these projects. I have benefited in many ways from other students and postdoc researchers I've known. I am honored to have worked with Fred Sala, Behzad Amiri, Clayton Schoeny, Yao Li, Nicolas Bitouze, Yuta Toriyama, Abbas Rahimi, Sadegh Tabatabaei, Supriyo Chakraborty, Derrick Chang, and Sherry Wu among others. To the new students in the group, Amir and Kayvon, I wish each of you the best of luck over the course of your graduate careers. To my friends and colleagues at Qualcomm, I am eternally grateful. I was an intern with the Qualcomm LTE advanced receiver group in the summer of 2014. This internship experience provided me with a wealth of knowledge and experience in communication system design, which has influenced my path to the present day. I am especially thankful for my manager Mike Mccloud and Raghuraman Krishnamoorthi; if I have enjoyed any success in my career, it is due in no small part to them.

I also want to thank all the friends I met at UCLA and in Los Angeles: Ayca Balkan, Shaunak Mishra, Can Karaku, Sina Basir-Kazeruni, Chun-Hao Liu, Alvin Chen from the UCLA EE department, Chao-Wei Tsai, John Yang, Shelly Teh, Jackie Lin, Rick Hsu, Dennis Huang, Jaylen Wang, Stephanie Tsung, Jason You, Zack Chang, Riva Chang, Charles Chao, Edward Yang, Brady Lin, Ben Chang, Janie Chen, En-Wei Lin, Vicent Wen, Maria Hsiao, Hanky Wang, Liza Li, Jason Chang in the UCLA Taiwanese Volleyball team, Shine Tu, Hung-Bin Chang, Kyle Liang, Wei-Hung Liu from the NTU BL504 Lab, Melanie Wu, Luming Huang, I-Chuan Hsu, Ling Chao, Sarah Tsai in the Pepperdine-UCLA enjoy life club, Sherry Lin, Selena Chueh, Fiona Hsiao, Shelley Wu, Paige Hsu, Doris Chen, Yumi Yang, Yen Chen, Ming Yu, Janice See, Kayleah Rhyme, John Loden, Angella Chen, Angela Chang, Dorothy Wang, and all the other good friends.

Finally I want to thank my girl friend Shelly, whose love is the strongest support during my four-year PhD life.

Vita

2007	B.S. in Electrical Engineering, National Taiwan University.
2009	M.S. in Electrical Engineering, National Taiwan University.
2010–2011	Research Assistant, INTEL-NTU Research Center.
2011–2012	Graduate Student Fellowship, University of California, Los Angeles.
2013–2014	Teaching Assistant, Department of Electrical Engineering, University of California, Los Angeles.
2012–2015	Research Assistant, Department of Electrical Engineering, University of California, Los Angeles.

Publications

S. M. S. Tabatabaei Yazdi, **C.-H. Huang**, and L. Dolecek, "Optimal design of a Gallager B noisy decoder for irregular LDPC codes," in *IEEE Commun. Lett.*, vol. 16, no. 12, pp. 2052–2055, Dec. 2012.

C.-H. Huang, Y. Li, and L. Dolecek, "Gallager B LDPC decoder with transient and permanent errors," *IEEE Trans. Commun.*, vol. 62, no. 1, pp. 15–28, Jan. 2014.

C.-H. Huang, Y. Li, and L. Dolecek, "Belief propagation algorithms on noisy hardware," *IEEE Trans. Commun.*, vol. 63, no. 1, pp. 11–24, Jan 2015.

Y. Li, **C.-H. Huang**, and L. Dolecek, "Orthogonal matching pursuit on faulty circuit," IEEE Trans. Commun., 2015, accepted.

C.-H. Huang, Y. Li, and L. Dolecek, "ACOCO: Adaptive coding for approximate computing on faulty memories," submitted to *IEEE Trans. Commun.*, 2015.

C.-H. Huang and L. Dolecek, "Analysis of finite-alphabet iterative decoders under processing errors," in IEEE ICCASP, 2013.

C.-H. Huang, Y. Li, and L. Dolecek, "Gallager B LDPC decoder with transient and permanent errors," in IEEE ISIT, 2013.

X. Wu, H.-I. Chang, **C.-H. Huang**, Y. Wang, L. Dolecek, and G. Pottie, "Virtual Inertial Measurements for Motion Inference in Wireless Health", in IEEE ACSSC, 2013.

M. Laghate, **C.-H. Huang**, C.-K. Yu, L. Dolecek, and D. Cabric, "Identifying Statistical Mimicry Attacks in Distributed Spectrum Sensing", in IEEE ACSSC, 2013.

C.-H. Huang, Y. Li, and L. Dolecek, "Noisy belief propagation decoder," in IEEE ACSSC, 2014.

C.-H. Huang, Y. Li, and L. Dolecek, ”Adaptive error correction coding scheme for computations in the noisy min-sum decoder,” in IEEE ISIT, 2015, accepted.

CHAPTER 1

Introduction

Modern CMOS circuits exhibit increasing variation in their performance, power consumption, and reliability parameters. The design of digital systems relying on these circuits must take such variation into consideration. A promising alternative to overdesigning and guardbanding is to develop robust system designs tolerant of hardware errors. Traditionally, the inference system design process considers only the uncertainty in the environment and the source. The inference machines themselves are assumed to be error-free (Fig. 1.1). In this dissertation, we consider a fundamentally new problem: how can we achieve reliable inference when the system components are unreliable (Fig. 1.2)? To answer this question, we study inference algorithms implemented on hardware subject to different kinds of errors (noisy hardware in brief), and develop robust system designs via an information theoretic approach. Information theory is one of the fundamental disciplines underlying the quantitative characterization of the randomness and uncertainty in inference systems. Many communication systems are developed based on information theory. Our study of unreliable hardware focuses on the modeling and analysis of the uncertainty of hardware operations. Therefore, information theory is a suitable foundation on which we can build our results on performance characterization and robust hardware-error-aware system design.

Since forthcoming nano-scale devices will be error-prone, characterizing the reliability of systems built out of unreliable hardware components has recently become an important concern in inference algorithm implementation. The study

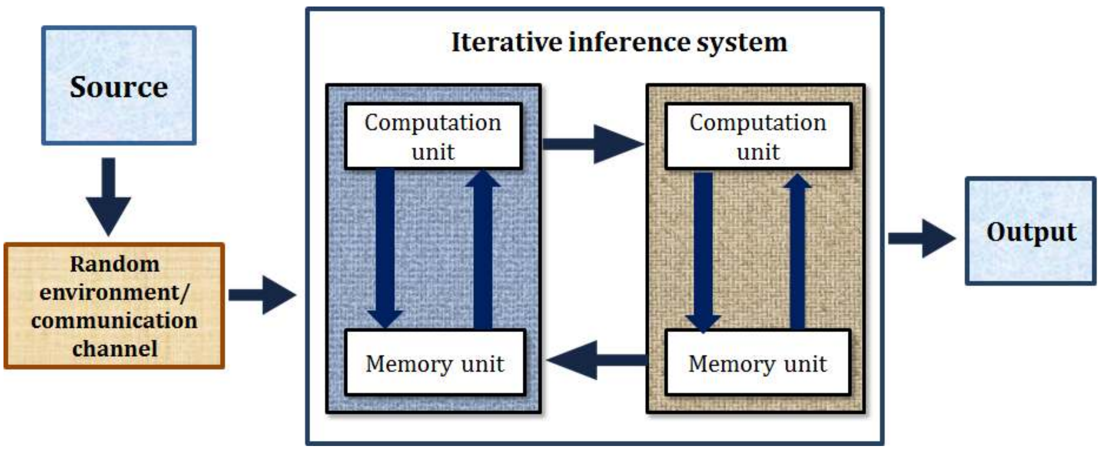

Source
Random environment/ communication channel
Iterative inference system
Computation unit
Memory unit
Computation unit
Memory unit
Output

Figure 1.1: System model: traditional inference system.

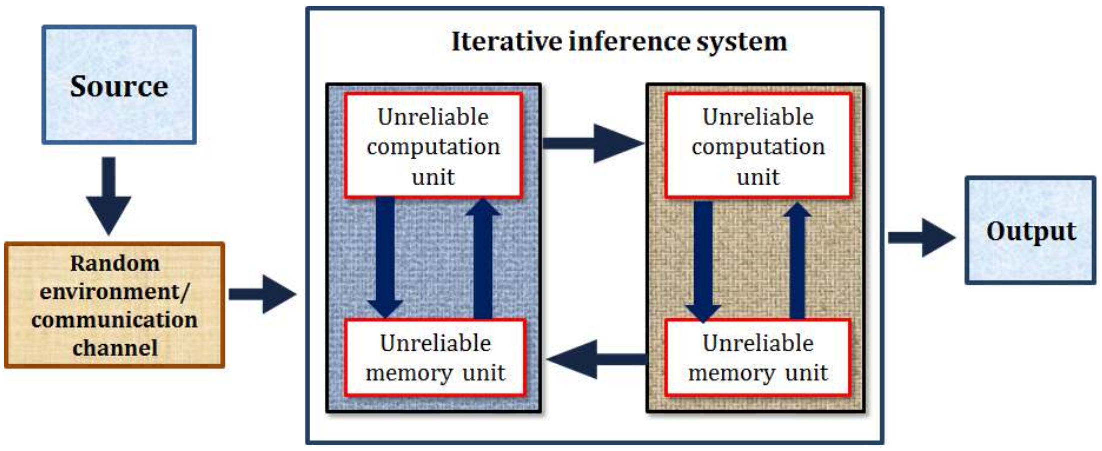

Source
Random environment/ communication channel
Iterative inference system
Unreliable computation unit
Unreliable memory unit
Unreliable computation unit
Unreliable memory unit
Output

Figure 1.2: System model: inference system on unreliable hardware.

of circuits made out of faulty gates dates back to von Neumann's model of transient errors [1]. He also proposed a triple-modular-redundancy (TMR) scheme, in which the same task is performed in three identical logic blocks and the output is decided by a majority vote, to combat transient errors [1]. TMR remains a popular concept in modern circuit design and many schemes based on TMR have been developed [2, 3]. Transient error modeling also provides a theoretical basis for fault-tolerant computing schemes [4, 5]. Besides the efforts in fault-tolerant system design, a number of more refined models were proposed for different types of hardware errors, such as, for instance, the soft adder model [6, 7] and the Markov chain model [8].

Besides logic gates, memory elements in digital circuits are also vulnerable to errors. Furthermore, memory failure rates are increasing due to the impact of shrinking dimensions, high integration densities, lower operating voltages, etc. [9–11]. Theoretical analysis for fault-tolerant memory system design was discussed in [12] and [13]. Error correction coding was introduced to mitigate the effects of memory cell errors by redundant bits stored in additional memory cells [14, 15].

Stochastic computation enables the application of communication-inspired methods to analyze and recover the processing errors by treating unreliable devices as noisy communication channels/networks [16]. This approach became popular in recent work on reliable computing [16, 17]. In [18], an optimization problem is formulated to establish a trade-off between energy consumption and processing error rate. Data fusion for error-correction in N-modular redundancy (NMR) with different processor error rates was shown to improve the efficiency of NMR schemes [19].

Approximate computing has attracted significant interest in recent years for its capability to trade computational accuracy for data processing throughput or energy efficiency [20–22]. This technology was motivated by a large and growing class of applications that are inherently error-tolerant. Such applications include multimedia signal processing, statistical learning and wireless communications [21]. A number of earlier efforts achieved promising results by exploring approximate computing both in software and in hardware [23].

In this dissertation, we study the effect of hardware errors on inference systems and develop robust inference system designs, capable of handling hardware errors, via an information theoretic approach. First, we characterize the fundamental performance limits of inference algorithms implemented on noisy hardware. Previous work mostly focused on the performance of the individual components. In the present work, we move to the system level in order to study the joint effect of environment/communication noise and hardware noise on the error performance of the entire inference system. Each component is described by an abstract model capturing its error characteristics. The performance analysis is derived based on these error models. Via such system level analysis, we quantify the effects that different operations within the algorithm have on the overall quality of inference, and then identify unconventional error characteristics of the inference algorithms on noisy hardware.

Based on this theoretical analysis, we offer design approaches for algorithm implementations that minimize the effect of hardware errors with performance guarantees. Though previous works proposed many schemes to improve system performance when using unreliable hardware, there is still a lack of theory-guided robust system design with guaranteed error mitigation capability. By utilizing the inherent robustness of inference algorithms, we develop algorithm implementations with mathematically-guaranteed robustness against hardware errors. Moreover, a new design dimension of inference algorithm implementation is explored based on the analysis of the joint effect of environment/communication noise and hardware noise, e.g., resource allocation across different units in inference systems when hardware components with different degrees of reliability are available.

For applications that can tolerate small errors, e.g., image processing or data classification, we further study approximate computing through our theoretical approach. Although many approximate computing techniques have been proposed recently, there is still a need for analytical evaluation of the effect of the distortion introduced by relaxing the computation accuracy requirement. In addition, we develop theory-guided approximate computing techniques. Based on our analysis, we propose a systematic methodology to analytically evaluate the effect of the introduced distortion on system error performance and we develop approximate computing techniques with a minimum effect on system output.

1.1 Summary of Contributions

We briefly outline the contributions of each chapter below.

1.1.1 Chapter 2 Contributions

We first investigate the performance of popular iterative decoders for broadly deployed low-density parity check (LDPC) codes implemented on noisy hardware. Through a recursive analysis, we prove that different components of an iterative decoder have different effects on the error performance. Specifically, we show theoretically and confirm experimentally that the decoder output error rate is dominated by the errors in the output messages at the variable nodes. These findings enable us to explore a new dimension in system design. We propose an optimal resource allocation scheme applicable when computational units with varying degrees of reliability (and, naturally, cost) are available, a scenario that is appropriate when the variable nodes and check nodes in an LDPC code have non-uniform degrees. Lower decoder error rates under the same implementation cost can therefore be achieved using informed resource allocation. Moreover, our theoretical analysis also reveals the inherent robustness of the iterative decoders: as long as the hardware error rate is small enough, iterative decoders are still able to correct most of the errors from the communication channel such that the residual errors are due to unreliable hardware only. Based on this finding and the observation that the check-sum constraints in LDPC codes can detect both errors from the communication channel and from unreliable hardware, we propose a scheme to detect permanent errors in the memory cells that store intermediate computations between successive decoding iterations. The proposed detection scheme utilizes the decoder structure (check-sum constraints) without adding any redundant components, and offers mathematically-guaranteed detection performance.

1.1.2 Chapter 3 Contributions

Given the initial success of the analysis of LDPC decoders implemented on noisy hardware, we broaden the scope of our analysis to include general belief propagation (BP) algorithms for inference over probabilistic graphical models. We characterize the BP algorithm on noisy hardware and propose robust implementations of BP with mathematical guarantees. In particular, we introduce averaging BP, in which the effects of computation noise are reduced by averaging messages computed by BP over all up-to-date iterations. Theoretical analysis of averaging BP shows that the accuracy of noise-free BP can be achieved by averaging BP on noisy hardware. In the application examples of BP for image denoising and BP LDPC decoding, we demonstrate the effectiveness of mitigating computation noise by averaging BP.

1.1.3 Chapter 4 Contributions

Inspired by approximate computing techniques, we further relax the requirement of accurate inference (that is, guaranteeing performance approaching that of the noise-free implementation) to design more efficient (in terms of overhead) systems robust against hardware errors. We propose the Adaptive Coding for Approximate Computing (ACOCO) framework. In ACOCO, we first compress the data by introducing distortion in the source encoder, and then add redundant bits to protect the data against memory errors in the channel encoder; thus we can protect the data against memory errors without additional memory overhead. Although we introduce a little distortion in the source encoder, the channel codes can correct harmful memory errors and therefore the proposed code is still guaranteed (mathematically) to improve the system performance. Most importantly, we design the source encoder by first specifying a cost function measuring the effect of the data compression on the system output, and then design the source code according to this cost function. We demonstrate the effectiveness of ACOCO by developing adaptive codes which improve the performance of several communication and machine learning systems, including max-product image denoising, naïve Bayesian classification, and min-sum LDPC decoders.

1.2 Hardware Error Models

In this dissertation, we use the modeling methodology proposed in the stochastic computation framework [16]. Under the stochastic computation framework, a unreliable hardware unit is interpreted as a noisy communication channel. To be more specific, a noisy hardware unit is modeled by passing the output of a hypothetical noise-free hardware unit through a hardware error channel that models the effects of hardware errors. The output of the hardware error channel is the real output of the noisy hardware unit.

The first hardware error channel model we consider is the transient error model. Following the transient error model proposed by von Neumann [1], we assume that the transient errors on different computation/memory units are independent, i.e., the noises in the hardware error channels modeling the transient errors are

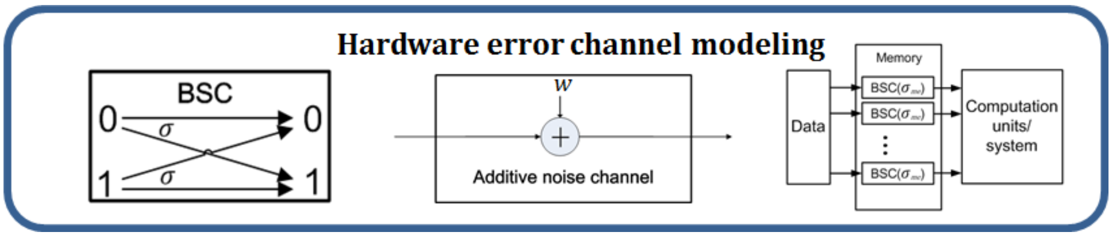


Figure 1.3: Example of hardware error channel models.

independent. Due to the transient errors, the actual output value can be different from its prescribed value.

In addition to the transient error model, we also consider the permanent error model. The permanent error model captures the effect of defects on a hardware unit output. Transient errors and permanent errors are quite different in nature. Under a transient error, the hardware unit provides an output that is erroneous with some probability less than one, independent of all other computations. Under a permanent error, the hardware unit repeatedly provides the same erroneous output.

In the following chapters, we use different channel models, e.g., binary symmetric channel (BSC) and additive noise channel (Fig. 1.3), designed for different kinds of transient and permanent errors in various types of inference systems. Each hardware error channel model captures the effect of a particular type of hardware error, and we analyze the fundamental performance limits of inference systems and develop robust algorithms based on these hardware error channel models.

CHAPTER 2

Noisy Iterative Message Passing Decoders

We investigate iterative message passing decoders implemented on noisy hardware in this chapter. First, we give a brief overview of iterative message passing decoders and previous work on noisy hardware related to LDPC codes and decoders. Next, we separately consider two cases: (1) decoders implemented on hardware experiencing only transient errors and (2) decoders implemented on hardware experiencing both transient and permanent errors. We characterize the performance of the decoders and develop schemes to reduce the error rates in both cases.

2.1 Background and Previous Work

2.1.1 LDPC Codes

A low-density parity-check (LDPC) code is a linear block code with relatively few parity-check constraints. LDPC codes can be conveniently represented by a sparse bipartite graph composed of variable nodes, check nodes, and edges each connecting a variable node with a check node. In a (d_v, d_c)-regular LDPC code, each variable node is connected to d_v check nodes, and each check node is connected to d_c variable nodes. The low edge density in the bipartite graph representation of an LDPC code allows for low-complexity local message-passing decoding algorithms that are known to approach the performance of high-complexity MAP (maximum *a posteriori*) decoding algorithms. Irregular LDPC codes are codes which have different node degrees for each variable and check node. Detailed definitions will be provided later in Section 2.2.1.

2.1.2 Noise-Free Gallager B and Gallager E Decoders

The Gallager B and Gallager E decoders are popular choices for message passing decoders due to their good performance and low complexity [24]. We describe both decoders in the following.

In the Gallager B decoder, the messages passed along edges of the bipartite graph are binary. Denote the set of check nodes connected to the variable node v on the bipartite graph as $\mathcal{N}_v$ and denote the set of variable nodes connected to the check node c as $\mathcal{N}_c$. Let y_v be the decoder input at the variable node v, $y_v \in \mathbb{F}_2$. For completeness, let us briefly summarize the main steps of the noise-free Gallager B decoder:

- (Initialization) At iteration $\ell = 0$: each variable node v sends the message $\tilde{m}_{v,c}^{(0)} = y_v$ to every check node $c \in \mathcal{N}_v$.
- (Variable node majority voting) At each iteration ℓ, $\ell > 0$, each variable node v sends the message $\tilde{m}_{v,c}^{(\ell)}$ to the check node $c \in \mathcal{N}_v$,

  $$\tilde{m}_{v,c}^{(\ell)} = \begin{cases} s, & \text{if } |\{c' \in \mathcal{N}_v \backslash \{c\} : \tilde{m}_{c',v}^{(\ell-1)} = s\}| \geq \tilde{b}^{(\ell)}, \\ y_v, & \text{otherwise,} \end{cases}$$

  where $\tilde{b}^{(\ell)}$ is the voting threshold for the noise-free Gallager B decoder. We assume that the threshold $\tilde{b}^{(\ell)}$ is at least $\lceil d_v/2 \rceil$ so that the variable-to-check message is uniquely specified.
- (Check node xor-sum) At iteration ℓ, $\ell \geq 0$, each check node c sends a message $\tilde{m}_{c,v}^{(\ell)} = \sum_{v' \in \mathcal{N}_{c \backslash \{v\}}} \tilde{m}_{v',c}^{(\ell)}$ to the variable node $v \in \mathcal{N}_c$. The summation operation is in $\mathbb{F}_2$.

The Gallager E decoder has message alphabet $\{-1, 0, +1\}$. We also summarize the main steps of the noise-free Gallager E decoder as follows:

- (Initialization) At iteration $\ell = 0$: each variable node v sends the message $\tilde{m}_{v,c}^{(0)} = y_v$ to every check node $c \in \mathcal{N}_v$.
- (Variable node) At each iteration ℓ, $\ell > 0$, each variable node v sends the message $\tilde{m}_{v,c}^{(\ell)}$ to the check node $c \in \mathcal{N}_v$,

  $$\tilde{m}_{v,c}^{(\ell)} = \operatorname{sgn}\Big(\varrho^{(\ell)} y_v + \sum_{c' \in \mathcal{N}_v \setminus c} m_{c',v}^{(\ell-1)}\Big),$$

  where $\varrho(\ell)$ is an appropriately chosen weight.
- (Check node) At iteration ℓ, $\ell \geq 0$, each check node c sends the message $\tilde{m}_{c,v}^{(\ell)} = \prod_{v' \in \mathcal{N}_c \setminus \{v\}} \tilde{m}_{v',c}^{(\ell)}$ to variable node $v \in \mathcal{N}_c$.

2.1.3 Previous Work: LDPC Codes and Decoders on Noisy Hardware

The use of low-density parity-check (LDPC) codes and their decoders implemented on noisy hardware has recently garnered substantial research attention. Vasic and Chilappagari [13] were the first to propose an approach to recover from errors in memory cells for the Taylor-Kuznetsov memory structure [25, 26] using an LDPC code with a Gallager B decoder. Additionally, in [27] and [28], the same authors analyzed the use of expander codes [29] to efficiently recover from memory errors. A one-step majority logic LDPC decoder and its application to recovery from errors in faulty hardware was studied by Vasic and coauthors in [30] and [31]. In [32], Yeung and Chugg experimentally studied the performance of an LDPC belief propagation decoder under permanent errors. Winstead and Howard empirically demonstrated in [33] that a faulty LDPC decoder can successfully recover the stored data from hardware with permanent and transient errors with the aid of error-free elements. As recently shown by Tang *et al.* [34], even without error-free elements, an LDPC decoding method can still significantly reduce the error probability. Leduc-Primeau and Gross explored a clever message repetition scheme in [35] to mitigate computational errors arising in a noisy Gallager B decoder.

The capacity and certain concentration results for a noisy LDPC message passing decoder were first computed in [36] by Varshney. Tarighati *et al.* analyzed the density evolution of a noisy sum-product message passing LDPC decoder in [37]. Tabatabaei *et al.* reported in [38] the fundamental performance limits of regular LDPC codes with binary and non-binary decoders implemented on noisy hardware with transient errors.

2.2 Decoding Under Transient Errors

In this section, we study the performance of irregular LDPC codes on a general M-alphabet iterative decoder. Processors with different processing error rates are assigned to different computational units (i.e., variable nodes and check nodes) of a noisy M-alphabet iterative decoder. We analyze the bit error rate (BER) of a noisy M-alphabet iterative decoder and develop the optimal resource assignment method to optimally distribute processors across different components in the decoder.

2.2.1 Faulty Decoder under Transient Errors

We consider an irregular LDPC code $\mathcal{C}$ described by a bipartite graph $G = G(V, E)$. Here V denotes the set of nodes in the bipartite graph, and E is the set of edges connecting variable nodes and check nodes. Let m denote the number of check nodes, and n denote the number of variable nodes. We also let $\mathcal{N}_v$ ($\mathcal{N}_c$) denote the set of checks (variables) incident to variable node v (check node c). Suppose d_v is the largest variable node degree and d_c is the largest check node degree. For $1 \leq i \leq d_v$, following popular notation [39], we let λ_i denote the fraction of edges in G that are connected to variable nodes of degree i. Also, for $1 \leq j \leq d_c$, we denote by ρ_j the fraction of edges in G that are connected to check nodes of degree j. It is useful to define $\lambda(t) := \sum_{i=1}^{d_v} \lambda_i t^{i-1}$ and $\rho(t) := \sum_{j=1}^{d_c} \rho_i t^{j-1}$, respectively, as the variable and check degree polynomials. Finally, the collection of graph codes whose bipartite graphs follow $\lambda(t), \rho(t)$ distributions is referred to as the (λ, ρ) ensemble.

Following the unreliable hardware inference system model in Fig. 1.2 and the hardware error model described in Section 1.2, we assume that iterative messages exchanged between variable and check nodes are subject to transient errors, and that these errors are independent across different computational units and across different iterations of the decoder. Following the set-up presented in [40], we consider a general framework that allows processors of different error probabilities at different check nodes and variable nodes. We remark that in practice, these error probabilities depend on the choice of implementation.

Let us assume that we have L types of processors available for the implementation of variable nodes and check nodes, and that that these processors are characterized by distinct error probabilities q_i, $1 \leq i \leq L$, ordered in ascending order from best to worst. We collectively refer to the ascending ordering of q_i's as Q.

For the noisy decoder, the message exchange is iteratively performed as follows: the message from variable node v at iteration ℓ is denoted $m_{v,c}^{(\ell)}$ and the message from check node c at iteration ℓ is denoted $m_{c,v}^{(\ell)}$. It is useful to specify two auxiliary messages, $\hat{m}_{v,c}^{(\ell)}$ and $\hat{m}_{c,v}^{(\ell)}$, which respectively represent the outgoing messages of noise-free processors. Let τ_{i,q_j}^{v} represent the fraction of edges that are connected to variable nodes of degree i, $1 \leq i \leq d_v$, and error q_j. Likewise, we let τ_{k,q_r}^{c} represent the fraction of edges that are connected to check nodes of degree k, $1 \leq k \leq d_c$, and error q_r. In this set-up, we are interested in estimating the performance of the algorithm, which we seek to express in terms of the density evolution of the propagated messages.

2.2.2 Error Analysis of Noisy Finite Alphabet Iterative Decoders Under Transient Errors

We remark that, as proved in [36], density evolution under transient errors is independent of the transmitted codeword. We thus follow [36] and assume the transmission of the all-zero codeword in our analysis. As in the conventional (noise-free) density evolution [39] and as in [36], and to make the analysis tractable, we assume, moreover, that the bipartite graph is sufficiently cycle-free.

It was shown in [36] that the well-known result on the concentration of message propagation [39] for LDPC decoding algorithms still holds even in the presence of processing noise for finite alphabet iterative decoders. We thus focus on the average performance of our noisy decoder. We denote by $p^{(\ell)}$ the average error in the messages from variable to check nodes in iteration ℓ; i.e., $p^{(\ell)} = \mathbb{E}\left[\Pr\{m_{v,c}^{(\ell)} \neq 0\}\right]$, where the average is taken over the (λ, ρ) code ensemble with degree and processor error rate distribution specified by the τ_{i,q_j}^{v}'s and the τ_{k,q_r}^{c}'s. In the first iteration, this error is simply the parameter $p^{(0)}$ of the transmission channel.

We now derive a recursive expression for the error rate $p^{(\ell+1)}$ for a general finite-alphabet noisy decoder. The derivations generalize the previous result in [40] obtained for a noisy Gallager B decoder (under a binary message alphabet).

Denote the message alphabet by Θ with $|\Theta| = M$. (If $\Theta = \{0, 1\}$ the alphabet is binary.) The error rate $p^{(\ell+1)}$ of a general finite-alphabet noisy iterative decoder is derived from the probability mass function (PMF) of noise-free messages, $\Pr\{\hat{m}_{v,c}^{(\ell)}\}$ and $\Pr\{\hat{m}_{c,v}^{(\ell)}\}$. We assume that when a processor outputs an erroneous value, it does so equiprobably over all choices,

$$\begin{aligned}&\Pr\{m_x^{(\ell)} = \theta_1 | \text{processor made an error}, \hat{m}_x^{(\ell)} = \theta_0\}\\ &= \Pr\{m_x^{(\ell)} = \theta_2 | \text{processor made an error}, \hat{m}_x^{(\ell)} = \theta_0\},\end{aligned}$$

for all $\theta_0 \neq \theta_1 \neq \theta_2$, $\theta_0, \theta_1, \theta_2 \in \Theta$ for x being either c or v.

Then, the PMF of the variable-to-check message $m_{v,c}^{(\ell+1)}$ in the noisy decoder can be expressed as a function of the variable-to-check message $\hat{m}_{v,c}^{(\ell+1)}$ of the noise-free decoder,

$$\begin{aligned}
&\Pr\{m_{v,c}^{(\ell+1)} = \theta\} \\
&= (1-q_j)\Pr\{\hat{m}_{v,c}^{(\ell+1)} = \theta\} + \sum_{\theta' \in \Phi \backslash \theta} \frac{q_j}{M-1} \Pr\{\hat{m}_{v,c}^{(\ell+1)} = \theta'\} \\
&= \left(1 - \frac{Mq_j}{M-1}\right) \Pr\{\hat{m}_{v,c}^{(\ell+1)} = \theta\} + \frac{q_j}{M-1}. \qquad (2.1)
\end{aligned}$$

Similarly, after some algebra, we have

$$\Pr\{m_{c,v}^{(\ell)} = \theta\} = \left(1 - \frac{Mq_r}{M-1}\right) \Pr\{\hat{m}_{c,v}^{(\ell)} = \theta\} + \frac{q_r}{M-1}. \qquad (2.2)$$

With (2.1) and (2.2), one can then recursively compute the PMF of the messages in the noisy decoder as we now show. Let $deg(\cdot)$ denote the degree and $\varepsilon(\cdot)$ denote the error rate assigned to a particular node. The average PMF of the check-to-variable messages at iteration ℓ can be derived by taking the average of (2.2) w.r.t. processor error rate distribution τ_{k,q_r}^c,

$$\begin{aligned}
&\mathbb{E}[\Pr\{m_{c,v}^{(\ell)} = \theta\}] \qquad (2.3) \\
&= \sum_{k=1}^{d_{c,v}} \sum_{r=1}^{L} \tau_{k,q_r}^c \mathbb{E}_{deg(c')=k, \varepsilon(c')=q_r}[\Pr\{m_{c,v}^{(\ell)} = \theta\}] \\
&= \sum_{k=1}^{d_{c,v}} \sum_{r=1}^{L} \tau_{k,q_r}^c ((1 - \frac{Mq_r}{M-1}) \Pr\{\hat{m}_{c,v}^{(\ell)} = \theta\} + \frac{q_r}{M-1}).
\end{aligned}$$

We note that $\Pr\{\hat{m}_{c,v}^{(\ell)} = \theta\}$ is computed from $\Pr\{m_{v,c}^{(\ell)} = \theta'\}$ as in the density evolution derivation in the noise-free case.

Next, we derive the average PMF of the messages from the variable nodes to the check nodes,

$$\begin{aligned}
&\mathbb{E}[\Pr\{m_{v,c}^{(\ell+1)} = \theta\}] \qquad (2.4)\\
&= \sum_{i=1}^{d_{v,c}} \sum_{j=1}^{L} \tau_{i,q_j}^{v} \mathbb{E}_{deg(v)=k,\varepsilon(v)=q_j} [\Pr\{m_{v,c}^{(\ell+1)} = \theta\}]\\
&= \sum_{i=1}^{d_{v,c}} \sum_{j=1}^{L} \tau_{i,q_j}^{v} ((1 - \frac{Mq_j}{M-1}) \Pr\{\hat{m}_{v,c}^{(\ell+1)} = \theta\} + \frac{q_j}{M-1}).
\end{aligned}$$

Similarly, $\Pr\{\hat{m}_{v,c}^{(\ell+1)} = \theta\}$ is computed from $\Pr\{m_{c,v}^{(\ell)} = \theta'\}$ as in the density evolution derivation in the noise-free case. Therefore, by computing $\Pr\{\hat{m}_{c,v}^{(\ell)} = \theta_1\}$ from $\Pr\{m_{v,c}^{(\ell)} = \theta_0\}$, then $\Pr\{m_{c,v}^{(\ell)} = \theta_1\}$ from $\Pr\{\hat{m}_{c,v}^{(\ell)} = \theta_1\}$, then $\Pr\{\hat{m}_{v,c}^{(\ell+1)} = \theta_2\}$ from $\Pr\{m_{c,v}^{(\ell)} = \theta_1\}$, and then $\Pr\{m_{v,c}^{(\ell+1)} = \theta_2\}$ from $\Pr\{\hat{m}_{v,c}^{(\ell+1)} = \theta_2\}$ we arrive at the recursive expression relating $\Pr\{m_{v,c}^{(\ell+1)} = \theta_2\}$ to $\Pr\{m_{v,c}^{(\ell)} = \theta_0\}$, where $\theta_0, \theta_1, \theta_2 \in \Theta$.

For an error-free iterative decoder, the overall error at iteration ℓ,

$$\sum_{\theta \in \Theta \backslash 0} \Pr\{\hat{m}_{v,c}^{(\ell)} = \theta\},$$

converges to zero for small enough $p(0)$. In contrast, for a noisy decoder this overall error converges to some strictly positive quantity which we call the *residual error* p (or final BER). We note that p is at least $\sum_{i=1}^{d_v} \sum_{j=1}^{L} \tau_{i,j}^{v} \frac{q_j}{M-1}$.

It can be shown that p improves with higher variable node degree (see also [38]).

As an illustrative example, we consider a noisy Gallager E decoder [39]. The noisy Gallager E decoder has $\Theta = \{-1, 0, +1\}$ and $M = 3$. Denote the probabilities $\Pr\{\hat{m}_{c,v}^{(\ell)} = \theta\}$ and $\Pr\{\hat{m}_{v,c}^{(\ell)} = \theta\}$ of the messages of the noise-free decoder by $\hat{p}_{c,\theta}^{(\ell)}$ and $\hat{p}_{v,\theta}^{(\ell)}$, respectively, for $\theta \in \Theta$. Also denote the noisy decoder message PMFs by $p_{c,\theta}^{(\ell)}$ and $p_{v,\theta}^{(\ell)}$.

We wish to derive a recursive expression for $p_{v,\theta_2}^{(\ell+1)}$ in terms of $p_{v,\theta_0}^{(\ell)}$. From the analysis of the nominal error-free decoder [39], the noise-free check messages are

$$\begin{aligned}
\hat{p}_{c,1}^{(\ell)} &= \frac{1}{2}[(p_{v,1}^{(\ell)} + p_{v,-1}^{(\ell)})^{d_c-1} + (p_{v,1}^{(\ell)} - p_{v,-1}^{(\ell)})^{d_c-1}],\\
\hat{p}_{c,-1}^{(\ell)} &= \frac{1}{2}[(p_{v,1}^{(\ell)} + p_{v,-1}^{(\ell)})^{d_c-1} - (p_{v,1}^{(\ell)} - p_{v,-1}^{(\ell)})^{d_c-1}],\\
\hat{p}_{c,0}^{(\ell)} &= 1 - (1 - p_{v,0}^{(\ell)})^{d_c-1}. \qquad (2.5)
\end{aligned}$$

Thus, $p_{c,\theta_1}^{(\ell)}$ is

$$p_{c,\theta_1}^{(\ell)} = \sum_{k=1}^{d_c}\sum_{r=1}^{L} \tau_{k,q_r}^{c}((1 - \frac{3q_l}{2})\hat{p}_{c,\theta_1}^{(\ell)} + \frac{q_r}{2}), \qquad (2.6)$$

where $\hat{p}_{c,\theta_1}^{(\ell)}$ is derived from (2.5) and uses $M = 3$.

From the standard noise-free decoder analysis, we can derive the noise-free variable message, $\hat{p}_{v,\theta_2}^{(\ell+1)}$, from $p_{c,\theta_1}^{(\ell)}$, $\theta_1, \theta_2 \in \Theta$ as follows:

$$\begin{aligned}
\hat{p}_{v,0}^{(\ell+1)} &= p_{v,0}^{(0)} \sum_{(\alpha,\beta):\alpha-\beta=0} \binom{d_v-1}{\alpha, \alpha, d_v-1-2\alpha} \cdot (p_{c,1}^{(\ell)})^{\alpha}(p_{c,-1}^{(\ell)})^{\alpha}(p_{c,0}^{(\ell)})^{d_v-1-2\alpha}\\
&+ p_{v,1}^{(0)} \sum_{(\alpha,\beta):\alpha-\beta=-w^{(\ell)}} \binom{d_v-1}{\alpha, \beta, d_v-1-\alpha-\beta}\\
&\cdot (p_{c,1}^{(\ell)})^{\alpha}(p_{c,-1}^{(\ell)})^{\beta}(p_{c,0}^{(\ell)})^{d_v-1-\alpha-\beta}\\
&+ p_{v,-1}^{(0)} \sum_{(\alpha,\beta):\alpha-\beta=w^{(\ell)}} \binom{d_v-1}{\alpha, \alpha, d_v-1-\alpha-\beta} \cdot (p_{c,1}^{(\ell)})^{\alpha}(p_{c,-1}^{(\ell)})^{\beta}(p_{c,0}^{(\ell)})^{d_v-1-\alpha-\beta}\\
\hat{p}_{v,-1}^{(\ell+1)} &= p_{v,0}^{(0)} \sum_{(\alpha,\beta):\alpha-\beta>0} \binom{d_v-1}{\alpha, \beta, d_v-1-\alpha-\beta} \cdot (p_{c,1}^{(\ell)})^{\alpha}(p_{c,-1}^{(\ell)})^{\beta}(p_{c,0}^{(\ell)})^{d_v-1-\alpha-\beta}\\
&+ p_{v,1}^{(0)} \sum_{(\alpha,\beta):\alpha-\beta<-w^{(\ell)}} \binom{d_v-1}{\alpha, \beta, d_v-1-\alpha-\beta}\\
&\cdot (p_{c,1}^{(\ell)})^{\alpha}(p_{c,-1}^{(\ell)})^{\beta}(p_{c,0}^{(\ell)})^{d_v-1-\alpha-\beta}\\
&+ p_{v,-1}^{(0)} \sum_{(\alpha,\beta):\alpha-\beta<w^{(\ell)}} \binom{d_v-1}{\alpha, \beta, d_v-1-\alpha-\beta} \cdot (p_{c,1}^{(\ell)})^{\alpha}(p_{c,-1}^{(\ell)})^{\beta}(p_{c,0}^{(\ell)})^{d_v-1-\alpha-\beta},\\
\hat{p}_{v,1}^{(\ell+1)} &= 1 - \hat{p}_{v,1}^{(\ell+1)} - \hat{p}_{v,0}^{(\ell+1)}. \qquad (2.7)
\end{aligned}$$

Then, $p_{v,\theta_2}^{(\ell+1)}$ is given by

$$p_{v,\theta_2}^{(\ell+1)} = \sum_{i=1}^{d_v}\sum_{j=1}^{L} \tau_{i,q_j}^{v}((1 - \frac{3q_j}{2})\hat{p}_{v,\theta_2}^{(\ell+1)} + \frac{q_j}{2}), \qquad (2.8)$$

where $\hat{p}_{v,\theta_2}^{(\ell+1)}$ is derived from (2.7). Note that we put $M = 3$ in (2.4) to derive (2.8). Then we have $\hat{p}_{v,\theta_2}^{(\ell+1)}$ as functions of $\hat{p}_{v,\theta_0}^{(\ell+1)}$ from (2.5) to (2.8).

2.2.3 Optimal Assignment of Processors

The optimal assignment of processors to check and variable nodes with different degrees can be derived by minimizing the residual error with respect to τ^v_{i,q_j} and τ^c_{k,q_r}

This minimization problem may, in general, be difficult to solve. Fortunately, when the channel error rate is small and the error rates of constituent processors are also sufficiently small, one can show that for a code that has all variable nodes of degree at least 3, by ignoring the second order terms involving $\hat{p}^{(\ell)}_{v,0}$, $\hat{p}^{(\ell)}_{v,-1}$, $\hat{p}^{(\ell)}_{c,0}$, and $\hat{p}^{(\ell)}_{c,-1}$, the minimization problem reduces to minimizing $\sum_{i=1}^{d_v} \sum_{j=1}^{L} \tau^v_{i,q_j} \cdot q_j$, which then simply becomes a linear programming problem.

We now study the optimal assignment of processors that offers different reliabilities of different processing nodes of the decoder. An optimal assignment is an assignment that minimizes the residual error p. Suppose that for every $1 \leq j \leq L$, the cost of implementing a variable node processor of degree i for $3 \leq i \leq d_v$ (we assume that there is no variable node of degree $1, 2$) and error q_j for $1 \leq j \leq L$ is $w^v_{i,j}$, and for every check node of degree k and error q_r for $1 \leq k \leq d_c$ and $1 \leq r \leq L$, the cost is $w^c_{k,r}$.

Suppose we fix the maximum allowable cost W. The total number of variable nodes of degree i and processing error q_j is $Z\tau^v_{i,q_j}/i$ and the total number of check nodes of degree k and processing error q_r is $Z\tau^c_{k,q_r}/k$, where Z denotes the total number of edges in the LDPC graph.

As previously discussed, our aim is to solve the following optimization problem:

$$\text{Minimize: } \sum_{i=3}^{d_v} \sum_{j=1}^{L} \tau^v_{i,q_j} \cdot q_j$$

$$\text{Subject to: } Z \sum_{i=3}^{d_v} \sum_{j=1}^{L} \frac{\tau^v_{i,q_j} w^v_{i,j}}{i} + Z \sum_{k=1}^{d_c} \sum_{r=1}^{L} \frac{\tau^c_{k,q_r} w^c_{k,r}}{k} \leq W,$$

$$\sum_{j=1}^{L} \tau^v_{i,q_j} = \lambda_i, \sum_{r=1}^{L} \tau^c_{k,q_r} = \rho_k. \tag{2.9}$$

We observe that the objective function and all constraints in the preceding optimization problem are linear in terms of the variables τ^v_{i,q_j}'s and τ^c_{k,q_r}'s so that efficient algorithms can be used to solve this linear programming problem.

It is interesting to note that the objective function in (2.9) does not depend on τ^c_{k,q_r}'s. As a result, for codes without variable node degrees of less than 3, all the check nodes admit the least expensive processors (of error parameter q_L) in the optimal solution.

2.2.3.1 Simulation Results

In this section we report on experimental results. We tested the performance of two irregular codes proposed by MacKay (codes are available at [41]). Code 1 has 9972 variable nodes of which 9141 nodes have degree $i_1 = 3$ and 831 nodes have degree $i_2 = 9$. The code has $m = 4986$ check nodes all with degree 7. Code 2 has 1920 variable nodes of which 640 nodes have degree 14 and 1280 nodes have degree 18. This code has 5760 check nodes in total, 1280 nodes with degree 4 and 4480 nodes with degree 6.

In our MATLAB simulations, we considered the case with two kinds of available processors with error rates $q_1 = 10^{-4}$ and $q_2 = 10^{-3}$. The channel error was 2×10^{-3}. We assigned the cost of 10 (resp. cost of 1) to the variable nodes of degree i_1 and error q_1 (resp. error q_2), and we assigned the cost of 100 (resp. cost of 10) to the variable nodes of degree i_2 and error q_1 (resp. error q_2). For code 1, we assigned cost of 10 (resp. cost of 1) to the check node with error q_1(resp. error q_2). For code 2, the cost assignment is the same over all nodes.

Two kinds of noisy Gallager E decoders are simulated: one based on the analysis-guided processor assignment and another one based on uninformed (ran-

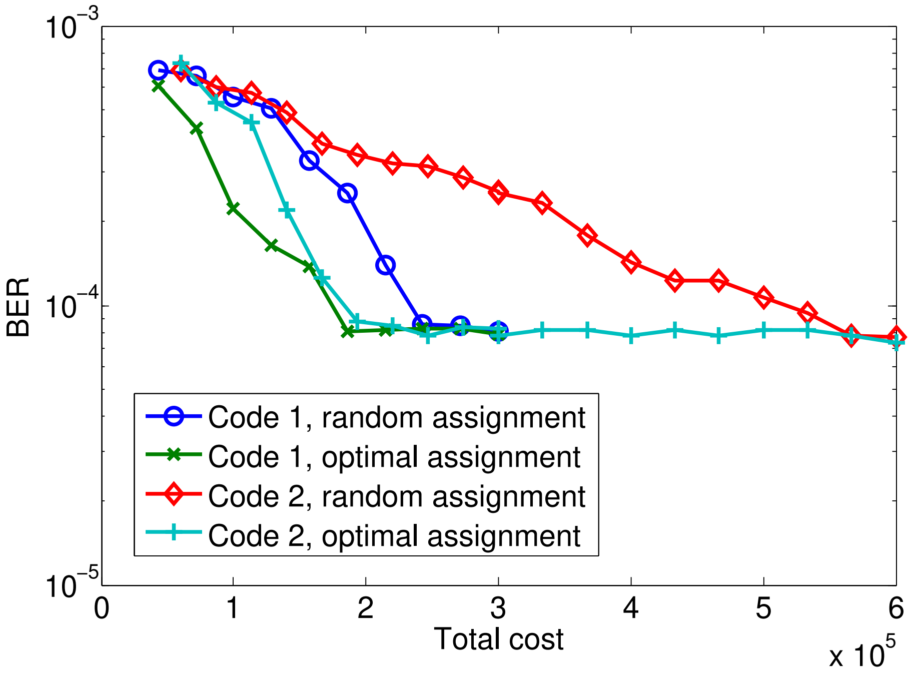


Figure 2.1: Performance comparison of optimal assignment and random assignment for the noisy Gallager E decoders.

dom) assignment of faulty processors. We plotted the resulting BERs for the two codes in Fig. 2.1 for a range of total costs (a part of the plot is suppressed to highlight the difference between the allocation choices). The simulation results are presented for the finite-length case but nonetheless corroborate the analysis (valid for the infinite-length case) and demonstrate the improvement in the performance of the decoder when processors are assigned based on the solution of (2.9). The improvement is more pronounced for code 2, which has a higher fraction of check nodes. The results show that the same BER (at about 2×10^{-4}) can be obtained by optimal assignment at about $\frac{1}{2}$ of the cost of the random assignment.

2.3 Decoding Under Transient and Permanent Errors

This section focuses on a noisy Gallager B LDPC decoder where both transient errors in faulty gates and stuck-at errors in memory cells may occur, i.e., the unreliable hardware units in Fig. 1.2 are subject to both transient and permanent errors. We develop an "asymmetric" density evolution of a noisy decoder, and propose a "self-error-detecting" scheme that identifies defective memory cells.

2.3.1 Faulty Decoder under Transient and Permanent Errors

We use BSC to model the effect of transient errors in a noisy Gallager B decoder. In addition to transient errors, we also consider permanent errors in memory cells in a noisy Gallager B decoder. We assume that the stuck-at error rates of memory cells storing the transmission channel output are γ_0 and γ_1 for stuck-at-0 and stuck-at-1 errors, respectively. Likewise, the stuck-at error rates of memory cells storing variable-to-check messages (check-to-variable messages) are α_0 and α_1 (β_0 and β_1) for stuck-at-0 and stuck-at-1 errors, respectively. Note that we allow all permanent error rates to be different.

In the next section, we characterize the performance of a noisy Gallager B decoder by establishing proper density evolution equations.

2.3.2 Analysis of a Noisy Gallager B Decoder with Memory Failures

Under the general set-up wherein the permanent error rates may depend on the stuck value (i.e., stuck-at-1 and stuck-at-0 errors occur with different probabilities), we first observe that in density evolution for asymmetric channels, the all-zero codeword is in general not a good representative for all codewords. Let w denote the relative Hamming weight (i.e., the fraction of 1's in the codeword) of a codeword as the codelength goes to infinity. We develop density evolution for two types of messages. Density evolution for asymmetric channels was treated in [42] by averaging the density over all codewords in the same codebook. In contrast, we develop the exact density evolution for a codeword of a given relative weight w. Additionally, we also allow asymmetry inside the decoder. Concentration results can be derived by following the same procedures as in [36], with the exception that the ensemble average is now parameterized by the relative codeword weight. The first type of messages are those propagating in or out of variable nodes associated with a transmitted codeword bit 1. The second type of messages are those propagating in or out of variable nodes associated with a transmitted codeword bit 0. We refer to these messages as type-1 and type-0 messages, respectively.

For $x \in \{0, 1\}$, we use $p_x^{(\ell)}$ to denote the error probability of a type-x variable-to-check message, i.e., the probability the message does not equal x, at iteration ℓ. Before presenting Theorem 1, which states the density evolution equations, we introduce some shorthand notation. First, for convenience, we let $\alpha = \alpha_0 + \alpha_1$, $\beta = \beta_0 + \beta_1$, and $\gamma = \gamma_0 + \gamma_1$.

Next, we define the following auxiliary quantities. For $w \in (0, 1)$, and $x \in$

$\{0, 1\}$, let

$$\hat{\phi}_{Cx}(z_0, z_1) = \frac{1}{2} - \frac{\big(1 - 2(wz_1 + (1-w)z_0)\big)^{d_c - 1}}{2(1 + (-1)^x (1-2w)^{d_c-1})} - (-1)^x \frac{\big(1 - 2w + 2(wz_1 - (1-w)z_0)\big)^{d_c-1}}{2(1 + (-1)^x (1-2w)^{d_c-1})}, \tag{2.10}$$

be a function of z_0 and z_1 for $z_0, z_1 \in [0, 1]$. (We will justify this expression later in this section.) Additionally, we define

$$\hat{\phi}_C(z) = \frac{1 - \big(1 - 2z\big)^{d_c-1}}{2}, \tag{2.11}$$

as a function of z for $z \in [0, 1]$.

The function $\hat{\phi}_{Cx}(z_0, z_1)$ represents the probability of error of a type-x message that is the output of a (hypothetical) noise-free check node. In particular, for this and subsequent auxiliary functions, the $\hat{\phi}(\cdot)$ notation indicates that the output is computed error-free. Here, we assume that the input to this check node consists of $d_c - 1$ messages such that the probability of error for a type-0 incoming message is z_0 and the probability of error for a type-1 incoming message is z_1.

For $w = 0$, with probability 1, every message is type-0. As a result, the expression in (2.10) no longer depends on x and simplifies to $\hat{\phi}_C(z)$. Likewise, for $w = 1$, with probability 1, every message is type-1, and the same simplification applies. Note that d_c is even in this case. In addition, when the probability of error of a type-1 message is equal to that of a type-0 message, the two types of messages are considered indistinguishable, and again the same simplification to $\hat{\phi}_C(z)$ applies.

For $x \in \{0, 1\}$, we then define

$$\hat{\phi}_{Vx}(b_x, z) = (1 - (1-\gamma)\epsilon - \gamma_{1-x}) \sum_{k=b_x}^{d_v-1} \binom{d_v - 1}{k} z^k (1-z)^{d_v-1-k} + ((1-\gamma)\epsilon + \gamma_{1-x}) \sum_{k=0}^{b_x-1} \binom{d_v-1}{k} (1-z)^k z^{d_v-1-k} \tag{2.12}$$

as a function of $z \in [0,1]$ and an integer b_x, $b_x \geq \lceil \frac{d_v}{2} \rceil$.

The function $\hat{\phi}_{Vx}(b_x, z)$ represents the error probability of a type-x message that is the output of a (hypothetical) noise-free variable node. Here we assume that the input to this variable node consists of: (i) the initial decoder input whose probability of error is $p_x^{(0)} = \gamma_{1-x} + (1-\gamma)\epsilon$, and (ii) $d_v - 1$ type-x messages whose probability of error is z each. The parameter b_x is the variable node voting threshold in the noisy Gallager B decoder, and is at least $\lceil \frac{d_v}{2} \rceil$ to guarantee a unique voting outcome. The optimal b_x minimizes the expression of $\hat{\phi}_{Vx}(b_x, z)$ for a given z. In the following, we compute the optimal value of b_x, and in particular we show that this optimal value is $\lceil \frac{d_v}{2} \rceil$ when $z < p_x^{(0)}$ for $x \in \{0,1\}$.

This derivation is similar to threshold derivations in [38,40]. The optimal value of b_x is the smallest integer b_x such that $\hat{\phi}_{Vx}(b_x, z)$ evaluated at b_x^* is less than or equal to $\hat{\phi}_{Vx}(b_x, z)$ evaluated at $b_x^* + 1$. The equation (2.12) implies that b_x^* is the minimum solution of the following inequality

$$\begin{aligned}\frac{1-(1-\gamma)\epsilon-\gamma_{1-x}}{(1-\gamma)\epsilon+\gamma_{1-x}} &\leq \frac{\binom{d_v-1}{b_x}(1-z)^{b_x} z^{d_v-1-b_x}}{\binom{d_v-1}{b_x} z^{b_x}(1-z)^{d_v-1-b_x}} \\ &= \left(\frac{1-z}{z}\right)^{2b_x-d_v+1}.\end{aligned}$$

Solving the preceding inequality, we find b_x^* as

$$b_x^* = \left\lceil \left(d_v - 1 + \frac{\log(\frac{1-p_x^{(0)}}{p_x^{(0)}})}{\log(\frac{1-z}{z})} \right) /2 \right\rceil. \tag{2.13}$$

Here we use the fact that $p_x^{(0)} = (1-\gamma)\epsilon + \gamma_{1-x}$. From Equation (2.13) if $\frac{\log(\frac{1-p_x^{(0)}}{p_x^{(0)}})}{\log(\frac{1-z}{z})} <$ 1 then $b_x^* = d_v/2$ for even d_v and $b_x^* = (d_v+1)/2$ for odd d_v. The inequality $\frac{\log(\frac{1-p_x^{(0)}}{p_x^{(0)}})}{\log(\frac{1-z}{z})} < 1$ holds if and only if $z < p_x^{(0)}$ holds.

Theorem 1. *Consider a codeword of relative weight w belonging to a (d_v, d_c)-regular LDPC code. Assume that the transmission and processing (both transient* *and permanent) errors are as specified in Section 2.3.1. Then, for $x \in \{0,1\}$, the initial (iteration $\ell = 0$) bit error rate of a type-x message is*

$$p_x^{(0)} = \gamma_{1-x} + (1-\gamma)\epsilon.$$

At iteration $\ell \geq 0$, let the type-x variable node voting threshold be $b_x^{(\ell)}$. We consider two cases: (a) For $0 < w < 1$ and $x \in \{0,1\}$, the bit error rates evolve recursively as

$$\begin{aligned} p_x^{(\ell+1)} &= \alpha_{1-x} + (1-\alpha)(\sigma_v + (1-2\sigma_v) \\ &\cdot \hat{\phi}_{Vx}(b_x^{(\ell)}, \beta_{1-x} + (1-\beta)(\sigma_c + (1-2\sigma_c)\hat{\phi}_{Cx}(p_0^{(\ell)}, p_1^{(\ell)})))). \end{aligned} \tag{2.14}$$

The average bit error rate, averaged over all codeword bits, at iteration ℓ is

$$p^{(\ell)} = (1-w)p_0^{(\ell)} + wp_1^{(\ell)}. \tag{2.15}$$

(b) For $w = 1$ and d_c even or for $w = 0$, the average bit error rate equals the bit error rate of the type-w messages, and the recursion simplifies to

$$\begin{aligned} p^{(\ell+1)} = p_w^{(\ell+1)} &= \alpha_{1-w} + (1-\alpha)(\sigma_v + (1-2\sigma_v) \\ &\cdot \hat{\phi}_{Vw}(b_w^{(\ell)}, \beta_{1-w} + (1-\beta)(\sigma_c + (1-2\sigma_c)\hat{\phi}_C(p_w^{(\ell)})))). \end{aligned}$$

Proof. In the following, we assume $x \in \{0,1\}$. Initially, at iteration $\ell = 0$, a type-x decoder input message is erroneous, i.e., does not equal x, if and only if (i) the message is stuck at $1-x$; or (ii) there is no stuck-at error but the corresponding channel output is erroneous. Therefore,

$$p_x^{(0)} = \gamma_{1-x} + (1-\gamma)\epsilon. \tag{2.16}$$

At iteration $\ell \geq 0$, observe that a hypothetical noise-free check-to-variable message is erroneous if and only if there is an odd number of erroneous messages among the $d_c - 1$ incoming variable-to-check messages. Meanwhile, due to the checksum constraint, a type-1 (type-0) check-to-variable message has to be produced from an odd (even) number of type-1 incoming messages. As the codelength $n \to \infty$, the pdf of the number of type-1 messages among the $d_c - 1$ incoming messages can be approximated by the Binomial$(d_c - 1, w)$ distribution where w is the relative codeword weight. Thus, the probability that a check node has an odd number of incoming type-1 messages is

$$\sum_{\substack{0 \le i \le d_c - 1 \\ i \text{ is odd}}} \binom{d_c - 1}{i} w^i (1-w)^{d_c - i - 1} = \frac{1}{2}(1 - (1-2w)^{d_c - 1}).$$

Meanwhile, the probability that an odd number of input messages are erroneous given that i of them are of type-1 and that the remaining $d_c - 1 - i$ of them are of type-0 is $\frac{1}{2}\big(1 - (1 - 2p_1^{(\ell)})^i (1 - 2p_0^{(\ell)})^{d_c - i - 1}\big)$. Then, for $0 < w < 1$, by the law of total probability, the error probability $\hat{q}_1^{(\ell)}$ of a hypothetical noise-free type-1 check-to-variable message at iteration ℓ, is then

$$\begin{aligned}\hat{q}_1^{(\ell)} &= \frac{1}{\frac{1}{2}(1 - (1-2w)^{d_c-1})} \sum_{\substack{0 \le i \le d_c - 1 \\ i \text{ is odd}}} \binom{d_c - 1}{i} w^i (1-w)^{d_c - i - 1} \\ &\cdot \frac{1}{2}\big(1 - (1 - 2p_1^{(\ell)})^i (1 - 2p_0^{(\ell)})^{d_c - i - 1}\big) = \hat{\phi}_{C1}(p_0^{(\ell)}, p_1^{(\ell)}). \end{aligned} \tag{2.17}$$

The expression for $\hat{\phi}_{C1}(p_0^{(\ell)}, p_1^{(\ell)})$ is provided in (2.10). The last equality is reached by applying the identity $\sum_{\substack{0 \le i \le k \\ i \text{ is odd}}} \binom{k}{i} a_1^i a_2^{k-i} = \frac{1}{2}((a_1 + a_2)^k - (a_2 - a_1)^k)$, setting $a_1 = w(1 - 2p_1^{(\ell)})$, $a_2 = (1-w)(1-2p_0^{(\ell)})$, and $k = d_c - 1$. Similarly, by applying the identity $\sum_{\substack{0 \le i \le k \\ i \text{ is even}}} \binom{k}{i} a_1^i a_2^{k-i} = \frac{1}{2}((a_1 + a_2)^k + (a_2 - a_1)^k)$, setting $a_1 = w(1 - 2p_1^{(\ell)})$, $a_2 = (1-w)(1-2p_0^{(\ell)})$, we have

$$\begin{aligned}\hat{q}_0^{(\ell)} &= \frac{1}{\frac{1}{2}(1 + (1-2w)^{d_c-1})} \sum_{\substack{0 \le i \le d_c - 1 \\ i \text{ is even}}} \binom{d_c - 1}{i} w^i (1-w)^{d_c - i - 1} \\ &\cdot \frac{1}{2}\big(1 - (1 - 2p_1^{(\ell)})^i (1 - 2p_0^{(\ell)})^{d_c - i - 1}\big) = \hat{\phi}_{C0}(p_0^{(\ell)}, p_1^{(\ell)}). \end{aligned} \tag{2.18}$$

Next, a type-x check-to-variable message is erroneous if and only if one of the following disjoint events occurs: (i) the message is stuck at $1 - x$; or (ii) there is no stuck-at error, but (iia) the hypothetical noise-free message is correct but it is flipped due to a transient error, or (iib) there is no transient error but the noise-free message itself is erroneous.

The error probability $q_x^{(\ell)}$ of a noisy type-x check-to-variable message is then

$$q_x^{(\ell)} = \beta_{1-x} + (1-\beta)(\sigma_c(1-\hat{q}_x^{(\ell)}) + (1-\sigma_c)\hat{q}_x^{(\ell)}). \tag{2.19}$$

Then, in the $(\ell+1)$st iteration, variable-to-check messages are calculated from the ℓth iteration check-to-variable messages. A hypothetical noise-free type-x variable-to-check message is erroneous if and only if one of the following two disjoint events occurs: (i) the decoder input is correct but at least $b_x^{(\ell)}$ of the $d_v - 1$ incoming messages (all of which are of type-x) are erroneous; or (ii) the decoder input is erroneous but at most $b_x^{(\ell)} - 1$ of the incoming messages are erroneous. Thus, given $q_x^{(\ell)}$, the error probability of a noise-free type-x variable-to-check message at iteration $\ell+1$, is

$$\hat{p}_x^{(\ell+1)} = (1-p_x^{(0)})\sum_{k=b_x^{(\ell)}}^{d_v-1}\binom{d_v-1}{k}(q_x^{(\ell)})^k(1-q_x^{(\ell)})^{d_v-1-k}$$
$$+ p_x^{(0)}\sum_{k=0}^{b_x^{(\ell)}-1}\binom{d_v-1}{k}(1-q_x^{(\ell)})^k(q_x^{(\ell)})^{d_v-1-k} = \hat{\phi}_{Vx}(b_x^{(\ell)}, q_x^{(\ell)}).$$

Finally, similar to the error rate derivation of check-to-variable messages as a function of the error rates of their hypothetical noise-free counterparts, we have

$$p_x^{(\ell+1)} = \alpha_{1-x} + (1-\alpha)(\sigma_v(1-\hat{p}_x^{(\ell)}) + (1-\sigma_v)\hat{p}_x^{(\ell)}). \tag{2.20}$$

By sequentially substituting $\hat{p}_x^{(\ell+1)}$, $q_x^{(\ell)}$, $\hat{q}_x^{(\ell)}$, and $p_x^{(0)}$ into (2.20), we arrive at (2.14). The average bit error rate $p^{(\ell)}$ is then the weighted sum of $p_0^{(\ell)}$ and $p_1^{(\ell)}$, with the former weighted by $1-w$ and the latter weighted by w. The expression for $p^{(\ell)}$ is given in (2.15).

For $w = 1$ and for $w = 0$, as noted previously, with probability 1, each message is of type-1 and type-0, respectively. Hence, density evolution is respectively on type-1 messages only and on type-0 messages only. Furthermore, for $w = 1$,

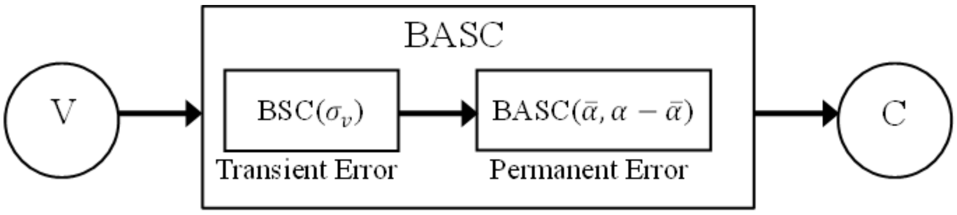


Figure 2.2: Decoder model for approximate density evolution. Variable node and check node in the diagram are hypothetically noise-free nodes.

if d_c is odd, with probability 1, the checksum constraints cannot be fulfilled, meaning that with probability 1 a codeword of relative weight w is not a valid codeword. Therefore, for $w = 1$, only the case where d_c is even is considered. Since the derivations of the $w = 1$ with even d_c case and the $w = 0$ case follow the procedures described above for the $w \in (0, 1)$ case, the details are omitted. □

Theorem 1 evolves on the bit error rate pair $(p_0^{(\ell)}, p_1^{(\ell)})$. In the following, we present a simpler density evolution that evolves on $p_{\text{app}}^{(\ell)}$, an approximation of the average bit error rate $p^{(\ell)}$. This simpler density evolution is derived via a suitable decoder approximation, which we describe next.

First, let us use BASC(z_0, z_1) to denote a binary asymmetric channel (BASC) with the 0-to-1 cross-over probability z_0 and the 1-to-0 cross-over probability z_1. We also let BSC(σ) denote the standard binary symmetric channel with cross-over probability σ. In our approximation, the two message types are replaced by an aggregate message. We capture the processing-noise-induced transformation of a message as having passed the message through a certain BASC channel. For the variable-to-check message, this BASC channel is viewed as a concatenation of two channels: (a) the BSC(σ_v) modeling the effects of transient errors at the output of the variable node, and (b) the BASC$(\bar{\alpha}_w,\, \alpha - \bar{\alpha}_w)$ modeling the effects of permanent errors. Here, the parameter $\bar{\alpha}_w = w\alpha_0 + (1 - w)\alpha_1$ "averages" the permanent error rates associated with different stored values. To be specific, the channel described in (b) takes the relative weight w as a model parameter. Passing the all-zero codeword through this channel yields an output in which the probability of a bit having value 1 is the same as the probability of a bit having value 1 when a codeword of relative weight w is passed through a channel that is realized as a stuck-at-0 channel with probability α_0 and stuck-at-1 channel with probability α_1. The concatenated channel is equivalent to a BASC$(\sigma_v(1-\alpha)+\bar{\alpha}_w, \sigma_v(1-\alpha)+(\alpha-\bar{\alpha}_w))$. This model is illustrated in Fig. 2.2.

Introducing w directly into the parameters of the constituent channels permits us to move the averaging operation away from the message types. As a result, we consider the evolution of the bit error rate $p_{\text{app}}^{(\ell)}$ of this system assuming the all-zero codeword is transmitted. While it is clear that this system provides only an approximation of the original system, as we shall see later, the resultant bit error rate $p_{\text{app}}^{(\ell)}$ is in fact close to the average bit error rate $p^{(\ell)}$ derived from the exact density evolution.

Before stating the density evolution equations under the decoder approximation model, we first define

$$\hat{\phi}_V(b,z) = (1-(1-\gamma)\epsilon-\bar{\gamma}_w)\sum_{k=b}^{d_v-1}\binom{d_v-1}{k}z^k(1-z)^{d_v-1-k}$$
$$+((1-\gamma)\epsilon+\bar{\gamma}_w)\sum_{k=0}^{b-1}\binom{d_v-1}{k}(1-z)^k z^{d_v-1-k},$$

as a function of $z \in [0,1]$.

The function $\hat{\phi}_V(b,z)$ represents the error probability of the output of a (hypothetical) noise-free variable node. The error probability of the output of a (hypothetical) noise-free check node is $\hat{\phi}_C(z)$, as defined in (2.11). With $\hat{\phi}_V(b,z)$ and $\hat{\phi}_C(z)$, we are ready to define the density evolution equations for the decoder approximation model. Initially, based on the characterization of the BASC associated with the decoder input, we have

$$p_{\text{app}}^{(0)} = (1-\gamma)\epsilon + \bar{\gamma}_w. \qquad (2.21)$$

For $\ell \geq 0$, the recursive expression for $p_{\text{app}}^{(\ell+1)}$ readily becomes

$$p_{\text{app}}^{(\ell+1)} = \bar{\alpha}_w + (1-\alpha)(\sigma_v + (1-2\sigma_v) \\ \cdot \hat{\phi}_V(b^{(\ell)}, \bar{\beta}_w + (1-\beta)(\sigma_c + (1-2\sigma_c)\hat{\phi}_C(p_{\text{app}}^{(\ell)})))). \tag{2.22}$$

Before stating Claim 1 on the simpler density evolution, we first define the approximation operator $\approx$ as used in the context of this chapter.

Definition 1. *The approximation operator* $\approx$ *denotes the equivalence of two functions* $f_1(p_0^{(\ell)}, p_1^{(\ell)}, |\beta_1-\beta_0|, |\gamma_1-\gamma_0|, \varsigma(w))$ *and* $f_2(p_0^{(\ell)}, p_1^{(\ell)}, |\beta_1-\beta_0|, |\gamma_1-\gamma_0|, \varsigma(w))$ *up to the linear order of* $p_0^{(\ell)}, p_1^{(\ell)}, |\beta_1-\beta_0|, |\gamma_1-\gamma_0|$, *and* $\varsigma(w)$ *terms, where* $\varsigma(w)$ *is any one of the following three terms: (i)* $(1-2w)^{d_c-2}$ *(ii)* w *(iii)* $(1-w)$.

We then have the following Claim 1.

Claim 1. *For* $\ell \geq 0$, $p^{(\ell)} \approx p_{\text{app}}^{(\ell)}$, *provided that* $p_x^{(\ell)}$,$|\beta_1-\beta_0|$, $|\gamma_1-\gamma_0|$, *and at least one of the following three quantities (i)* $(1-2w)^{d_c-2}$, *(ii)* w, *and (iii)* $(1-w)$ *is small.*

Proof. We prove Claim 1 by induction. The approximation operator $\approx$ is as defined in Definition 1.

We first consider the $\ell = 0$ case. By (2.15) and (2.16), we have

$$p^{(0)} = w(\gamma_1 + (1-\gamma)\epsilon) + (1-w)(\gamma_0 + (1-\gamma)\epsilon) = \bar{\gamma}_w + (1-\gamma)\epsilon. \tag{2.23}$$

Thus, according to (2.21), we immediately have $p^{(0)} \approx p_{\text{app}}^{(0)}$.

Assume $p^{(\ell)} \approx p_{\text{app}}^{(\ell)}$ is true for some $\ell, \ell \geq 0$. We are going to show that $p^{(\ell+1)} \approx p_{\text{app}}^{(\ell+1)}$. Given that $p_0^{(\ell)}$ and $p_1^{(\ell)}$ are small, we approximate $\hat{\phi}_{C0}(p_0^{(\ell)}, p_1^{(\ell)})$ (the expression in (2.10)) by its first order Taylor expansion around the point $(p_0^{(\ell)}, p_1^{(\ell)}) = (0,0)$:

$$\hat{q}_0^{(\ell)} \approx \frac{p_1^{(\ell)} w(d_c-1)(1-(1-2w)^{d_c-2})}{(1+(1-2w)^{d_c-1})} \\ + \frac{p_0^{(\ell)}(1-w)(d_c-1)(1+(1-2w)^{d_c-2})}{(1+(1-2w)^{d_c-1})}. \tag{2.24}$$

First, suppose that $(1-2w)^{dc-2}$ is small. By using the fact that $\frac{1}{1+\tau} \approx 1-\tau$ and $1-k\tau \approx (1-\tau)^k$ for small τ, we further have

$$\begin{aligned}
\hat{q}_0^{(\ell)} &\approx \frac{1}{2} - \big[\frac{1}{2} - p_1^{(\ell)} w(d_c-1)(1-(1-2w)^{d_c-2})(1-(1-2w)^{d_c-1}) \\
&- p_0^{(\ell)}(1-w)(d_c-1)(1+(1-2w)^{d_c-2})(1-(1-2w)^{d_c-1})\big] \\
&\approx \frac{1}{2} - \frac{1}{2}\left[1-2(d_c-1)p^{(\ell)}\right] \approx \frac{1}{2} - \frac{1}{2}(1-2p^{(\ell)})^{d_c-1} = \hat{\phi}_C(p^{(\ell)}). \qquad (2.25)
\end{aligned}$$

Next, consider the case where w is small. Starting from (2.24) and using the fact that $\frac{1}{1+\tau} \approx 1-\tau$ and $1-k\tau \approx (1-\tau)^k$ for small τ, we have

$$\begin{aligned}
\hat{q}_0^{(\ell)} &\approx \frac{p_1^{(\ell)} w(d_c-1)(1-(1-2w)^{d_c-2})}{(1+(1-2w)^{d_c-1})} \\
&+ \frac{p_0^{(\ell)}(1-w)(d_c-1)(1+(1-2w)^{d_c-2})}{(1+(1-2w)^{d_c-1})} \\
&\approx \frac{p_0^{(\ell)}(1-w)(d_c-1)(1+(1-2w)^{d_c-2})}{(1+(1-2w)^{d_c-1})} \\
&\approx \frac{p_0^{(\ell)}(1-w)(d_c-1)(2-2(d_c-2)w))}{(2-2(d_c-1)w)} \\
&\approx p_0^{(\ell)}(1-w)(d_c-1)(1-(d_c-2)w)(1+(d_c-1)w) \\
&\approx p_0^{(\ell)}(d_c-1) \approx \frac{1}{2} - \frac{1}{2}(1-p_0^{(\ell)})^{d_c-1} = \hat{\phi}_C(p^{(\ell)}). \qquad (2.26)
\end{aligned}$$

Similarly, we have that when $1-w$ is small, $\hat{q}_0^{(\ell)} \approx \hat{\phi}_C(p^{(\ell)})$. Following similar procedures, we also have $\hat{q}_1^{(\ell)} \approx \hat{\phi}_C(p^{(\ell)})$. Hence we can derive an approximation of (2.19) as

$$q_x^{(\ell)} \approx \beta_{1-x} + (1-\beta)(\sigma_c + (1-2\sigma_c)\hat{\phi}_C(p^{(\ell)})). \qquad (2.27)$$

Next, we consider the approximation at the variable node. For simplicity, we only consider the case where $b_1^{(\ell)}$ and $b_0^{(\ell)}$ are equal and define $b^{(\ell)} = b_1^{(\ell)} = b_0^{(\ell)}$. Let the average bit error rate of check-to-variable messages be $q^{(\ell)}$. The quantity $q^{(\ell)}$ is then the sum of $q_0^{(\ell)}$ and $q_1^{(\ell)}$ weighted by $1-w$ and w,

$$q^{(\ell)} = wq_1^{(\ell)} + (1-w)q_0^{(\ell)}. \qquad (2.28)$$

Also let

$$\hat{\psi}(\zeta, z) = (1-(1-\gamma)\epsilon-\zeta)\sum_{k=b^{(\ell)}}^{d_v-1}\binom{d_v-1}{k}z^k(1-z)^{d_v-1-k}$$
$$+((1-\gamma)\epsilon+\zeta)\sum_{k=0}^{b^{(\ell)}}\binom{d_v-1}{k}(1-z)^k z^{d_v-1-k} \tag{2.29}$$

be a function of $\zeta \in [0,1]$ and $z \in [0,1]$. Then, we have $\hat{\phi}_{V0}(b^{(\ell)}, q_0^{(\ell)}) = \hat{\psi}(\gamma_1, q_0^{(\ell)})$ and $\hat{\phi}_{V1}(b^{(\ell)}, q_1^{(\ell)}) = \hat{\psi}(\gamma_0, q_1^{(\ell)})$. Given that $|\gamma_1-\gamma_0|$ and $|\beta_1-\beta_0|$ are small, for $x \in \{0,1\}$, $|\gamma_x-\bar{\gamma}_w|$ and $|q_x^{(\ell)}-q^{(\ell)}|$ are small. Note that $\hat{\psi}(\zeta, z)$ is a polynomial of degree no higher than d_v, and so its second order partial derivatives are bounded. Hence we approximate $\hat{\psi}(\gamma_0, q_1^{(\ell)})$ and $\hat{\psi}(\gamma_1, q_0^{(\ell)})$ by their first order Taylor expansions around the point $(\bar{\gamma}_w, q^{(\ell)})$, which is also expressed as $w(\gamma_0, q_1^{(\ell)})+(1-w)(\gamma_1, q_0^{(\ell)})$. Then, we have

$$w\hat{\psi}(\gamma_0, q_1^{(\ell)}) + (1-w)\hat{\psi}(\gamma_1, q_0^{(\ell)}) \approx \hat{\psi}(\bar{\gamma}_w, q^{(\ell)}) = \hat{\phi}_V(b^{(\ell)}, q^{(\ell)}). \tag{2.30}$$

Using (2.20) and (2.30), the expression for the average bit error rate $p^{(\ell+1)}$ becomes

$$\begin{aligned}
p^{(\ell+1)} &= wp_1^{(\ell+1)} + (1-w)p_0^{(\ell+1)} \\
&\approx \bar{\alpha}_w + (1-\alpha)\big[\sigma_v + (1-2\sigma_v)\hat{\phi}_V(b^{(\ell)}, q^{(\ell)})\big]. \\
&\approx \bar{\alpha}_w + (1-\alpha)(\sigma_v + (1-2\sigma_v) \\
&\cdot \hat{\phi}_V(b^{(\ell)}, \bar{\beta}_w + (1-\beta)(\sigma_c + (1-2\sigma_c)\hat{\phi}_C(p^{(\ell)})))) \\
&\approx \bar{\alpha}_w + (1-\alpha)(\sigma_v + (1-2\sigma_v) \\
&\cdot \hat{\phi}_V(b^{(\ell)}, \bar{\beta}_w + (1-\beta)(\sigma_c + (1-2\sigma_c)\hat{\phi}_C(p_{\text{app}}^{(\ell)})))) \\
&= p_{\text{app}}^{(\ell+1)}.
\end{aligned} \tag{2.31}$$

In the second approximation we use (2.27) and (2.28), and in the third approximation we use the inductive hypothesis. Hence we have shown that $p^{(\ell+1)} \approx p_{\text{app}}^{(\ell+1)}$, thus completing the proof. □

We assume that the noisy Gallager B decoder successfully reduces the bit error rate of the messages in each iteration, so that there exists an iteration index ℓ_0, $\ell_0 \geq 1$, such that for all $\ell, \ell \geq \ell_0$, both $q_x^{(\ell)}$ and $p_x^{(\ell)}$ are smaller than $p_x^{(0)}$. Then, the variable node voting threshold $b_x^{(\ell)}$ is equal to $\lceil \frac{d_v}{2} \rceil$. Hence, in the remainder of the discussion we will simply assume that $\hat{\phi}_{Vx}(b_x, z) = \hat{\phi}_{Vx}(\lceil \frac{d_v}{2} \rceil, z)$ and $\hat{\phi}_V(b, z) = \hat{\phi}_V(\lceil \frac{d_v}{2} \rceil, z)$.

Based on the density evolution equations (2.21) and (2.22), the following Theorem 2 gives the residual error rate of the noisy Gallager B decoder for a transmitted codeword of relative weight w.

Theorem 2. *Assume that $p_x^{(0)}$, $|\beta_1 - \beta_0|$, $|\gamma_1 - \gamma_0|$, α, β, σ_v, σ_c, and at least one of the following three quantities (i) $(1-2w)^{d_c-2}$, (ii) w, and (iii) $(1-w)$ is small. Suppose that the optimal variable node threshold is $\lceil \frac{d_v}{2} \rceil$ for all iterations. Then, the residual error rate p_r of the Gallager B decoder subject to both transient and permanent errors (as specified in Section 2.3.1) is*

$$p_r \approx \begin{cases} \frac{\bar{\alpha}_w + \sigma_v + 2(\bar{\gamma}_w + (1-\gamma)\epsilon)(\bar{\beta}_w + \sigma_c)}{1 - 2(d_c-1)(\bar{\gamma}_w + (1-\gamma)\epsilon)} & d_v = 3, \\ \bar{\alpha}_w + \sigma_v & d_v > 3. \end{cases}$$

Proof. The residual error rate is derived by finding a fixed point p_r of the recursive equation (2.22) assuming that p_r is small,

$$\begin{aligned} p_r = {} & \bar{\alpha}_w + (1-\alpha)(\sigma_v + (1-2\sigma_v) \\ & \cdot \hat{\phi}_V(\lceil d_v/2 \rceil, \bar{\beta}_w + (1-\beta)(\sigma_c + (1-2\sigma_c)\hat{\phi}_C(p_r)))). \end{aligned} \tag{2.32}$$

For ℓ large enough, $p_{\text{app}}^{(\ell)}$ is small enough since this error is monotonically decreasing with ℓ. Then, for sufficiently large ℓ, $\hat{\phi}_C(p_r)$ can be approximated by $(d_c-1)p_r$. By dropping the cross-terms, we have

$$\begin{aligned} p_r \approx {} & \bar{\alpha}_w + (1-\alpha)(\sigma_v + (1-2\sigma_v) \\ & \cdot \hat{\phi}_V(\lceil d_v/2 \rceil, \bar{\beta}_w + \sigma_c + (d_c-1)p_r)). \end{aligned} \tag{2.33}$$

We first consider the $d_v = 3$ case. Under the given assumptions (and in particular by dropping cross-terms), $\hat{\phi}_V(2, \bar{\beta}_w + \sigma_c + (d_c - 1)p_r)$ can be approximated as $2(\bar{\gamma}_w + (1-\gamma)\epsilon)(\bar{\beta}_w + \sigma_c + (d_c - 1)p_r)$. Then, by dropping the cross-terms of σ_v, α, σ_c, β, and p_r, (2.33) becomes

$$p_r \approx \bar{\alpha}_w + \sigma_v + 2(\bar{\gamma}_w + (1-\gamma)\epsilon)(\bar{\beta}_w + \sigma_c + (d_c - 1)p_r)), \tag{2.34}$$

and the expression for p_r follows immediately.

For $d_v > 3$, $\hat{\phi}_V(\lceil d_v/2 \rceil, \bar{\beta}_w + \sigma_c + (d_c - 1)p_e)$ does not have any linear terms. Hence, we have

$$p_r \approx \bar{\alpha}_w + \sigma_v. \tag{2.35}$$

□

We provide in the following a brief discussion of decoding threshold for noisy Gallager B decoder. Since for noisy decoders, $\lim_{\ell\to\infty} p^{(\ell)} > 0$, we use the modified definition of the decoding threshold as given in [36]. For a target error rate η, the decoding threshold ϵ^* is defined as

$$\epsilon^*(\eta) = \sup\{\epsilon : \lim_{\ell\to\infty} p^{(\ell)} < \eta\}. \tag{2.36}$$

Following the derivation of Lemma 4 and Section II.D in [38] (taking the first-order partial derivatives of the iterative expression with respect to $p_{\text{app}}^{(\ell)}$ and deriving the conditions for the derivatives to be positive), one can establish the monotonicity of the iterative expression (7) (as a function of $p_{\text{app}}^{(\ell)}$) and thus the convergence of the $p_{\text{app}}^{(\ell)}$ sequence. A set of sufficient conditions for this function to be monotonic is that both the transient error rates and the message error rates in variable nodes and check nodes are less than $\frac{1}{2}$, and that the voting threshold is $\lceil \frac{d_v}{2} \rceil$.

Assuming the aforementioned conditions for convergence are satisfied, the threshold $\epsilon^*(\eta)$, i.e., the maximum value of ϵ to achieve residual error rate η, is derived as the value of ϵ satisfying $p_{\text{app}}^{(\ell+1)} = p_{\text{app}}^{(\ell)} = \eta$:

$$\epsilon^*(\eta) = \frac{\frac{\eta - \bar{\alpha}_w}{(1-\alpha)(1-2\sigma_v)} - \frac{\sigma_v}{1-2\sigma_v} - \Theta_1(\eta) - \bar{\gamma}_w(\Theta_2(\eta) - \Theta_1(\eta))}{(1-\gamma)(\Theta_2(\eta) - \Theta_1(\eta))} \tag{2.37}$$

Target residual error ($\times 10^{-3}$)	Decoding threshold
1.60	unachievable
1.61	1.5×10^{-4}
1.62	6.3×10^{-4}
1.63	1.1×10^{-3}
1.64	1.6×10^{-3}
1.65	2.0×10^{-3}
1.66	2.5×10^{-3}
1.67	2.9×10^{-3}
1.68	3.4×10^{-3}
1.69	3.8×10^{-3}
1.70	4.3×10^{-3}

Table 2.1: Decoding thresholds for different target residual error rates

where $\Theta_1(\eta) = \sum_{k=b}^{d_v-1} \binom{d_v-1}{k} (z(\eta))^k (1 - z(\eta))^{d_v-1-k}$, $\Theta_2(\eta) = \sum_{k=0}^{b-1} \binom{d_v-1}{k} (1 - z(\eta))^k (z(\eta))^{d_v-1-k}$, and $z(\eta) = \bar{\beta}_w + (1-\beta)(\sigma_c + (1-2\sigma_c)\hat{\phi}_C(\eta))$.

The decoding threshold of Gallager B decoder for a $(3,6)$-regular LDPC code is 0.04 for a binary symmetric channel [39]. In Table 2.1, we consider a range of target residual error rates under permanent error rates $\alpha_0 + \alpha_1 = \beta_0 + \beta_1 = \gamma_0 + \gamma_1 = 2 \times 10^{-3}$ (60:40 for the ratio of stuck-at-1 and stuck-at-0 errors) and transient error rates $\sigma_v = \sigma_c = 5 \times 10^{-4}$, and show the corresponding decoding thresholds.

Theorem 2 shows that when permanent errors are present, the effect of transient errors on the residual error rate is reminiscent of the transient error effects derived in [38], which considered the Gallager B decoder with transient errors only. We can also observe that in (2.35), both the transient error rate and the permanent error rate contribute to linear terms in the residual error rate. In the following section, to enhance the robustness of our noisy decoder, we propose a hardware error detection-and-correction scheme that combats both permanent and transient errors.

2.3.3 Hardware Error Detection and Correction

In this section, as an alternative to BIST and BISR hardware error test and repair technologies [43], we propose and evaluate a scheme that not only detects memory cells with permanent errors, but also corrects the decoder output to further reduce the residual error rate. As observed from the expressions in Theorem 2, the transient errors at variable nodes and the permanent errors in (the memory cells storing) the variable-to-check messages are the two major contributors to the residual errors associated with our decoder. With our proposed schemes, we are able to locate the permanent errors, prevent the detected defective cells from future use, and consequently reduce the residual error rate of the decoder output in subsequent transmissions. Furthermore, we can also correct the detected residual errors arising from both transient and permanent errors. For convenience, in this section, the scope of the term "permanent error" is restricted to the permanent errors in variable-to-check messages.

We first classify the permanent errors into two categories, *perceptible* errors and *imperceptible* errors, depending on whether the stuck-at value is different from the associated variable node bit value or not. We note that the imperceptible errors cannot be identified because the stuck-at value coincides with the true value. However, since the positions of 0s and 1s may change from one codeword transmission to the next, an imperceptible error may become perceptible in subsequent transmissions (likewise perceptible error may become imperceptible; our detection scheme aims to identify perceptible errors before they become imperceptible).

We design a hardware error detection-and-correction scheme that is performed in rounds. Each round starts with the transmission of a new codeword, followed by a decoding phase, a detection phase in which memory cells with perceptible permanent errors are detected and replaced with backup cells, and finishes with an additional residual error correction phase. The undetected perceptible errors and the imperceptible errors constitute the *undiscovered errors* of the current round. They remain as the major contributor of the residual errors of the next decoding phase, and are left to be detected and eliminated in the next detection and residual error correction phases.

2.3.3.1 Permanent Error Detection

Next, we describe the steps performed in the permanent error detection phase of each round.

Let $\{m_{v,c}^{(\ell)}\}$ and $\{m_{c,v}^{(\ell-1)}\}$, respectively, be the collection of variable-to-check messages and the collection of check-to-variable messages at the final iteration, say iteration ℓ, of the decoding phase. We assume that ℓ is large enough so that the empirical residual decoding error is close enough to the residual error rate specified in Theorem 2.

Let Γ denote the set of error candidates. At the beginning of each round, we initialize the set Γ to be the empty set. To start the detection phase, each variable node v selects one of its d_v outgoing messages (given by the collection $\{m_{v,c}^{(\ell)}\}$), referred to as θ_v, and sends θ_v to *all* of its neighboring check nodes. Each check node then XORs all the d_c messages coming from all of its d_c variable node neighbors. If, for a variable node v, none of its neighboring check nodes is satisfied, the location of the memory cell storing θ_v is recorded in the set Γ of error candidates. The procedure is repeated d_v times. At each time, each of the n variable nodes selects a new message among its d_v outgoing messages, and the set Γ is expanded accordingly.

Note that Γ contains both transient and perceptible permanent errors. Observe that the probability of two consecutive erroneous computations due to transient errors is small provided that the transient error rate is small. Hence, repeating selected computations can suppress the effects of transient errors. We then perform the following procedures to find the cells experiencing perceptible permanent errors in Γ.

We take $\{m_{c,v}^{(\ell-1)}\}$ as the input, and repeat the computations at the variable nodes. Each newly computed variable-to-check message overwrites $m_{v,c}^{(\ell)}$, and reads as $\breve{m}_{v,c}^{(\ell)}$ when accessed from the memory cell. Note that $\{\breve{m}_{v,c}^{(\ell)}\}$ and $\{m_{v,c}^{(\ell)}\}$ are the same except for the messages that only experience transient errors in exactly one of the two message collections. We then repeat the same procedures previously performed on $\{m_{v,c}^{(\ell)}\}$: based on $\breve{m}_{v,c}^{(\ell)}$ we identify variable nodes with all checks unsatisfied, and derive the set $\breve{\Gamma}$ from $\{\breve{m}_{v,c}^{(\ell)}\}$. Finally, the memory cells in $\Gamma \cap \breve{\Gamma}$ are labeled as having permanent errors, and are replaced. By taking the intersection of Γ and $\breve{\Gamma}$, the risk of mislabeling a transient error as a permanent error is reduced.

The detection process requires $2d_v$ additional iterations after the regular Gallager B decoding process. This overhead could be too large in some high-speed communication systems. To reduce the overhead, instead of repeating the detection scheme d_v times (since we perform the detection scheme for both $m_{v,c}^{(\ell)}$ and $\breve{m}_{v,c}^{(\ell)}$, we have $2d_v$ additional iterations in total) to examine every memory cells in variable nodes in each codeword transmission, we may repeat the scheme only J $(1 \leq J < d_v)$ times to examine J memory cells in each variable node. The decoder output is then derived only from J memory cells that have been examined. In the next transmission, we may examine another J memory cells and so on.

2.3.3.2 Residual Error Correction

The permanent error detection phase is followed by an error correction phase to further reduce the residual error rate. Error correction is performed by flipping the values of messages $\{\breve{m}_{v,c}^{(\ell)}\}$ stored in memory cells in $\breve{\Gamma}$. Note that for messages stored in memory cells in $\Gamma \cap \breve{\Gamma}$, which have been labeled as having permanent errors, the flipping of stored message values is conveniently accomplished at memory cell replacement. Hence, in the error correction phase, we only need to flip the message values stored in $\breve{\Gamma} \backslash \Gamma$. The final decoder output is then derived from the set of messages in $\{\breve{m}_{v,c}^{(\ell)}\}$ after the flipping. Since most of the memory cells in $\breve{\Gamma}$ store erroneous messages (residual errors)[1], the probability of generating new errors by flipping the bits stored in the memory cells in $\breve{\Gamma}$ is small.

The action performed in the permanent error detection and the residual error correction phases is summarized below.

1. Derive the error candidate set Γ from the message collection $\{m_{v,c}^{(\ell)}\}$.
2. Compute $\{\breve{m}_{v,c}^{(\ell)}\}$ from the message collection $\{m_{c,v}^{(\ell-1)}\}$.
3. Derive $\breve{\Gamma}$ from the message collection $\{\breve{m}_{v,c}^{(\ell)}\}$.
4. Label the memory cells in $\Gamma \cap \breve{\Gamma}$ as having permanent errors, replace the labeled cells, and write into each replacement cell the complement of the value previously stored in the corresponding replaced cell.
5. Flip the values of messages $\{\breve{m}_{v,c}^{(\ell)}\}$ stored in $\breve{\Gamma} \backslash \Gamma$.

2.3.3.3 Performance Analysis

It remains to evaluate the permanent error detection and residual error correction capability of the proposed scheme. We first characterize the performance of

[1]The probability that a memory cell in $\breve{\Gamma}$ stores a correct message is $\big((1-2\sigma_c)\hat{\phi}_C(p_r)+\sigma_c\big)^{d_v}$, which is small when p_r and σ_c are small.

residual error correction, and then the performance of permanent error detection.

We first analyze the residual error correction performance. Consider a message in $\{\breve{m}_{v,c}^{(\ell)}\}$ which experiences either a perceptible permanent error or a transient error. Suppose that the variable node this message is associated with is v^*. The error will not be corrected if at least one check node neighbor of variable node v^* is satisfied. Thus, the probability that a residual error is not detected by our correction scheme is

$$P_{nc}(p_r) = 1 - \Big[1 - \big((1-2\sigma_c)\hat{\phi}_C(p_r) + \sigma_c\big)\Big]^{d_v}, \tag{2.38}$$

where p_r is the residual error rate approximately derived in Theorem 3. It is interesting to observe that (2.38) is reminiscent of the Gallager A decoder error expressions with suitable reparameterization of the exponent and without the terms containing the decoder input.

Next, we analyze the permanent error detection. We measure the permanent error detection performance by the undiscovered error rate $P_{\text{ud},k}$ at the end of the detection phase in round k. We first derive $P_{\text{ud},k}$ as a function of $P_M(\rho_k)$, where $P_M(\rho_k)$ is the probability that a perceptible permanent error remains undetected after the detection phase in round k, and ρ_k is the perceptible permanent error rate during round k. Then, we derive the expression of $P_M(\rho_k)$.

The following lemma characterizes the undiscovered error rate $P_{\text{ud},k}$ at the end of round k.

Lemma 1. *The undiscovered stuck-at-*0 *and stuck-at-*1 *error rates at the end of round* 0 *are* $P_{\text{ud},0,0} = \alpha_0$ *and* $P_{\text{ud},0,1} = \alpha_1$*, respectively. The undiscovered stuck-at-*0 *error rate at the end of round k ($k \geq 1$) is* $P_{\text{ud},k,0} = P_{\text{ud},k-1,0}[w_k P_M(\rho_k) + (1-w_k)]$*, and the undiscovered stuck-at-*1 *error rate at the end of round k ($k \geq 1$) is* $P_{\text{ud},k,1} = P_{\text{ud},k-1,1}[(1-w_k)P_M(\rho_k) + w_k]$*, where* $\rho_k = w_k P_{\text{ud},k-1,0} + (1-w_k)P_{\text{ud},k-1,1}$ *is the perceptible permanent error rate at round k, and where w_k is the relative weight of the kth transmitted codeword.*

Proof. Initially, $P_{\text{ud},0,0} = \alpha_0$ and $P_{\text{ud},0,1} = \alpha_1$ by definition. Given that the relative weight of the kth transmitted codeword is w_k, the perceptible stuck-at-0 and stuck-at-1 permanent error rates are $w_k P_{\text{ud},k-1,0}$ and $(1-w_k)P_{\text{ud},k-1,1}$, respectively, and the total perceptible permanent error rate is $\rho_k = w_k P_{\text{ud},k-1,0} + (1 - w_k)P_{\text{ud},k-1,1}$. Similarly, the imperceptible stuck-at-0 and stuck-at-1 error rates are $(1 - w_k)P_{\text{ud},k-1,0}$ and $w_k P_{\text{ud},k-1,1}$, respectively.

Recall that the undiscovered errors consist of the imperceptible errors and the undetected perceptible errors. Hence, at the end of round k, given that the probability of undetected perceptible stuck-at-0 and perceptible stuck-at-1 errors are both $P_M(\rho_k)$, we have

$$P_{\text{ud},k,0} = (1 - w_k)P_{\text{ud},k-1,0} + P_M(\rho_k) \cdot w_k P_{\text{ud},k-1,0},$$

and

$$P_{\text{ud},k,1} = w_k P_{\text{ud},k-1,1} + P_M(\rho_k) \cdot (1 - w_k)P_{\text{ud},k-1,1}.$$

□

Lemma 1 enables us to analytically compute the undiscovered permanent error rate $P_{\text{ud},k}$, simply viewed as $P_{\text{ud},k,0} + P_{\text{ud},k,1}$. However, since the error rate is a function of the relative weight of the transmitted codeword, we may have different undiscovered permanent error rates for different codeword weights. This complicates the error detection performance characterization. Fortunately, as shown in the following Theorem 3, for sufficiently large block lengths, the undiscovered permanent error rate is close to the case where all the transmitted codewords are typical (i.e., with relative weight $w = \frac{1}{2}$) with high probability.

Theorem 3. *For an arbitrarily small positive ϵ, when the codeword length is sufficiently large, we have*

$$\begin{aligned} &Pr\left\{|P_{\text{ud},k} - \bar{P}_{\text{ud},k}| \leq \frac{\alpha}{2^k}\left[(1+\epsilon)^k(1 + P_M(\alpha))^k - 1\right]\right\} \\ &\geq (1-\epsilon)^k, \end{aligned} \tag{2.39}$$

where $\bar{P}_{\mathrm{ud},k}$ is the undiscovered error rate when all the k transmitted codewords have relative weight $\frac{1}{2}$ (typical codeword).

Proof. For convenience, we use $[k]$ to denote the set $\{1, 2, \ldots, k\}$. By the LDPC codeword typicality (Chapter 5.2 in [44]), for arbitrarily small ϵ, with large enough codelength, we have

$$Pr\left\{|w_i - \frac{1}{2}| \leq \frac{\epsilon}{2}\right\} \geq 1 - \epsilon \tag{2.40}$$

for the relative weight w_i of the ith transmitted codeword, $\forall i \in [k]$. Thus, to prove (2.39), it suffices to show that when $|w_i - \frac{1}{2}| \leq \frac{\epsilon}{2}, \forall i \in [k]$, it is guaranteed that $|P_{\mathrm{ud},k} - \bar{P}_{\mathrm{ud},k}| \leq \frac{\alpha}{2^k}\left[(1+\epsilon)^k(1+P_M(\alpha))^k - 1\right]$. We prove this next.

We recall from Lemma 1 that

$$\begin{aligned} P_{\mathrm{ud},k} = \alpha_0 \prod_{i=1}^{k} \left[w_i P_M(\rho_i) + (1 - w_i)\right] \\ + \alpha_1 \prod_{i=1}^{k} \left[(1 - w_i) P_M(\rho_i) + w_i\right]. \end{aligned} \tag{2.41}$$

For $w_i = \frac{1}{2}, \forall i \in [k]$, we have

$$\bar{P}_{\mathrm{ud},k} = \frac{\alpha}{2^k} \prod_{i=1}^{k} (P_M(\bar{\rho}_i) + 1), \tag{2.42}$$

where $\bar{\rho}_i$ denotes the perceptible permanent error rate at the beginning of the detection phase of round i.

We decompose $P_{\mathrm{ud},k}$ into a sum of $k+1$ terms, and establish the desired upper bound on the difference $|P_{\mathrm{ud},k} - \bar{P}_{\mathrm{ud},k}|$ by establishing the upper bounds on the constituent terms. Let

$$Y_k = \alpha_0 \prod_{i=1}^{k} (1 - w_i) + \alpha_1 \prod_{i=1}^{k} w_i.$$

For $j \in [k]$, let

$$X_{k,j} = \alpha_0 \sum_{J \in \Omega_j^k} \left[\prod_{h \in J} (w_h P_M(\rho_h)) \prod_{m \in [k] \setminus J} (1 - w_m) \right]$$
$$+ \alpha_1 \sum_{J \in \Omega_j^k} \left[\prod_{h \in J} (1 - w_h) P_M(\rho_h) \prod_{m \in [k] \setminus J} w_m \right],$$

where the set Ω_j^k is the set of all j-element subsets of $[k]$. Also, we let

$$\bar{Y}_k = \alpha \left(\frac{1}{2}\right)^k \quad \text{and} \quad \bar{X}_{k,j} = \alpha \left(\frac{1}{2}\right)^k \sum_{I \in \Omega_j^k} \left[\prod_{h \in J} P_M(\bar{\rho}_h) \right].$$

By this construction, we have $P_{\text{ud},k} = Y_k + \sum_{j=1}^k X_{k,j}$ and $\bar{P}_{\text{ud},k} = \bar{Y}_k + \sum_{j=1}^k \bar{X}_{k,j}$. We express Y_k as

$$\begin{aligned} Y_k = {} & \alpha_0 \prod_{i=1}^k \left[\left(\frac{1}{2} - w_i\right) + \frac{1}{2} \right] + \alpha_1 \prod_{i=1}^k \left[\left(w_i - \frac{1}{2}\right) + \frac{1}{2} \right] \\ = {} & \alpha_0 \sum_{j'=1}^k \sum_{J' \in \Omega_{j'}^k} \left[\left(\frac{1}{2}\right)^{k-j'} \prod_{h \in J'} \left(\frac{1}{2} - w_h\right) \right] \\ & + \alpha_1 \sum_{j'=1}^k \sum_{J' \in \Omega_{j'}^k} \left[\left(\frac{1}{2}\right)^{k-j'} \prod_{h \in J'} \left(w_h - \frac{1}{2}\right) \right] + (\alpha_0 + \alpha_1) \left(\frac{1}{2}\right)^k. \end{aligned}$$

Given that $|w_i - \frac{1}{2}| \leq \frac{\epsilon}{2}, \forall i \in [k]$, we have

$$\begin{aligned} |Y_k - \bar{Y}_k| = {} & \alpha_0 \sum_{j'=1}^k \left(\frac{1}{2}\right)^{k-j'} \sum_{J' \in \Omega_{j'}^k} \prod_{h \in J'} \left(\frac{1}{2} - w_h\right) \\ & + \alpha_1 \sum_{j'=1}^k \left(\frac{1}{2}\right)^{k-j'} \sum_{J' \in \Omega_{j'}^k} \prod_{h \in J'} \left(w_h - \frac{1}{2}\right) \\ \leq {} & \alpha \sum_{j'=1}^k \binom{k}{j'} \left[\left(\frac{\epsilon}{2}\right)^{j'} \left(\frac{1}{2}\right)^{k-j'} \right] = \frac{\alpha}{2^k} [(1+\epsilon)^k - 1]. \end{aligned} \tag{2.43}$$

Also, given that $|w_i - \frac{1}{2}| \leq \frac{\epsilon}{2}, \forall i \in [k]$, $X_{k,j}$ is upper bounded as

$$\begin{aligned}X_{k,j} &= \alpha_0 \sum_{J \in \Omega_j^k} \left[\prod_{h \in J} (w_h P_M(\rho_h)) \prod_{m \in [k] \setminus J} (1 - w_m) \right] \\ &+ \alpha_1 \sum_{J \in \Omega_j^k} \left[\prod_{h \in J} (1 - w_h) P_M(\rho_h) \prod_{m \in [k] \setminus J} w_m \right] \\ &\leq \frac{\alpha}{2^k} (1+\epsilon)^k \binom{k}{j} (P_M(\alpha))^j. \qquad (2.44)\end{aligned}$$

Similarly, we have

$$\bar{X}_{k,j} \leq \frac{\alpha}{2^k} \binom{k}{j} (P_M(\alpha))^j. \qquad (2.45)$$

Since both $X_{k,j}$ and $\bar{X}_{k,j}$ are non-negative quantities, we have

$$|X_{k,j} - \bar{X}_{k,j}| \leq \max(X_{k,j}, \bar{X}_{k,j}) \leq \frac{\alpha}{2^k} (1+\epsilon)^k \binom{k}{j} (P_M(\alpha))^j. \qquad (2.46)$$

Hence with (2.43) and (2.46), we have

$$\begin{aligned}|P_{\mathrm{ud},k} - \bar{P}_{\mathrm{ud},k}| &\leq |Y_k - \bar{Y}_k| + \sum_{j=1}^{k} |X_{k,i} - \bar{X}_{k,j}| \\ &= \frac{\alpha}{2^k} \left[(1+\epsilon)^k (1 + P_M(\alpha))^k - 1 \right]. \qquad (2.47)\end{aligned}$$

Therefore, we have (2.39). □

The undiscovered permanent error rate approaches zero as more codewords are transmitted, and so does $P_M(\rho)$ (since $P_M(\rho)$ is small when ρ is small, which we will show later in the derivation of $P_M(\rho)$). Thus, $\frac{\bar{P}_{\mathrm{ud},k}}{\bar{P}_{\mathrm{ud},k-1}} = \frac{P_M(\bar{\rho}_k)+1}{2}$ approaches 1/2. In each round, we roughly detect half of the undiscovered permanent errors left from the last round.

Next, we derive the expression for $P_M(\rho_k)$. Recall that $\{m_{v \to c}^{(\ell)}\}$ denotes the collection of variable-to-check messages at the end of iteration ℓ and that $\{\breve{m}_{v \to c}^{(\ell)}\}$ denotes the collection of variable-to-check messages that are re-computed based on the messages from the preceding iteration. Consider a check node c. Let $\hat{\xi}(c)$

be the indicator that c is not satisfied based on the messages from the collection $\{m_{v\to c}^{(\ell)}\}$. Likewise, let $\hat{\breve{\xi}}(c)$ be the indicator that c is not satisfied based on the messages from the collection $\{\breve{m}_{v\to c}^{(\ell)}\}$. We declare a variable node as having a permanent error if all its neighboring check nodes $c \in \mathcal{N}_v$ have $\xi(c) = \breve{\xi}(c) = 1$. Here, $\xi(c)$ is the output of BSC(σ_c) with $\hat{\xi}(c)$ as its input and $\breve{\xi}(c)$ is the output of BSC(σ_c) with $\hat{\breve{\xi}}(c)$ as its input.

Let us suppose that a variable node v has a perceptible permanent error. This error will not be detected if there exists c, $c \in \mathcal{N}_v$, such that $\xi(c) = 0$ or $\breve{\xi}(c) = 0$. We express the event of interest as the complement of the event: $\xi(c) = \breve{\xi}(c) = 1$, $\forall c \in \mathcal{N}_v$.

Let us consider the event $\xi(c) = \breve{\xi}(c) = 1$ for a given $c, c \in \mathcal{N}_v$. This event can be partitioned into the following events: (i) no transient error occurs: $\hat{\xi}(c) = \hat{\breve{\xi}}(c) = 1$, (ii) transient errors occur in both $\hat{\xi}(c)$ and $\hat{\breve{\xi}}(c)$: $\hat{\xi}(c) = \hat{\breve{\xi}}(c) = 0$, and (iii) a transient error occurs in $\hat{\xi}(c)$ or $\hat{\breve{\xi}}(c)$ but not both: $\hat{\xi}(c) = 0$ and $\hat{\breve{\xi}}(c) = 1$ or $\hat{\breve{\xi}}(c) = 1$, and $\hat{\xi}(c) = 0$.

Let the perceptible permanent error rate at the end of round k be ρ_k, following the notation in previous subsection. Let us suppose that in addition to v, there are t variable nodes with perceptible permanent errors connected to the check node c. We first assume that choice of these t variable nodes is fixed. Under these assumptions, let $g(t)$ be the probability of the event $\hat{\xi}(c) = 1$, i.e., that the check node is unsatisfied. If a check node is unsatisfied, the total number of adjacent erroneous variable nodes must be odd. Since v has a permanent error by assumption, there must be an even number of additional erroneous variable nodes, including both transient and permanent errors. Therefore,

$$g(t) = \sum_{\substack{0\le j-t\le d_c-1-t \\ j \text{ is even}}} \binom{d_c-1-t}{j-t} (\sigma_v)^{j-t}(1-\sigma_v)^{d_c-j-1}$$
$$= \begin{cases} \frac{1}{2}\left[1+(1-2\sigma_v)^{d_c-1-t}\right] \text{ if } t \text{ is even}, \\ \frac{1}{2}\left[1-(1-2\sigma_v)^{d_c-1-t}\right] \text{ if } t \text{ is odd}. \end{cases} \tag{2.48}$$

Likewise, the probability of the event $\hat{\check{\xi}}(c) = 1$ when $t+1$ variable nodes have permanent errors is also given by (2.48). We now calculate the mis-detection (false negative) probabilities for each case m_i, $1 \le i \le 3$.

m_1: For case 1, considering all choices for t permanent errors, we get

$$\begin{aligned} p_1(\rho_k) =& \sum_{t=0}^{d_c-1} \binom{d_c-1}{t} \rho_k^t (1-\rho_k)^{d_c-1-t} g^2(t) \\ =& \frac{1}{4} + \frac{1}{4}((1-\rho_k)(1-2\sigma_v)^2+\rho_k)^{d_c-1} \\ &+ \frac{1}{2}((1-\rho_k)(1-2\sigma_v)-\rho_k)^{d_c-1}. \end{aligned} \tag{2.49}$$

m_2: Let us now consider case 2. Continuing with the assumption that v has a perceptible permanent error, the probability of the event $\hat{\xi}(c) = \hat{\check{\xi}}(c) = 0$ for a given $c, c \in \mathcal{N}_v$ is

$$\begin{aligned} p_2(\rho_k) =& \sum_{t=0}^{d_c-1} \binom{d_c-1}{t} \rho_k^t (1-\rho_k)^{d_c-1-t} [1-g(t)]^2 \\ =& \frac{1}{4} + \frac{1}{4}((1-\rho_k)(1-2\sigma_v)^2+\rho_k)^{d_c-1} \\ &- \frac{1}{2}((1-\rho_k)(1-2\sigma_v)-\rho_k)^{d_c-1}. \end{aligned} \tag{2.50}$$

m_3: Similarly, case 3 has probability

$$\begin{aligned} p_3(\rho_k) &= 1 - p_1(\rho_k) - p_2(\rho_k) \\ &= \frac{1}{2} - \frac{1}{2}((1-\rho_k)(1-2\sigma_v)^2+\rho_k)^{d_c-1}. \end{aligned} \tag{2.51}$$

Then, the probability that a perceptible permanent error is not detected in its round k is

$$\begin{aligned}P_M(\rho_k) &= 1 - \left[p_1(\rho_k)(1-\sigma_c)^2 + p_2(\rho_k)\sigma_c^2 + p_3(\rho_k)\sigma_c(1-\sigma_c)\right]^{d_v} \\ &= 1 - [\frac{1}{4} + \frac{(1-2\sigma_c)^2}{4}((1-\rho_k)(1-2\sigma_v)^2 + \rho_k)^{d_c-1} \\ &+ \frac{1-2\sigma_c}{2}((1-\rho_k)(1-2\sigma_v) - \rho_k)^{d_c-1}]^{d_v}.\end{aligned} \tag{2.52}$$

Note that when σ_v and σ_c are small, $P_M(\rho_k)$ is close to zero when ρ_k approaches zero.

We proceed to analyze the probability of false alarm.

In a false alarm event, the variable node of interest, v, is mistakenly classified as having a permanent error. This occurs when all $c \in \mathcal{N}_v$ have $\xi(c) = \breve{\xi}(c) = 1$. We can partition this event into three cases, f_1, f_2 and f_3:

f_1: v experiences transient errors in both $\{m_{v,c}^{(\ell)}\}$ and $\{\breve{m}_{v,c}^{(\ell)}\}$, and $\xi(c) = \breve{\xi}(c) = 1, \forall c \in \mathcal{N}_v$:

For $\xi(c) = \breve{\xi}(c) = 1, \forall c \in \mathcal{N}_v$, the probability that v experiences a transient error in both $\{m_{v,c}^{(\ell)}\}$ and $\{\breve{m}_{v,c}^{(\ell)}\}$ is the same as the (perceived) probability of the event that v has a permanent error. Thus, similar to (2.52),

$$p_{f_1}(\rho_k) = \sigma_v^2[p_1(\rho_k)(1-\sigma_c)^2 + p_2(\rho_k)\sigma_c^2 + p_3(\rho_k)\sigma_c(1-\sigma_c)]^{d_v}.$$

f_2: v does not experience a transient error in either $\{m_{v,c}^{(\ell)}\}$ or $\{\breve{m}_{v,c}^{(\ell)}\}$, and $\xi(c) = \breve{\xi}(c) = 1, \forall c \in \mathcal{N}_v$:

Similar to the previous case and by careful book-keeping we reach

$$p_{f_2}(\rho_k) = (1-\sigma_v)^2[p_1(\rho_k)\sigma_c^2 + p_2(\rho_k)(1-\sigma_c)^2 + p_3(\rho_k)\sigma_c(1-\sigma_c)]^{d_v}.$$

f_3: v experiences a transient error in exactly one of $\{m_{v,c}^{(\ell)}\}$ and $\{\breve{m}_{v,c}^{(\ell)}\}$, and $\xi(c) = \breve{\xi}(c) = 1, \forall c \in \mathcal{N}_v$:

$$p_{f_3}(\rho_k) = \sigma_v(1-\sigma_v)[2(p_1(\rho_k) + p_2(\rho_k))\sigma_c(1-\sigma_c) + p_3(\rho_k)((1-\sigma_c)^2 + \sigma_c^2)]^{d_v}.$$

The false alarm rate is $P_F(\rho_k) = p_{f_1}(\rho_k) + p_{f_2}(\rho_k) + p_{f_3}(\rho_k)$. The cost of a false alarm is determined by the cost and availability of backup memory cells. With limited but non-zero on-chip redundancy, it is reasonable to assume that the cost of false alarm is higher than that of mis-detection, but is still bounded. Therefore, it is beneficial to reduce P_F (and the implied replacement cost) by labeling only the candidates recorded in $\Gamma \cap \check{\Gamma}$ instead of those in Γ. Further, observe that the false alarm rate decreases as the variable node degree increases. Also, as more permanent errors are detected and fixed by cell replacement, P_F approaches σ_v^2. Hence with a higher variable node degree and a lower transient error rate in the variable nodes, the additional cost of permanent error detection, incurred by unnecessary memory cell replacements, is reduced. We then generally prefer codes with a higher variable node degree and lower variable node transient error rates.

In fact, we can apply the march test [45], which is commonly implemented in a memory built-in-self-test (BIST) circuit, to decrease the false alarm rate in the proposed permanent error detection scheme. We can run the march test on the memory cells in $\Gamma \cap \check{\Gamma}$ (memory cells labeled as defective) to check whether they are defect memory cells. To reuse the memory cells with intermittent faults, we can also periodically perform march tests on the memory cells already labeled as defect cells.

We remark that the effectiveness of the hardware detection-and-correction scheme hinges on the reliability of the additional circuit implementing the scheme, e.g., the memory storing the addresses of the labeled cells (to be replaced). The reliability can be ensured by deploying sufficient redundancy. Since the fraction of labeled cells should not exceed $\alpha + P_F$, whose order does not exceed that of $\alpha + \sigma_v$, such redundancy can be deployed with tolerable cost.

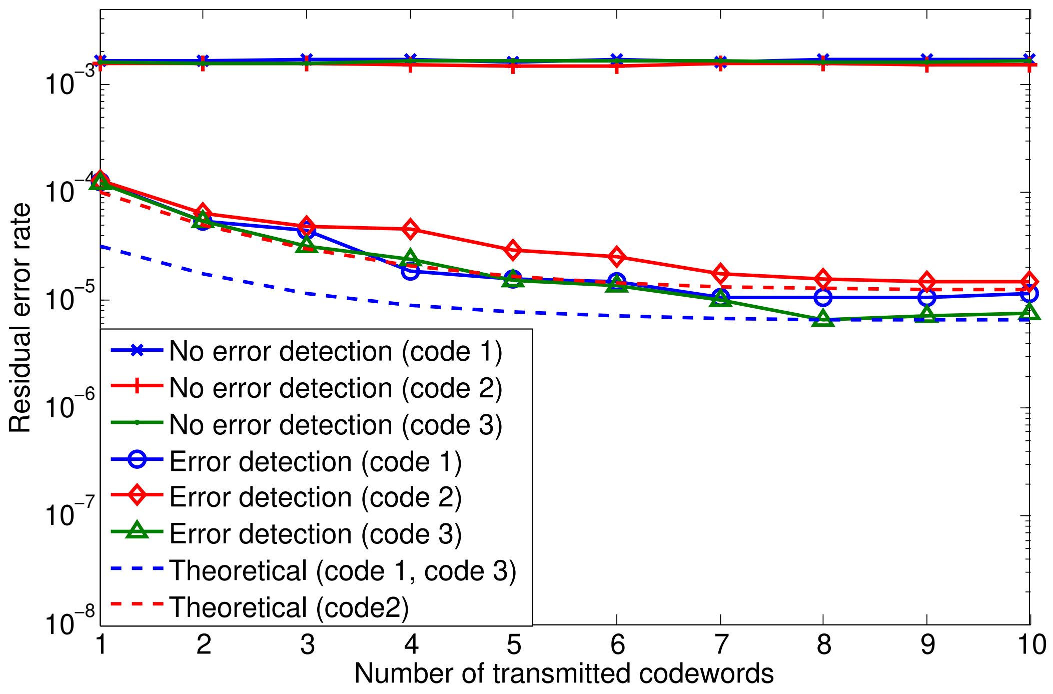


Figure 2.3: Residual error rate comparison: permanent error rate 2×10^{-3}, transient error rate 5×10^{-4}.

2.3.3.4 Simulation Results

In this section we discuss the simulation results for two representative LDPC codes, taken from [41]. The codes are both of length 204. Code 1 is a $(3, 6)$-regular code and code 2 is a $(5, 10)$-regular code. In the implementation of the noisy decoder the permanent error rates were set as $\alpha_1 = \beta_1 = \gamma_1 = 1.2 \times 10^{-3}$ and $\alpha_0 = \beta_0 = \gamma_0 = 8 \times 10^{-4}$. The overall permanent error rate was thus 2×10^{-3}, and the transient error rates were $\sigma_v = \sigma_c = 5 \times 10^{-4}$. The channel error rate was $\epsilon = 2 \times 10^{-3}$. We compared the noisy decoder that incorporates the proposed hardware error detection-and-correction scheme with the nominal noisy decoder (without error detection).

Fig. 2.3 shows the residual error rates versus the number of codeword transmissions for the proposed scheme. The theoretical results are the product of the

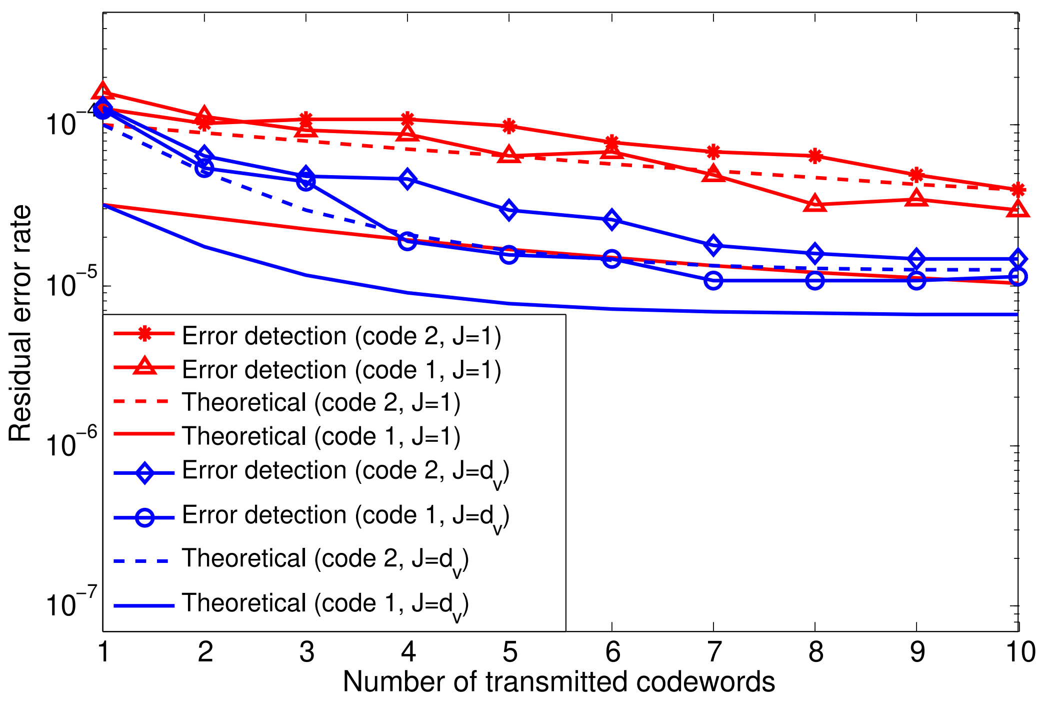


Figure 2.4: Residual error rate comparison for $J = 1$ and $J = d_v$ (the original scheme). Permanent error rate 2×10^{-3}, transient error rate 5×10^{-4}.

residual error rates derived in Theorem 2 and P_{nc}, the probability of an undetected residual error under our hardware error detection-and-correction scheme, derived in (2.38). These results reveal the improvement in the performance of the noisy decoder for both codes when we deploy the proposed detection-and-correction scheme. As indicated by the reduced residual error rate in the first transmission, the proposed scheme is able to correct most of the residual errors in the current transmission, lowering the error rate to 10^{-5}, even below the transient error rate 5×10^{-4}. Both the theoretical and experimental residual error rates decrease as more codewords are transmitted (as more permanent errors are detected).

The gap between the experimental and the theoretical curves is attributed to (i) the linear approximation that we made when deriving Theorem 2, and (ii) finite-length effects. In the derivation of Theorem 2, the linear approximation of the function $\hat{\phi}_V(b, z)$ is better with higher node degrees and smaller hardware error rates. Accordingly, we observe from Fig. 2.3 that the theory-experiment gap is smaller for the code with a higher node degree (code 2). We also observe that the gaps have a shrinking trend as more codewords are transmitted. Additionally, we plot the residual error rates of code 3 [41], a length-2640 $(3, 6)$-regular code that is longer than code 1 (length-204) but has the same node degrees. The reduced theory-experiment gap compared with that of code 1 further suggests that finite-length effects are less prominent for longer codes.

In Fig. 2.4, we compare the original scheme with the low-overhead modification mentioned in Section 2.3.3.1. In this modified scheme, we examine J $(J < d_v)$ instead of all d_v memory cells associated with each variable node by repeating the detection scheme J times only. We show simulation results for $J = 1$ and $J = d_v$ (the original scheme). Expectedly, it is observed that a smaller J results in a slower descent in the residual error rate as codeword transmissions proceed. The slowing down is more evident with the code of higher variable node degree (code 2), since a smaller fraction of memory cells are examined in each codeword transmission.

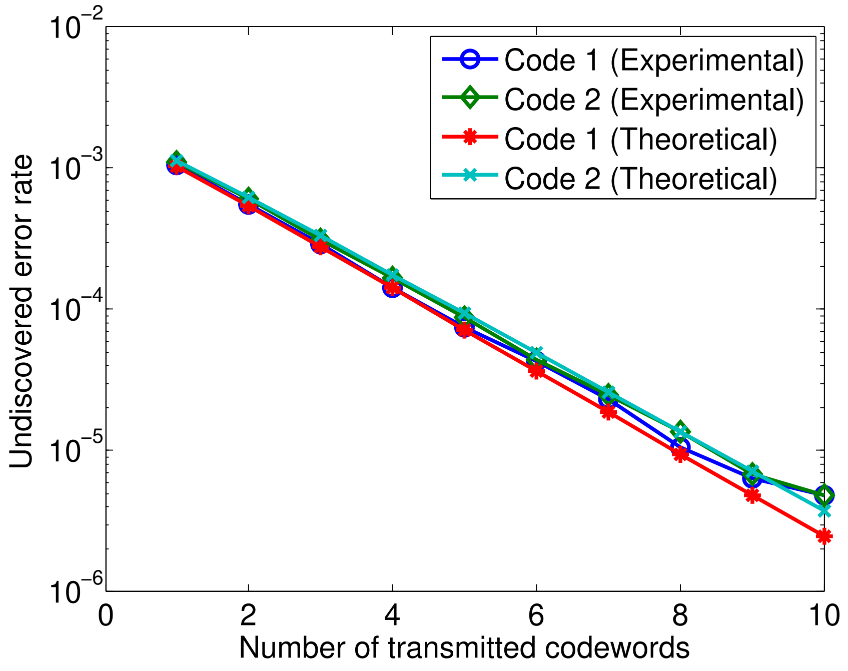
With the low-overhead scheme, it takes roughly $i \cdot d_v/J$ transmissions to reach the residual error rate achieved in i transmissions running the original scheme.

Meanwhile, permanent error detection and cell replacement reduce the residual error in future transmissions. The more codewords are transmitted, the more permanent errors are detected, and the better the performance of the noisy decoder becomes. The residual error rate of the nominal noisy decoder, however, remains the sum of the transient error rate and the perceptible permanent error rate throughout the transmissions.

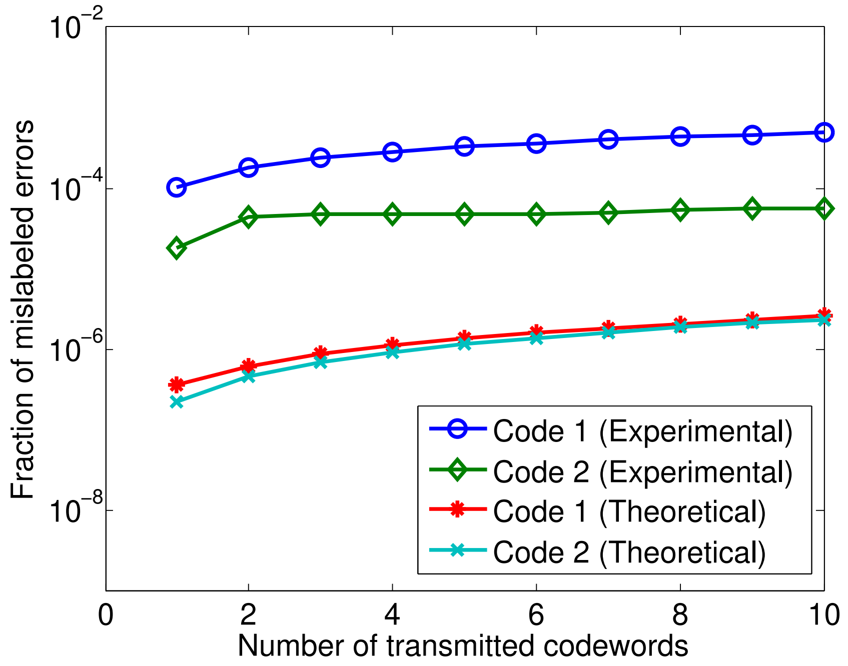
Fig. 2.5a and 2.5b plots the undiscovered error rate and the fraction of mislabeled errors (with respect to the total number of memory cells) versus the number of codeword transmissions. In Fig. 2.5a, the theoretical curve is computed as $P_{\text{ud},k} = P_{\text{ud},k,0} + P_{\text{ud},k,1}$ from the expressions in Lemma 1, with $P_M(\cdot)$ as in (2.52). We observe from Fig. 2.5a that almost half of the undiscovered permanent errors can be detected in each transmission round, as predicted in Section 2.3.3.3. In Fig. 2.5b, the experimental results are plotted alongside $\sum_{\kappa=1}^{k} P_F(\rho_\kappa)$. Both the theoretical curves and the experimental curves exhibit the same trend: the growth rate of the number of the memory cells that are mislabeled as permanent errors diminishes as more codewords are transmitted. This trend occurs because the perceptible error rate ρ_k decreases after each codeword transmission, leading to the decrease in $P_F(\rho_k)$.

The gap between the experimental and the theoretical curves again reflects the finite-length effect. Also, our derivation of P_M and P_F ignores the effects of errors in check nodes and check-to-variable messages on the residual errors. However, as permanent errors in variable-to-check messages are being eliminated over transmissions, the effects of check node errors become dominant. This also partly accounts for the gap.

Finally, we also briefly comment on the effects of the variable node degree on the fraction of mislabeled errors. As is shown in Fig. 2.5b, the fraction of misla-



(a)



(b)

Figure 2.5: Performance comparison: (a) undiscovered error rate, (b) fraction of mislabeled errors (to total number of cells).

beled errors of the code with a higher variable node degree (code 2) is significantly lower than that of the code with a lower variable node degree (code 1), although the undiscovered error rates of code 1 and code 2 are shown to be comparable in Fig. 2.5a.

CHAPTER 3

Noisy Belief Propagation

In this chapter, we apply the analysis and design methodology developed in the previous chapter for iterative decoders to study a more general inference algorithm, belief propagation (BP) on probabilistic graphical models (factor graphs), implemented on noisy hardware. Based on the unreliable hardware inference system model in Fig. 1.2, we analyze the performance of BP on noisy hardware and propose two robust implementations of the BP algorithm targeting different computation noise distributions. Simulations and application examples are provided to demonstrate the effectiveness of the proposed implementations.

3.1 Background and Previous Work

3.1.1 BP on Factor Graphs

In this section, we review the BP algorithm on factor graphs and introduce necessary notation.

3.1.1.1 Graphical Models

Consider a random vector $X = (X_1, X_2, \ldots, X_N)$, with X_i taking values in a discrete space $\mathcal{X} = \{1, 2, \ldots d\}$. We are interested in a class of factorizable probability distributions defined on graphical models such as Markov random fields and Bayesian networks. As in the standard literature, we write the probability distribution of X as a product of *factors* (also called *potentials*)

$$P_{X_1,\ldots,X_N}(x_1,\ldots,x_N) = \frac{1}{Z}\prod_{I\in\mathcal{C}}\psi_I(x_I), \tag{3.1}$$

where $\mathcal{C}$ is the set of all maximal cliques in the graph, x_I is the vector whose elements are in the set $\{x_i | i \in I\}$, I is an index set, and Z is the normalization constant to ensure that $\sum_{x\in\mathcal{X}^N} P_X(x) = 1$, where $\mathcal{X}^N$ denote the N-fold cartesian product of $\mathcal{X}$.

The probability distribution (3.1) can be associated with a factor graph. A factor graph is a bipartite graph with vertex set $\mathcal{V}\cup\mathcal{F}$, where $\mathcal{F}$ is the set of factors and $\mathcal{V}$ is the set of variable nodes. In the factor graph, we denote a factor by its associated set I, which is the set of indices of the variable nodes connected to this factor, and we denote a variable by its index $i, i \in \{1, 2, \ldots, N\}$. The variable node i is associated with the random variable X_i in X, and each variable node $i \in \mathcal{V}$ is connected with the factors $I \in \mathcal{F}$ iff $i \in I$. We denote the neighbors of variable node i by $\mathcal{N}_i$, hence, $\mathcal{N}_i = \{I : i \in I\}$. For each variable node i, we denote the set of neighboring variable nodes (via connecting factors) as $\Psi_i = \bigcup_{I\in\mathcal{N}_i} I\backslash\{i\}$. Each variable X_i interacts with its neighboring variables, $X_j, j \in \Psi_i$, through the connecting factors in $\mathcal{N}_i$.

3.1.1.2 Noise-Free BP

The BP algorithm computes the approximate or exact marginals $\{P_{X_I}(x_I)\}, I \in \mathcal{F}$ and $\{P_{X_i}(x_i)\}, i \in \mathcal{V}$ of the joint distribution (3.1). The computation is conducted by iterative message passing on the factor graph: each node updates its messages according to the messages received from its neighbors, and sends its updated messages back to its neighbors. In BP, we have two types of messages: messages from variable nodes to factors, denoted by $\tilde{m}^{(\ell)}_{i\to I}(x_i)$, and messages from factors to variable nodes, denoted by $\tilde{m}^{(\ell)}_{I\to i}(x_i)$, where ℓ is the iteration number. Each message is associated with a variable node i and a factor I on the factor graph.

The messages are interpreted as proxies of marginal probabilities and take non-negative values. At iteration ℓ, every variable node i computes its messages $\tilde{m}^{(\ell)}_{i\to I}(x_i)$ from the received messages in the last iteration and sends $\tilde{m}^{(\ell)}_{i\to I}(x_i)$ to its neighboring factors $I \in \mathcal{N}_i$, and then factor I computes its messages $\tilde{m}^{(\ell)}_{I\to i'}(x_{i'})$ by the received messages from neighboring variable nodes and sends the computed messages to its neighboring variable nodes $i' \in I$. The noise-free BP update rules are summarized as follows. The variable-to-factor message update rule is

$$\tilde{m}^{(\ell)}_{i\to I}(x_i) \propto \prod_{J\in\mathcal{N}_i\setminus I} \tilde{m}^{(\ell-1)}_{J\to i}(x_i). \tag{3.2}$$

The factor-to-variable message update rule is

$$\tilde{m}^{(\ell)}_{I\to i'}(x_{i'}) \propto \sum_{x_{I\setminus i'}\in\mathcal{X}^{|I|-1}} \psi_I(x_I) \prod_{i\in I\setminus i'} \tilde{m}^{(\ell)}_{i\to I}(x_i). \tag{3.3}$$

We combine (3.2) and (3.3) into a recursive expression for updates of factor-to-variable messages:

$$\tilde{m}^{(\ell)}_{I\to i'}(x_{i'}) \propto \sum_{x_{I\setminus i'}\in\mathcal{X}^{|I|-1}} \psi_I(x_I) \prod_{i\in I\setminus i'} \prod_{J\in\mathcal{N}_i\setminus I} \tilde{m}^{(\ell-1)}_{J\to i}(x_i). \tag{3.4}$$

To simplify the notation, we concatenate the local messages $\tilde{m}^{(\ell)}_{I\to i'}(x_{i'})$, for all $I \in \mathcal{F}, i' \in \mathcal{V}$, and $x_{i'} \in \mathcal{X}$, to form an aggregate message (vector) $\tilde{m}^{(\ell)}$. Denote the operations on the aggregate message in each iteration by a function f. We then have

$$\tilde{m}^{(\ell)} = f(\tilde{m}^{(\ell-1)}). \tag{3.5}$$

When all the messages have converged to a fixed point (i.e., $\tilde{m}^{(\ell)}_{i\to I}(x_i) = \tilde{m}^{(\ell-1)}_{i\to I}(x_i)$ and $\tilde{m}^{(\ell)}_{I\to i'}(x_{i'}) = \tilde{m}^{(\ell-1)}_{I\to i'}(x_{i'})$ for all $I \in \mathcal{F}, i \in \mathcal{V}$, and $x_i \in \mathcal{X}$), which we describe by $\tilde{m}^*_{i\to I}(x_i)$ and $\tilde{m}^*_{I\to i'}(x_{i'})$, we compute the approximate marginals or *beliefs* by

$$\tilde{b}_I(x_I) = C_I\psi_I(x_I)\prod_{i\in I}\tilde{m}^*_{i\to I}(x_i),\ \tilde{b}_{i'}(x_{i'}) = C_{i'}\prod_{I\in\mathcal{N}_{i'}}\tilde{m}^*_{I\to i'}(x_{i'}),$$

where C_I's and $C_{i'}$'s are normalizing constants chosen such that $\sum_{x_I \in \mathcal{X}^{|I|}} \tilde{b}_I(x_I) = 1$ and $\sum_{x_{i'} \in \mathcal{X}} \tilde{b}_{i'}(x_{i'}) = 1$. Then, the proxies of[1] marginals $P_{X_{i'}}(x_{i'})$ and $P_{X_I}(x_I)$ are $\tilde{b}_{i'}(x_{i'})$ and $\tilde{b}_I(x_I)$. Note that in the aggregate representation (3.5), the fixed point is then described by the aggregate message satisfying $\tilde{m}^* = f(\tilde{m}^*)$.

It is well-known that on a tree-structured graph, BP converges to a unique fixed point. On a general graph (with some mild conditions on the potentials [46]), it is known that BP has at least one fixed point, but it is not necessarily unique. Sufficient convergence conditions based on the contractivity of the message update operations were proposed as guarantees for BP to converge to a unique fixed point [47–49].

3.1.2 BP Decoder for LDPC codes

On an LDPC code bipartite graph, following the notation in Section 2.1.2, we denote the neighbors of variable (check) node v (c) by the set $\mathcal{N}_v$ ($\mathcal{N}_c$). Denote the variable-to-check messages (at iteration ℓ) by $\tilde{m}^{(\ell)}_{v \to c}$, and the check-to-variable messages by $\tilde{m}^{(\ell)}_{c \to v}$. Let y_v be the decoder input at the variable node v, and $x_v \in \{+1, -1\}$ be the transmitted bit (using binary phase-shift keying modulation) corresponding to v. Let us summarize the well-known steps of the noise-free BP decoder in the following.

- (Initialization) At iteration $\ell = 0$, each variable node v sends the message $\tilde{m}^{(0)}_{v,c} = \ln \frac{P(y_v|x_v=1)}{P(y_v|x_v=-1)}$ to each check node c, $c \in \mathcal{N}_v$.
- (Check node) At each iteration ℓ, $\ell \geq 0$, each check node c sends a message $\tilde{m}^{(\ell)}_{c,v}$ to each variable node v, $v \in \mathcal{N}_c$:
$$\tilde{m}^{(\ell)}_{c,v} = 2 \tanh^{-1} \left(\prod_{v' \in \mathcal{N}_c \setminus \{v\}} \tanh \frac{\tilde{m}^{(\ell)}_{v',c}}{2} \right).$$

[1]In tree-structured graphs, $\tilde{b}_{i'}(x_{i'})$ and $\tilde{b}_I(x_I)$ are exact marginals $P_{X_{i'}}(x_{i'})$ and $P_{X_I}(x_I)$, respectively.

- (Variable node) At each iteration ℓ, $\ell \geq 1$, each variable node v sends a message $\tilde{m}_{v,c}^{(\ell)}$ to each check node c, $c \in \mathcal{N}_v$:
$$\tilde{m}_{v,c}^{(\ell)} = \tilde{m}_{v,c}^{(0)} + \sum_{c' \in \mathcal{N}_v \setminus \{c\}} \tilde{m}_{c',v}^{(\ell-1)}.$$

Since the messages in the BP decoder are log-likelihood ratios, they take values in $\mathbb{R}$. When the decoding process terminates at iteration L, the decoded bit of variable node v is decided by the sign of $\tilde{m}_v^{(L)}$, where $\tilde{m}_v^{(L)} = \tilde{m}_{v,c}^{(0)} + \sum_{c' \in \mathcal{N}_v} \tilde{m}_{c',v}^{(L-1)}$.

3.1.3 Previous Work: BP and Noisy Infinite Precision LDPC Decoders

The belief propagation (BP) algorithm on graphical models has found wide spread applications, including image processing, digital communication systems, error correction coding, and bioinformatics [50]. Performance characterization of the BP algorithm, especially the loopy BP for graphs with cycles, was extensively studied in previous work. The optimality conditions of loopy BP were shown in [46]. Various conditions on convergence were proposed [47–49] and different modified BP algorithms were developed [51, 52]. Message quantization for hardware implementation has been investigated in [47, 53] and clever quantization schemes with performance analysis of quantization effects are derived

Study of LDPC decoders implemented on noisy hardware has recently attracted a lot of attention. In Chapter 2, we introduced some previous works on finite precision decoders on noisy hardware and study the density evolution analysis and robust system design for noisy finite precision decoders. In this Chapter, we study infinite precision decoders as an application example of the BP algorithm. Density evolution analysis for the min-sum decoder decoder under different hardware error models was investigated in [54, 55]. A density evolution analysis for BP decoders was derived in [36] which considered the constant (worst case) computation noise.

3.2 BP on Noisy Hardware

3.2.1 System Model and Convergence Analysis

In this section, we consider the transient error model for the BP algorithm implemented on noisy hardware. We further assume that the effect of transient errors is modeled as zero-mean additive noise, and the message update at iteration ℓ is

$$m_{I\to i}^{(\ell)}(x_i) = \hat{m}_{I\to i}^{(\ell)}(x_i) + w_{I\to i}^{(\ell)}(x_i), \tag{3.6}$$

where $w_{I\to i}^{(\ell)}(x_i)$ is the additive computation noise due to transient errors and $\hat{m}_{I\to i}^{(\ell)}(x_i)$ is the hypothetical noise-free message. Such zero-mean additive computation noise model was also considered in related works [36, 56]. Using the aggregate representation in (3.5), we have

$$m^{(\ell)} = f(m^{(\ell-1)}) + w^{(\ell)}. \tag{3.7}$$

Since the transient errors on different messages are independent, the entries in $w^{(\ell)}$ are independent.

We assume that the factor graph we consider satisfies the following condition:

$$\sup_{u\in U} \|f'(u)\| < 1, \tag{3.8}$$

where U is the domain of the function $f(u)$ and $\|\cdot\|$ denotes the vector norm. Under this condition, BP converges to a unique fixed point [48]. This condition is called the *contraction mapping condition.* Let $\sup_{u\in U}\|f'(u)\| = K$. We first derive an upper bound for the distances (in terms of the vector norm) between the fixed point and the noisy BP messages $m^{(\ell)}$. Note that "fixed point" refers to the fixed point of noise-free BP in the rest of this chapter. We first introduce the following lemma from [48] , which is the consequence of the well-known Mean Value Theorem.

Lemma 2. *Let $g : U \to U$ be a differentiable mapping on a normed space $(U, \|\cdot\|)$. We have*

$$\|g(u_2) - g(u_1)\| \leq \|u_2 - u_1\| \cdot \sup_{y \in [u_2, u_1]} \|g'(y)\|, \tag{3.9}$$

where $[u_2, u_1]$ is the segment between u_2 and u_1, $\{\lambda u_1 + (1-\lambda)u_2 : \lambda \in [0,1]\}$, and $u_1, u_2 \in U$.

The following lemma gives an upper bound on the distance between the fixed point m^* and the message $m^{(\ell)}$ as a function of ℓ. Denote the maximum number of iterations by L.

Lemma 3. *Let $\xi > 0$. Suppose that for all $\ell \leq L$, $\|w^{(\ell)}\| \leq \xi$. Let $\sup_{u \in U} \|f'(u)\| = K$. We then have $\|m^{(\ell)} - m^*\| \leq \|m^{(0)} - m^*\|K^\ell + \frac{(1-K^\ell)}{1-K}\xi$, where m^* denotes the fixed point.*

Proof. We prove this lemma by induction. For $\ell = 1$, by applying Lemma 2, we have

$$\begin{aligned}\|m^{(1)} - m^*\| &= \|f(m^{(0)}) + w^{(1)} - m^*\| \\ &\leq \|f(m^{(0)}) - m^*\| + \xi \leq \|m^{(0)} - m^*\|K + \xi.\end{aligned}$$

Let

$$\|m^{(\ell)} - m^*\| \leq \|m^{(0)} - m^*\|K^\ell + \frac{(1-K^\ell)}{1-K}\xi.$$

For iteration $\ell + 1$, by using Lemma 2, we have

$$\begin{aligned}&\|m^{(\ell+1)} - m^*\| = \|f(m^{(\ell)}) + w^{(\ell+1)} - m^*\| \\ &\leq \|f(m^{(\ell)}) - m^*\| + \xi \leq \|m^{(\ell)} - m^*\|K + \xi \\ &\leq \|m^{(0)} - m^*\|K^{\ell+1} + \frac{(1-K^{\ell+1})}{1-K}\xi.\end{aligned}$$

□

Corollary 1. *Let ξ and K be as defined in Lemma 3. Then, when $\ell \to \infty$, $\|m^{(\ell)} - m^*\|$ is upper bounded by $\frac{\xi}{1-K}$.*

Proof. Since $\|K\| < 1$, $\lim_{\ell\to\infty} K^\ell = 0$, the corollary follows immediately. $\square$

Corollary 1 shows that when we run noisy BP for a large enough number of iterations, the distance bound depends only on ξ and K. Although the distance bound is not tight, Corollary 1 has an important implication. Since the noisy messages are confined within this bound in all iterations, we are able to develop robust implementations to improve BP on noisy hardware, as shown in the following sections.

3.2.2 The Censoring BP Algorithm

We first consider the transient errors with the following distribution

$$P(w^{(\ell)}(j) = \phi) = \begin{cases} p_w & \text{if } \phi = 0, \\ h(\phi) & \text{otherwise,} \end{cases} \tag{3.10}$$

where $w^{(\ell)}(j)$ denotes the jth entry in $w^{(\ell)}$, h is an arbitrary function satisfying $\int_{-\infty}^{\infty} h(z)dz = 1 - p_w$ and $\int_{-\infty}^{\infty} zh(z)dz = 0$, and p_w is positive. In this case, with probability p_w, noisy BP computes a correct message. If we are able to discard the erroneous computations and replace them by recomputed messages, noisy BP will converge to the fixed point as noise-free BP.

We observe that under the contraction mapping condition, $\|m^{(\ell)} - m^{(\ell-1)}\|$ is a strictly decreasing function of ℓ, and converges to zero as $\ell \to \infty$, when we assume noise-free computation. Hence, by comparing $\|m^{(\ell)} - m^{(\ell-1)}\|$ with $\|m^{(\ell-1)} - m^{(\ell-2)}\|$, and recomputing $m^{(\ell)}$ when $\|m^{(\ell)} - m^{(\ell-1)}\| > K\|m^{(\ell-1)} - m^{(\ell-2)}\|$, we are able to discard most of the erroneous computations and keep the messages on the right track to convergence. However, the problem of this idea is that the iterative message update could be trapped in some points other than the fixed point, when $w^{(\ell-1)}$ happens to cause $m^{(\ell-1)}$ to be very close to $m^{(\ell-2)}$. In this situation, even when $w^{(\ell)}$ is zero, we still have $\|m^{(\ell)} - m^{(\ell-1)}\| >$

$K\|m^{(\ell-1)} - m^{(\ell-2)}\|$, since the small value of $\|m^{(\ell-1)} - m^{(\ell-2)}\|$ is the result of detrimental computation noise $w^{(\ell-1)}$. We therefore include a backoff mechanism in our algorithm to solve this problem. That is, when the computation results of $m^{(\ell)}$ are repeatedly rejected (up to τ times, where τ is the recomputation threshold), we recompute $m^{(\ell-1)}$, and based on the recomputed $m^{(\ell-1)}$, we compute $m^{(\ell)}$. We summarize the censoring BP algorithm in Algorithm 1. The BP stopping rule is $\|m^{(\ell)} - m^*\| \leq K^T\|m^{(0)} - m^*\|$, where T is a positive integer (T is determined by the desired accuracy of estimated beliefs). Note that in practice, since we do not know m^* (and even K) *a priori*, we can compute $\|m^{(\ell)} - m^{(\ell-1)}\|$ for every ℓ, and stop when the differences are small for several consecutive ℓ's.

Algorithm 1 Censoring BP

```
$\ell \leftarrow 0$
**while** $\|m^{(\ell)} - m^*\| > K^T\|m^{(0)} - m^*\|$ **do** $\ell \leftarrow \ell + 1,\ m^{(\ell)} \leftarrow f(m^{(\ell-1)}),\ i \leftarrow 0;$
    **while** $\|m^{(\ell)} - m^{(\ell-1)}\| > K\|m^{(\ell-1)} - m^{(\ell-2)}\|$ & $i \leq \tau$ **do** $i \leftarrow i + 1,$
$m^{(\ell)} \leftarrow f(m^{(\ell-1)})$
        **if** $i = \tau$ **then** $\ell \leftarrow \ell - 1$
        **end if**
    **end while**
**end while**
```

We discuss the convergence of the censoring BP algorithm in the following. With censoring BP, the distance between the current message and the fixed point can be modeled as a 1-D random walk starting at 0 and ending at T. To be more specific, we start from position 0, where we have $\|m^{(0)} - m^*\| \leq K^0\|m^{(0)} - m^*\|$, and reach position T at the final iteration Λ, where we have $\|m^{(\Lambda)} - m^*\| \leq K^T\|m^{(0)} - m^*\|$. When the message at the iteration ℓ, $m^{(\ell)}$, is at position t, we have $\|m^{(\ell)} - m^*\| \leq K^t\|m^{(0)} - m^*\|$. A correct computation is equivalent to one step forward towards T. An erroneous (but accepted) computation is equivalent to a few steps backwards away from T. Without detecting the erroneous computations (for example, by checking if $\|m^{(\ell)} - m^{(\ell-1)}\| \leq K\|m^{(\ell-1)} - m^{(\ell-2)}\|$) and replacing them with recomputed messages, the number of steps taken backward is $\log_K(\frac{K^{t+1}\|m^{(0)}-m^*\|+\|w^{(\ell)}\|}{K^t\|m^{(0)}-m^*\|})$ when the current message is at iteration ℓ and position t. This number grows as t increases, and the convergence to the fixed point is not guaranteed. However, by ensuring that $\|m^{(\ell)} - m^{(\ell-1)}\| \leq K\|m^{(\ell-1)} - m^{(\ell-2)}\|$ is satisfied, we are able to upper bound the number of steps backward for the (accepted) erroneous computations as follows.

To simplify the analysis, we assume that the first computation is noise-free. This can be achieved by repeated computations and a majority voting. We first derive an upper bound on $\|w^{(\ell)}\|$ in terms of $\|m^{(\ell)} - m^{(\ell-1)}\|$ when $m^{(\ell)}$ is accepted by our algorithm in the following:

$$\begin{aligned}\|m^{(\ell)} - m^{(\ell-1)}\| &= \|f(m^{(\ell-1)}) + w^{(\ell)} - m^{(\ell-1)}\| \\ &\geq \|w^{(\ell)}\| - \|f(m^{(\ell-1)}) - m^{(\ell-1)}\|. \end{aligned} \tag{3.11}$$

Since $\|m^{(i)} - m^{(i-1)}\| \leq K\|m^{(i-1)} - m^{(i-2)}\|$ for all $i \leq \ell$, we have $\|m^{(\ell)} - m^{(\ell-1)}\| \leq K^{\ell-1}\|m^{(1)} - m^{(0)}\|$. Therefore, we have the following upper bound for $w^{(\ell)}$:

$$\|w^{(\ell)}\| \leq K^{\ell-1}\|m^{(1)} - m^{(0)}\| + \|f(m^{(\ell-1)}) - m^{(\ell-1)}\|. \tag{3.12}$$

The first term, $K^{\ell-1}\|m^{(1)} - m^{(0)}\|$, is upper bounded by

$$\begin{aligned}K^{\ell-1}\|m^{(1)} - m^{(0)}\| &\leq K^{\ell-1}(\|m^{(1)} - m^*\| + \|m^{(0)} - m^*\|) \\ &\leq K^{\ell-1}(K+1)\|m^{(0)} - m^*\|. \end{aligned} \tag{3.13}$$

The second term in (3.12) is upper bounded by

$$\begin{aligned}&\|f(m^{(\ell-1)}) - m^{(\ell-1)}\| \leq \|f(m^{(\ell-1)}) - m^*\| + \|m^{(\ell-1)} - m^*\| \\ &\leq (K+1)\|m^{(\ell-1)} - m^*\| \leq K^{t-1}(K+1)\|m^{(0)} - m^*\|, \end{aligned} \tag{3.14}$$

where $t-1$ is the position of the previous message $m^{(\ell-1)}$ in the random walk we described in the previous paragraph, and $\|m^{(\ell-1)} - m^*\| \leq K^{t-1}\|m^{(0)} - m^*\|$.

Therefore, we have

$$\|w^{(\ell)}\| \leq 2(K+1)K^{t-1}\|m^{(0)} - m^*\|. \tag{3.15}$$

We used the fact that $\ell \geq t$, since at the ℓth iteration, the message cannot go beyond the distance bound $K^{\ell}\|m^{(0)} - m^*\|$. We then have

$$\begin{aligned} \|m^{(\ell)} - m^*\| &\leq K \cdot K^{t-1}\|m^{(0)} - m^*\| + \|w^{(\ell)}\| \\ &\leq K^t\|m^{(0)} - m^*\| + 2(K+1)K^{t-1}\|m^{(0)} - m^*\| \\ &= (3K+2)K^{t-1}\|m^{(0)} - m^*\|. \end{aligned} \tag{3.16}$$

Hence, the number of steps backward when we have an (accepted) erroneous computation is upper bounded by $\log_K(3K+2)$. We denote probability of satisfying $\|m^{(\Lambda)} - m^*\| \leq K^T\|m^{(0)} - m^*\|$ at the final iteration $\Lambda \leq L$ by p_f, where L is the maximum number of iterations. Then, p_f is lower bounded by the probability of the random walk hitting the destination, position T, within L steps. When $(1-p_w)\log_K(3K+2) < p_w$, as $L \to \infty$, censoring BP converges to the fixed point w.p.1.

For finite but sufficiently large L and p_w, we ignore the small probability[2] that the algorithm does not stop before the iteration number reaches L, i.e., in L steps the random walk described previously has not reached position T, and derive the expected number of steps (iterations) when the random walk reaches position T for the first time in the following. Let the position at the ℓth step (corresponding to the ℓth iteration in BP) be Z_ℓ, the displacement of the ℓth step be Y_ℓ, and $A_\ell = T - Z_\ell$. Let η be the stopping time, hence $A_\eta = 0$ (the BP messages reach T and stop at iteration η). Denote the expectation of Y_ℓ by α. According to the previous discussion, in the random walk process, the message takes 1-step forward with probability p_w, and takes $\log_K(3K+2)$-steps backwards with probability $1 - p_w$ (Here we consider the worst case in which all the erroneous computations

[2]This probability is small because the probability of hitting T within L steps increases with L and p_w.

are accepted). Hence we have $\alpha = \mathbb{E}[Y_\ell] = p_w - (1-p_w)\log_K(3K+2)$. Let $B_\ell = A_\ell + \alpha\ell$. B_ℓ is a martingale since

$$\begin{aligned}
&\mathbb{E}[B_\ell | B_{\ell-1}] \\
&= (A_{\ell-1} - 1)p_w + (A_{\ell-1} + \log_K(3K+2))(1-p_w) + \alpha\ell \\
&= B_{\ell-1}. \qquad (3.17)
\end{aligned}$$

Then, according to the optional stopping theorem [57], we have $T = \mathbb{E}[B_0] = \mathbb{E}[B_\eta] = \alpha\mathbb{E}[\eta]$. The expected stopping time is then $\frac{T}{\alpha}$, which decreases as p_w increases. To sum up, we showed that censoring BP converges w.p.1 when $L \to \infty$ and $(1-p_w)\log_K(3K+2) \le p_w$, and for finite and sufficiently large L and p_w, we derived the expected stopping time.

The advantage of this algorithm is that it only requires a small overhead; the only additional noise-free operation in this algorithm is testing if $\|m^{(\ell)} - m^{(\ell-1)}\| \le K\|m^{(\ell-1)} - m^{(\ell-2)}\|$ holds. In practice, we could set $K = 1$ to simplify the computation.

Censoring BP performs well when the hardware noise distribution (as specified in (3.10)) has a large mass at zero and has non-negligible masses at some points sufficiently away from zero. Such a distribution is found in, for example, the *soft adder model* [7]. For this noise distribution, censoring BP rejects almost all erroneous computations when the current message is close to the fixed point (that is, $\|m^{(\ell-1)} - m^{(\ell-2)}\|$ is small when the messages are both from correct computations). However, for the general case, especially when p_w is small or zero, censoring BP may fail to converge. In the next section we therefore propose the averaging BP algorithm, which applies to general zero-mean computation noise distributions.

3.2.3 The Averaging BP Algorithm

As we have seen in the preceding section, censoring BP is guaranteed to converge when we assume that p_w (cf. (3.10)) is sufficiently large. In this section, we release this assumption, consider the general zero-mean computation noise, and propose the averaging BP algorithm. From Corollary 1 and the simulation results to be discussed (Fig. 3.2b), we observe that the messages oscillate around the fixed point when zero-mean computation noise is considered. This phenomenon inspires us to propose averaging BP: instead of using $m^{(\ell)}$ in the final iteration to compute the beliefs of the random variables X_i's, we use the average over all up-to-date messages

$$\bar{m}^{(\ell)} = \frac{1}{\ell} \sum_{i=1}^{\ell} m^{(i)}, \tag{3.18}$$

to compute the beliefs. Although such operation induces a slightly larger overhead than censoring BP, it guarantees the convergence of $\bar{m}^{(\ell)}$ to the fixed point, as we will show later.

We first intuitively explain the convergence of averaging BP as follows. As observed in the simulation results in Fig. 3.2b, when the iteration number is large enough, the noisy BP messages are approximately the fixed point message plus a noise term due to the computation noise. When the noise has finite variance and is zero-mean and independent across iterations, its effects gradually disappear over subsequent iterations due to the averaging operation. The effects of the first few iteration messages, whose distances to the fixed point could be large, also diminish due to the averaging operation as the iteration number grows. Hence the average message $\bar{m}^{(\ell)}$ in (3.18) converges to the fixed point when $\ell \to \infty$. In fact, the operation in (3.18) is reminiscent of the averaging method in stochastic optimization, in which the gradient being used in the iterative optimization procedure is noisy. The convergence of the averaging method for stochastic optimization was proved in [58]. The authors in [52] applied the Robbins-Monro algorithm, which is the original form of stochastic optimization, to develop the stochastic message passing scheme, and proved its convergence. Due to the similarity of the underlying mathematical models, we are able to adapt some of the proof techniques used in [52] for our use. The convergence of the averaging BP algorithm is formally stated in the following.

Proposition 1. *Assume that the averaging operation is noise-free. Assume the following four conditions are satisfied:*

1. $\|w^{(\ell)}\|$ *is upper bounded by a finite constant,*
2. $\|\mathbb{E}[w^{(\ell)}]\| = 0$,
3. $w^{(\ell)}$*'s are i.i.d, and*
4. *entries in* $w^{(\ell)}$ *are with finite variances for all* ℓ.

We then have

$$\lim_{\ell\to\infty} \left\| \frac{1}{\ell}\sum_{i=1}^{\ell} m^{(i)} - m^* \right\| = 0. \tag{3.19}$$

That is, the average of the messages over all up-to-date iterations converges to the fixed point.

Proof. Since $\bar{m}^{(\ell)} = \frac{1}{\ell}\sum_{i=1}^{\ell} m^{(i)}$, we have

$$\begin{aligned}\bar{m}^{(\ell)} - m^* &= \frac{1}{\ell}\sum_{i=1}^{\ell} f(m^{(i-1)}) + \frac{1}{\ell}\sum_{i=1}^{\ell} w^{(i)} - m^* \\ &= \frac{1}{\ell}\sum_{i=1}^{\ell} \left(f(m^{(i-1)}) - m^*\right) + \frac{1}{\ell}\sum_{i=1}^{\ell} w^{(i)}.\end{aligned} \tag{3.20}$$

Since $\|f(m^{(i)}) - m^*\| \le K\|m^{(i)} - m^*\|$, let

$$f(m^{(i)}) - m^* = K(m^{(i)} - m^*)U_i, \tag{3.21}$$

where U_i is a rotation matrix multiplied by a positive number less than or equal to 1. That is, we shrink and rotate the vector $K(m^{(i)} - m^*)$ to make it equal to the vector $f(m^{(i)}) - m^*$. Then, for $i \geq 2$, we expand the terms $f(m^{(i-1)}) - m^*$ in (3.20) in the following:

$$\begin{aligned}
&f(m^{(i-1)}) - m^* = K(m^{(i-1)} - m^*)U_{i-1} \\
&= K(f(m^{(i-2)}) - m^* + w^{(i-1)})U_{i-1} \\
&= K^2(f(m^{(i-2)}) - m^*)U_{i-2}U_{i-1} + Kw^{(i-1)}U_{i-1} \\
&= K^i(m^{(0)} - m^*)\prod_{j=0}^{i-1} U_j + \sum_{j=1}^{i-1} K^j w^{(i-j)} \prod_{r=1}^{j} U_{i-r}. \qquad (3.22)
\end{aligned}$$

We repeatedly applied (3.21) to derive the last equality in (3.22).

$$\begin{aligned}
&\bar{m}^{(\ell)} - m^* \\
&= \frac{1}{\ell}\sum_{i=1}^{\ell}\Big(K^i(m^{(0)} - m^*)\prod_{j=1}^{i-1} U_j + \sum_{j=0}^{i-1} K^j w^{(i-j)} \prod_{r=1}^{j} U_{i-r}\Big) \\
&+ \frac{1}{\ell}\sum_{i=1}^{\ell} w^{(i)} \\
&= \frac{1}{\ell}\sum_{i=1}^{\ell} K^i(m^{(0)} - m^*)\prod_{j=0}^{i-1} U_j + \frac{1}{\ell}\sum_{i=1}^{\ell} w^{(i)} \\
&+ \frac{1}{\ell}\sum_{i=2}^{\ell}\sum_{j=0}^{i-1} K^j w^{(i-j)} \prod_{r=1}^{j} U_{i-r}. \qquad (3.23)
\end{aligned}$$

We then have

$$\begin{aligned}
\|\bar{m}^{(\ell)} - m^*\| \leq & \left\| \frac{1}{\ell}\sum_{i=1}^{\ell} K^i(m^{(0)} - m^*)\prod_{j=0}^{i-1} U_j \right\| \\
& + \left\| \frac{1}{\ell}\sum_{i=1}^{\ell} w^{(i)} \right\| + \left\| \frac{1}{\ell}\sum_{i=2}^{\ell}\sum_{j=1}^{i-1} K^j w^{(i-j)} \prod_{r=1}^{j} U_{i-r} \right\|. \qquad (3.24)
\end{aligned}$$

The first term on the R.H.S. of (3.24) converges to zero as ℓ approaches infinity since

$$\begin{aligned}
&\lim_{\ell\to\infty}\left\|\frac{1}{\ell}\sum_{i=1}^{\ell}K^i(m^{(0)}-m^*)\prod_{j=0}^{i-1}U_j\right\| \\
&\le \lim_{\ell\to\infty}\frac{1}{\ell}\sum_{i=1}^{\ell}K^i\|m^{(0)}-m^*\| \\
&= \lim_{\ell\to\infty}\frac{K(1-K^\ell)\|m^{(0)}-m^*\|}{\ell(1-K)} = 0. \qquad (3.25)
\end{aligned}$$

Since we assume that $w^{(i)}$'s are independent and their entries are with finite variances, the second term on the R.H.S. of (3.24) also converges to zero. For the third term on the R.H.S. of (3.24):

$$\begin{aligned}
&\frac{1}{\ell}\sum_{i=2}^{\ell}\sum_{j=1}^{i-1}K^j w^{(i-j)}\prod_{r=1}^{j}U_{i-r} \\
&= \frac{1}{\ell}\sum_{i'=1}^{\ell-1}\Big(\sum_{j'=1}^{i'}K^{j'}\prod_{r'=0}^{j'-1}U_{\ell-r'}\Big)w^{(\ell-i')} \\
&= \frac{1}{\ell}\sum_{i=1}^{\ell-1}\sum_{j=1}^{i}K^j\prod_{r=0}^{j-1}U_{\ell-r}w^{(\ell-i)}. \qquad (3.26)
\end{aligned}$$

The first equality in (3.26) is derived by collecting $w^{(i-j)}$ terms with the same superscript. The second equality in (3.26) is derived by replacing i',j', and r' by i, j, and r, respectively. Note that

$$\begin{aligned}
\left\|\sum_{j=1}^{i}K^j\prod_{r=0}^{j-1}U_{\ell-r}w^{(\ell-i)}\right\| &\le \sum_{j=1}^{i}K^j\|w^{(\ell-i)}\| \\
&= \frac{K(1-K^i)}{1-K}\|w^{(\ell-i)}\|. \qquad (3.27)
\end{aligned}$$

Recall that in (3.21), we express $f(m^{(i)})-m^*$ as $K(m^{(i)}-m^*)U_i$ since $\|f(m^{(i)})-m^*\| \le K\|m^{(i)}-m^*\|$. Here, since we have the inequality (3.27), we can likewise express $\sum_{j=1}^{i}K^j\prod_{r=0}^{j-1}U_{\ell-r}w^{(\ell-i)}$ as $\frac{K(1-K^i)}{1-K}w^{(\ell-i)}U_i'$, where U_i' is a rotation matrix multiplied by a positive number less than or equal to 1. We then have

$$\frac{1}{\ell}\sum_{i=1}^{\ell-1}\Big(\sum_{j=1}^{i}K^j\prod_{r=0}^{j-1}U_{\ell-r}w^{(\ell-i)}\Big) = \frac{1}{\ell}\sum_{i=1}^{\ell-1}\frac{K(1-K^i)}{1-K}w^{(\ell-i)}U_i'. \qquad (3.28)$$

Therefore,

$$\frac{1}{\ell}\sum_{i=1}^{\ell-1}\frac{K(1-K^i)}{1-K}w^{(\ell-i)}U_i' = \frac{K}{\ell(1-K)}\sum_{i=1}^{\ell-1}w^{(\ell-i)}U_i' + \frac{1}{\ell(1-K)}\sum_{i=1}^{\ell-1}K^{i+1}w^{(\ell-i)}U_i'. \tag{3.29}$$

The norm of the second term on the R.H.S. of (3.29) converges to zero since

$$\lim_{\ell\to\infty}\left\|\frac{1}{\ell(1-K)}\sum_{i=1}^{\ell-1}K^{i+1}w^{(\ell-i)}U_i'\right\| \leq \lim_{\ell\to\infty}\frac{K^2(1-K^{\ell-1})\max_j\|w^{(j)}\|}{\ell(1-K)^2} = 0. \tag{3.30}$$

We used the fact that $\|w^{(j)}\|$ is upper bounded. Finally, we discuss the convergence of the norm of the first term on the R.H.S. of (3.29). Here we assume that $w^{(\ell-i)}U_i'$'s are independent. We cannot rigorously reason the validity of the independence assumption, but we draw support from an argument provided in [47]. In Proposition 17 of [47] (Section 5.5), a similar assumption was made on "message errors", i.e., the differences between BP messages and the fixed point messages in every iteration. As was mentioned in [47], similar assumptions yield useful analysis of the effects of quantization (which can be considered as a type of computation noise) in digital processing systems such as digital filters, and empirically such systems often behave close to the predictions made under such assumptions [59]. In fact, simulation results for Ising model demonstrate that averaging BP indeed converges to the fixed point, as predicted in this proposition. Since $w^{(\ell-i)}U_i'$'s are independent, the first term on the R.H.S. of (3.29) converges to zero as $\ell\to\infty$. Note that even if $w^{(\ell-i)}U_i'$'s are not independent, as long as they are not "too much positively correlated" and the growth of $\sum_{i=1}^{\ell}w^{(\ell-i)}U_i'$ is slower than linear w.r.t. ℓ, the first term on the R.H.S. of (3.29) still converges to zero as $\ell\to\infty$. Therefore, the three terms on the R.H.S. of (3.24) all converges to zero as $\ell\to\infty$, and this completes our proof. $\square$

Next, we derive an upper bound on the expectation of the L_1 norm of the difference between $\bar{m}^{(\ell)}$ and the fixed point, $\|\bar{m}^{(\ell)} - m^*\|_1$, w.r.t. ℓ.

Proposition 2. *Assume that the computation noise $\|w^{(\ell)}\|_\infty \leq \beta$, where $\beta > 0$. If $w^{(\ell)}$ satisfies the four conditions in Proposition 1, the expectation of the L_1 norm of the difference between $\bar{m}^{(\ell)}$ and the fixed point m^* is upper bounded as*

$$\begin{aligned} \mathbb{E}[\|\bar{m}^{(\ell)} - m^*\|_1] \leq \frac{K(1-K^\ell)\|m^{(0)} - m^*\|_1}{\ell(1-K)} \\ + \beta\Gamma\Big(\frac{K^2(1-K^{\ell-1})}{\ell(1-K)^2} + \sqrt{\frac{2\pi}{\ell}}\frac{K}{1-K}\Big), \end{aligned} \tag{3.31}$$

where Γ is the dimension [3] *of $w^{(\ell)}$.*

Proof. We first show that the sequence $\{w^{(\ell)}(j)\}_{\ell=0}^\infty$ has the martingale difference property [57] for every j. Let the σ-field $\mathcal{F}_\ell$ be the σ-field on $\{w^{(1)}(j), \ldots, w^{(\ell)}(j)\}$. Since we assume that the computation noise is zero-mean and independent across different iterations, we have $\mathbb{E}[w^{(\ell+1)}(j)|\mathcal{F}_\ell] = 0$. We also assume that the entries in $w^{(\ell)}$'s are bounded. Hence the sequence $\{w^{(\ell)}(j)\}_{\ell=0}^\infty$ forms a bounded martingale difference sequence. Given the assumption that $|w^{(i)}(j)| \leq \beta$ (as stated in the proposition), we apply the Azuma-Hoeffding inequality [60] to yield the bound

$$P\left(\frac{1}{\ell}\left|\sum_{i=1}^{\ell} w^{(i)}(j)\right| > \gamma\right) \leq 2\exp\left(-\frac{\ell\gamma^2}{2\beta^2}\right), \tag{3.32}$$

for all $\gamma > 0$. We upper bound the mean of $\frac{1}{\ell}|\sum_{i=1}^{\ell} w^{(i)}(j)|$ by integrating both sides of (3.32):

$$\begin{aligned} \mathbb{E}\left[\frac{1}{\ell}\left|\sum_{i=1}^{\ell} w^{(i)}(j)\right|\right] &= \int_0^\infty P\left(\frac{1}{\ell}\left|\sum_{i=1}^{\ell} w^{(i)}(j)\right| > \gamma\right) d\gamma \\ &\leq \beta\sqrt{\frac{2\pi}{\ell}}. \end{aligned} \tag{3.33}$$

For Γ, the dimension of $w^{(i)}$, we have

$$\mathbb{E}\left[\frac{1}{\ell}\left\|\sum_{i=1}^{\ell} w^{(i)}\right\|_1\right] \leq \beta\Gamma\sqrt{\frac{2\pi}{\ell}}. \tag{3.34}$$

[3] $\Gamma = d^{\|E\|}$, *d is the size of $\mathcal{X}$ and $\|E\|$ is the number of edges in the factor graph.*

From (3.24), (3.25), (3.29), and (3.30) in the proof of Proposition 1, we have

$$\begin{aligned}
&\|\bar{m}^{(\ell)} - m^*\|_1 \\
&\leq \frac{K(1-K^\ell)\|m^{(0)} - m^*\|_1}{\ell(1-K)} + \frac{K^2(1-K^{\ell-1})\beta\Gamma}{\ell(1-K)^2} \\
&+ \left\|\frac{1}{\ell}\sum_{i=1}^{\ell} w^{(i)}\right\|_1 + \left\|\frac{K}{\ell(1-K)}\sum_{i=1}^{\ell-1} w^{(\ell-i)}U_i'\right\|_1 .
\end{aligned} \tag{3.35}$$

As in the proof of Proposition 1, here we also assume that the vectors $w^{(\ell-1)}U_1'$, $w^{(\ell-2)}U_2'$, ..., $w^{(1)}U_{\ell-1}'$ are independent. Their entries then also form a martingale difference sequence. Therefore, by substituting (3.34) in (3.35), we have

$$\begin{aligned}
&\mathbb{E}[\|\bar{m}^{(\ell)} - m^*\|_1] \\
&\leq \frac{K(1-K^\ell)\|m^{(0)} - m^*\|_1}{\ell(1-K)} + \frac{K^2(1-K^{\ell-1})\beta\Gamma}{\ell(1-K)^2} \\
&+ \Big(\frac{K}{1-K} + 1\Big)\beta\Gamma\sqrt{\frac{2\pi}{\ell}} \\
&\leq \frac{K(1-K^\ell)\|m^{(0)} - m^*\|_1}{\ell(1-K)} \\
&+ \beta\Gamma\Big(\frac{K^2(1-K^{\ell-1})}{\ell(1-K)^2} + \sqrt{\frac{2\pi}{\ell}}\frac{1}{1-K}\Big).
\end{aligned} \tag{3.36}$$

This completes our proof. $\square$

The previous proposition shows that the decay of the upper bound for $\mathbb{E}[\|\bar{m}^{(\ell)} - m^*\|_1]$ is roughly $\sqrt{\frac{1}{\ell}}$ for sufficiently large ℓ. To accelerate the convergence, we can start the averaging operation from $i = i^*, i^* > 1$ in (3.19) instead of $i = 1$, since the first few messages might be far away from the fixed point, slowing down the convergence in our averaging operations.

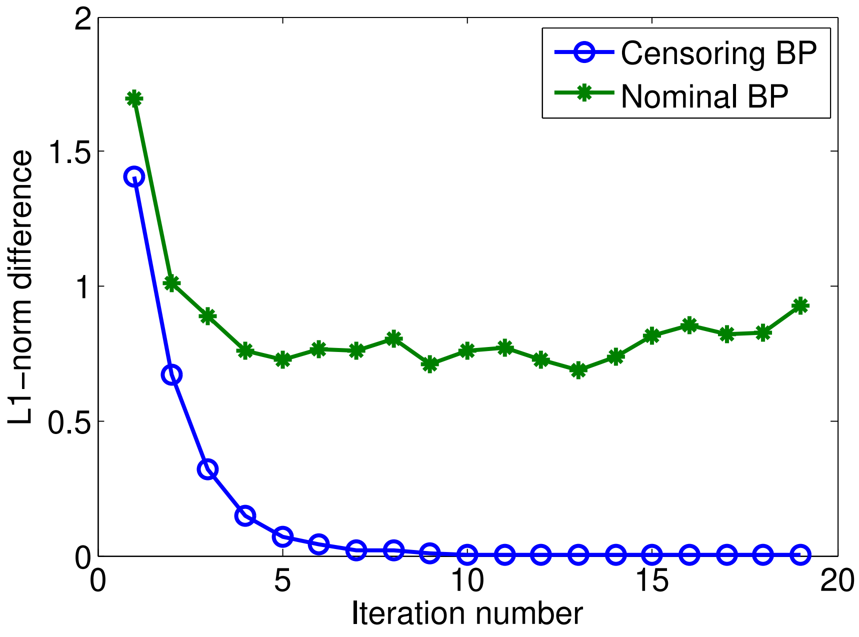
Although the overhead of averaging BP is slightly larger than censoring BP, the overhead of averaging BP is still very small compared to other fault-tolerant computing approaches such as Triple Modular Redundancy [3]. The only required noise-free operation in the averaging BP is the averaging operation. Even the summation operation in (3.18) is allowed to be noisy, since the noise-free division will help diminish the effects of computation noise in the summation operation. Moreover, the averaging operation is done off-line, that is, the message update does not rely on the average message $\bar{m}^{(\ell)}$. Therefore, instead of performing the averaging operation at every iteration, we can just perform the averaging operation only when we want to compute the beliefs of the variable nodes. Note that $\bar{m}^{(\ell)}$ (see (3.18)) is computed from the sum of $m^{(i)}$'s. Therefore, instead of storing the messages in past iterations separately, we only need to store the sum of the messages up to the current iteration. As a consequence, the number of additional memory cells required can be kept constant.

3.2.4 Experiments on Ising Model

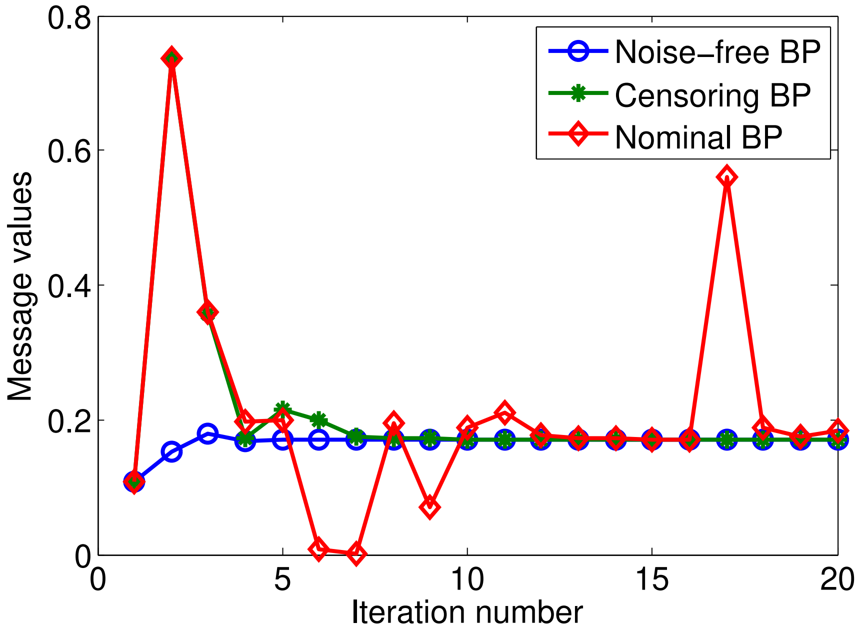
In this section, we compare the convergence of the proposed algorithms and the nominal BP algorithm under different computation noise models. We use a fully connected, uniform coupling, and uniform local field Ising model. In the Ising model, all the potentials consist of at most two variables, and all the N random variables in the Ising model are binary. Denote the set of factors as $\mathcal{F} = \mathcal{F}_1 \cup \mathcal{F}_2$, where $\mathcal{F}_1 = \mathcal{V}$ are the factors connected to one variable node, and $\mathcal{F}_2 \subseteq \{(i,j) : i,j \in \mathcal{V}, i \neq j\}$ are the factors connected to two variable nodes. The joint distribution of the variables is

$$P_X(x) = \frac{1}{Z} \exp\Big(\sum_{(i,j)\in\mathcal{F}_2} J_{i,j} x_i x_j + \sum_{i\in\mathcal{F}_1} \theta_i x_i \Big). \tag{3.37}$$

For the uniform coupling and uniform local field Ising models, all $J_{i,j}$'s are the same (denote their common value by J) and all θ_i's are also the same (denote their common value by θ). The J and θ ranges that satisfy the contraction mapping condition for different N values were derived in [48], and we select $N = 4, J = 0.3$ and $\theta = -0.3$ in our experiments accordingly.



(a)



(b)

Figure 3.1: Censoring BP: (a) L_1 norm distance comparison. (b) Belief trajectory comparison.

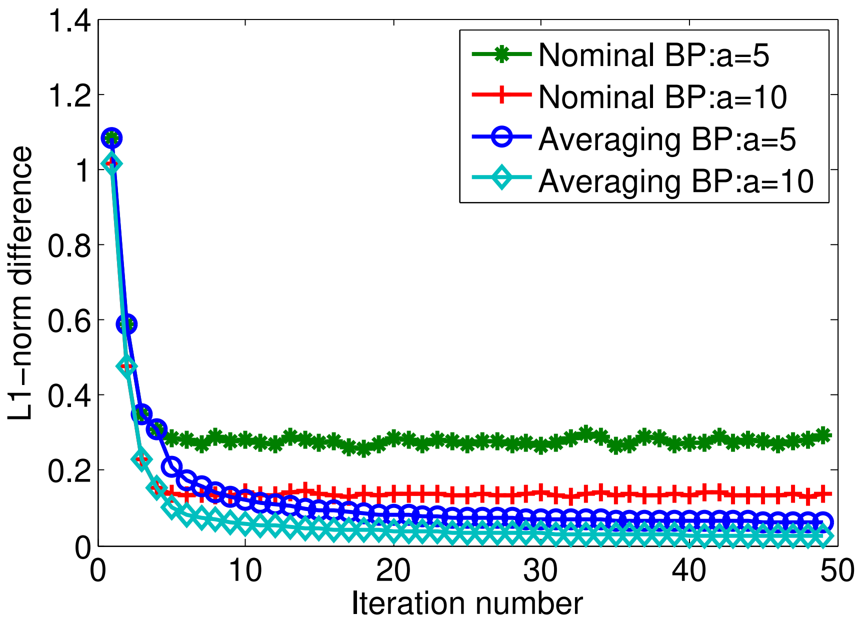
3.2.4.1 Censoring BP on Ising model

For censoring BP, we consider the computation noise model adapted from the soft adder model proposed in [7]: the computation noise is either zero or has a large magnitude. To be more specific, the noise distribution follows (3.10), with $p_w = 0.95$ and $h(\phi)$ being a mixed Gaussian distribution with variance $\frac{\gamma}{4}$ and mean 0.5γ with probability 0.5 and mean -0.5γ with probability 0.5, where γ is the magnitude of the message that the noise is added to. Since the messages in BP are always positive, we set the noisy messages to be the absolute values of the sum of the noise-free messages and the computation noise, namely, $m^{(\ell)} = |f(m^{(\ell-1)}) + w^{(\ell)}|$.

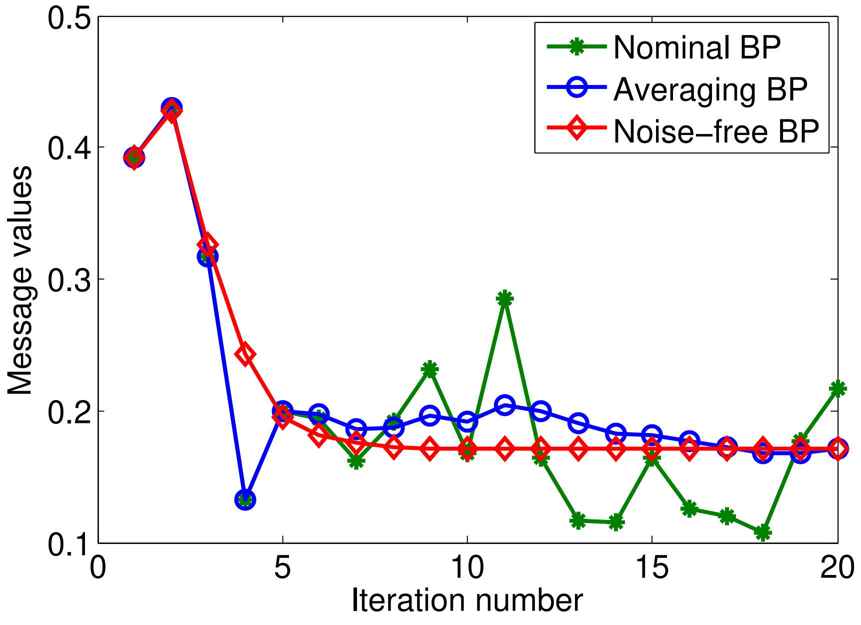
We compare the L_1 norm distance of the beliefs computed by censoring BP to the beliefs computed by noise-free BP with the L_1 norm distance of the beliefs computed by nominal BP to the beliefs computed by noise-free BP. We show the average L_1 norm distances in each iteration over 200 simulations in Fig. 3.1a. The average distance between censoring BP and noise-free BP decreases to zero as the iteration number grows, while the average distance between nominal BP and noise-free BP remains large even when the maximum iteration number (20 in this example) is reached. We also show the trajectory of the beliefs of variable node 1 w.r.t. the iteration number in a single simulation in Fig. 3.1b. We observe that although censoring BP could make mistakes in the first few iterations, as the beliefs get closer to the fixed point, most of the erroneous computations are rejected, and censoring BP successfully converges to the fixed point.

3.2.4.2 Averaging BP on Ising model

For averaging BP, we consider the zero-mean computation noise with $p_w = 0$ (cf. (3.10)). To be more specific, we consider the case where the entries of $w^{(\ell)}$ are uniformly distributed in $[-\frac{\gamma}{a}, \frac{\gamma}{a}]$, where γ is the magnitude of the message that



(a)



(b)

Figure 3.2: Averaging BP: (a) L_1 norm distance comparison. (b) Belief trajectory comparison ($a = 5$).

the noise is added to and a is greater than 1 to ensure the message is larger than zero. The maximum message value is 1. Therefore, the noisy message is expressed as $m^{(\ell)} = (f(m^{(\ell-1)}) + w^{(\ell)} - 1)^{-} + 1$, where $(\zeta)^{-}$ denotes the negative part of ζ, i.e., the message is clipped at 1.

We compare the L_1 norm distance of the beliefs computed by averaging BP to the beliefs computed by noise-free BP with the distance of the beliefs computed by nominal BP to the beliefs computed by noise-free BP. We use two different computation noise distributions for comparison, namely, $a = 5$ and $a = 10$, and show the average L_1 distances in each iteration over 200 simulations (Fig. 3.2a). We observe that the average distance between averaging BP and noise-free BP is very close to zero after a sufficiently large number of iterations. The increase in computation noise variance only slightly increases the distance to the noise-free BP message for averaging BP. We also show that the trajectories of the beliefs of variable node 1 in a sampled simulation in Fig. 3.2b (up to 20 iterations). We observe that the beliefs computed by averaging BP quickly converges to the fixed point, while nominal BP beliefs keep oscillating around the fixed point. To test averaging BP under computation noise with different distributions, we also run simulations for averaging BP under Gaussian distributed computation noise, and the results are qualitatively the same as the uniform distributed noise, thus showing that the shape of computation noise distribution has limited effect on averaging BP performance. We thus conclude that the proposed algorithms successfully converge to the fixed point, while the nominal BP algorithm fails to do so.

3.3 Application Example I:Image Denoising

BP can be applied to image denoising where we seek to recover the original image from the noise-contaminated observation of the image. We briefly explain

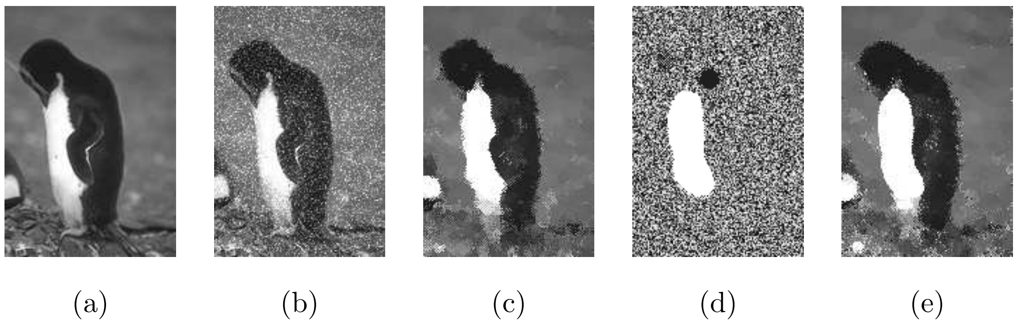


Figure 3.3: Image denoising via BP: (a) original image (b) contaminated image (c) recovered image by noise-free BP (d) recovered image by nominal BP (e) recovered image by averaging BP.

a simple BP-based image denoising algorithm as follows. Consider a 2-D image. For every color on the image, we assign a unique number to the color. The value of each pixel on the image is then the number corresponding to the pixel's color. In a 2-D image, our observation of each pixel value is contaminated by additive noise. Assume that the noises added to different pixels are independent. To recover the image from the contaminated pixel value observations, we construct a 2-D grid with an appropriate choice of potential functions (e.g., Potts model) as our factor graph. Each pixel in the image is then associated to a variable node on the factor graph. Each message is a vector representing the probabilities of all the possible colors of the associated pixel. To be more specific, suppose we are on a 256-level grayscale; a message associated with a variable node is a vector with 256 entries. Each entry is the probability that the pixel associated with the variable node is at a darkness level corresponding to the entry. We run the BP algorithm on the factor graph to obtain the most likely value of every pixel based on the contaminated observations.

We use the "penguin" image (with size 179×122 and 256 gray-scale levels) from [61], and associate the 2-D grid constructed for the image to a Potts model [61] with coefficient $\rho = 0.001$. In a Potts model, the potential function ψ_I of a factor connecting two variable nodes is 1 for the two variables with the same value, and ρ for the two variables with different values. We assign the integers 0 to 255 to the 256 gray-scale levels. Every pixel value in the image is contaminated with an independent Gaussian random variable with zero-mean and standard deviation 50. The original image and the contaminated image are shown in Fig. 3.3(a) and Fig. 3.3b. We use the computation noise model in Section 3.2.4.2, and run the nominal BP and averaging BP algorithm for 20 iterations to recover the original image from the contaminated one.

Images recovered by noise-free BP, nominal BP, and averaging BP are shown in Fig. 3.3c, Fig. 3.3d, and Fig. 3.3e, respectively. Comparing Fig. 3.3d and Fig. 3.3e, we observe that for the nominal BP, most of the pixels are determined incorrectly and the recovered image is very different from the image recovered using noise-free BP (Fig. 3.3c). On the other hand, the image recovered by averaging BP is very close to the image recovered by noise-free BP (Fig. 3.3c). We then conclude that averaging BP indeed improves the image denoising BP on noisy hardware.

3.4 Application Example II: BP Decoder for LDPC Codes

3.4.1 Noisy BP Decoder

BP can be applied to decode the messages transmitted over a communication channel. In Chapter 2, we study finite precision decoders implemented on noisy hardware. In this section, we consider the BP decoder, which is an infinite precision decoder. Following the noisy decoder model in Chapter 2, in addition to receiving a noisy input from a communication channel, the noisy BP decoder is also subject to errors arising from noisy hardware. According to the BP computation noise model (3.6), the messages exchanged in our noisy BP decoder at iteration ℓ are represented as:

$$m_{v,c}^{(\ell)} = \hat{m}_{v,c}^{(\ell)} + \breve{w}_{v,c}^{(\ell)},\ m_{c,v}^{(\ell)} = \hat{m}_{c,v}^{(\ell)} + \breve{w}_{c,v}^{(\ell)}, \tag{3.38}$$

where $\hat{m}_{v,c}^{(\ell)}$ and $\hat{m}_{c,v}^{(\ell)}$ are the noise-free messages, and $\breve{w}_{v,c}^{(\ell)}$ and $\breve{w}_{c,v}^{(\ell)}$ are the computation noises. Note that since the computation noise is additive and the variable node operation is simply the summation of all incoming messages, we can combine the computation noise in check nodes and a variable node into a single term. Then, in our noisy BP decoder, a check node is the same as a check node in the noise-free decoder, and the variable node operation becomes

$$m_{v,c}^{(\ell)} = m_{v,c}^{(0)} + \sum_{c' \in \mathcal{N}_v \backslash \{c\}} \hat{m}_{c',v}^{(\ell-1)} + w_{v,c}^{(\ell)}, \tag{3.39}$$

where $w_{v,c}^{(\ell)} = \breve{w}_{v,c}^{(\ell)} + \sum_{c' \in \mathcal{N}_v \backslash \{c\}} \breve{w}_{c',v}^{(\ell)}$ is the aggregate computation noise. We assume that all $w_{v,c}^{(\ell)}$'s are identically distributed and independent across ℓ. Note that $m_{v,c}^{(0)}$ is the same as $\tilde{m}_{v,c}^{(0)}$ since we have the same communication channel for the noise-free and noisy decoders.

3.4.2 Density Evolution and Decoding Threshold Analysis of a Noisy BP Decoder

Following the transient error model in Chapter 2, we assume that $w_{v,c}^{(\ell)}$'s are symmetric. We also assume that the underlying bipartite graph for our decoder is cycle-free. Since $w_{v,c}^{(\ell)}$'s are symmetric, according to [36], the error behavior of a noisy decoder is independent of the chosen codeword. We therefore assume that the all-zero codeword is transmitted.

We use Gaussian approximate density evolution techniques [62] to analyze the noisy BP decoder. Gaussian approximate density evolution for the noise-free BP decoders utilizes the facts that (a) when d_v is sufficiently large, the distribution of the summation of check-to-variable messages is close enough to a Gaussian distribution (by the central limit theorem), and (b) when the communication channel noise is Gaussian, the log-likelihood ratios of the decoder input are Gaussian with mean $\frac{2}{\sigma_{ch}^2}$ and variance $\frac{4}{\sigma_{ch}^2}$, where σ_{ch}^2 is the variance of the communication channel noise. Experimental results confirm the validity of such approximation [62]. Applying the Gaussian approximate density evolution to the noisy BP decoder, we have the following lemma:

Lemma 4. *If d_v is sufficiently large, and if one of the two following conditions holds: (c) $\breve{w}_{c,v}^{(\ell)}$ and $\breve{w}_{v,c}^{(\ell)}$ in* (3.38) *are identically distributed with zero mean, (d) $\breve{w}_{v,c}^{(\ell)}$ in* (3.38) *is Gaussian with zero mean, then the distribution of $m_{v,c}^{(\ell)}$ can be approximated by the Gaussian distribution with mean $\frac{2}{\sigma_{ch}^2} + \mu_{\ell-1}$ and variance $\frac{4}{\sigma_{ch}^2} + \sigma_{\ell-1}^2 + \sigma_{cmp}^2$, where σ_{cmp}^2 is the variance of $w_{v,c}^{(\ell)}$, and $\mu_{\ell-1}$ and $\sigma_{\ell-1}^2$ are mean and variance of $\sum_{c' \in \mathcal{N}_v \setminus \{c\}} \hat{m}_{c',v}^{(\ell-1)}$, respectively.*

Proof. When d_v is sufficiently large and when condition (c) or (d) holds, by the central limit theorem, the aggregate computation noise, $\breve{w}_{v,c}^{(\ell)} + \sum_{c' \in \mathcal{N}_v \setminus \{c\}} \breve{w}_{c',v}^{(\ell-1)}$, can be approximated by a Gaussian random variable with zero mean and variance σ_{cmp}^2. Then, given facts (a) and (b) mentioned in the preceding paragraph, the initial message $m_{v,c}^{(0)}$ is Gaussian distributed with mean $\frac{2}{\sigma_{ch}^2}$ and variance $\frac{4}{\sigma_{ch}^2}$, and $\sum_{c' \in \mathcal{N}_v \setminus \{c\}} \hat{m}_{c',v}^{(\ell-1)}$ can be approximated by a Gaussian random variable with its mean and variance denoted by $\mu_{\ell-1}$ and $\sigma_{\ell-1}^2$ respectively. Relying on the fact that the sum of independent Gaussian random variables is still a Gaussian random variable, with mean and variance being the sums of the means and variances of the summand variables, the distribution of $m_{v,c}^{(\ell)}$ is then approximated by a Gaussian distribution with mean $\frac{2}{\sigma_{ch}^2} + \mu_{\ell-1}$ and variance $\frac{4}{\sigma_{ch}^2} + \sigma_{\ell-1}^2 + \sigma_{cmp}^2$. $\square$

We adapt the semi-Gaussian method, a variant of the Gaussian approximation method, proposed in [63] to analyze the density evolution of our noisy BP decoder according to Lemma 4. The quantities $\mu_{\ell-1}$ and $\sigma_{\ell-1}^2$ in the semi-Gaussian method are derived by Monte-Carlo simulations, and the residual error rates are derived from the approximated Gaussian distributions. Details of the semi-Gaussian

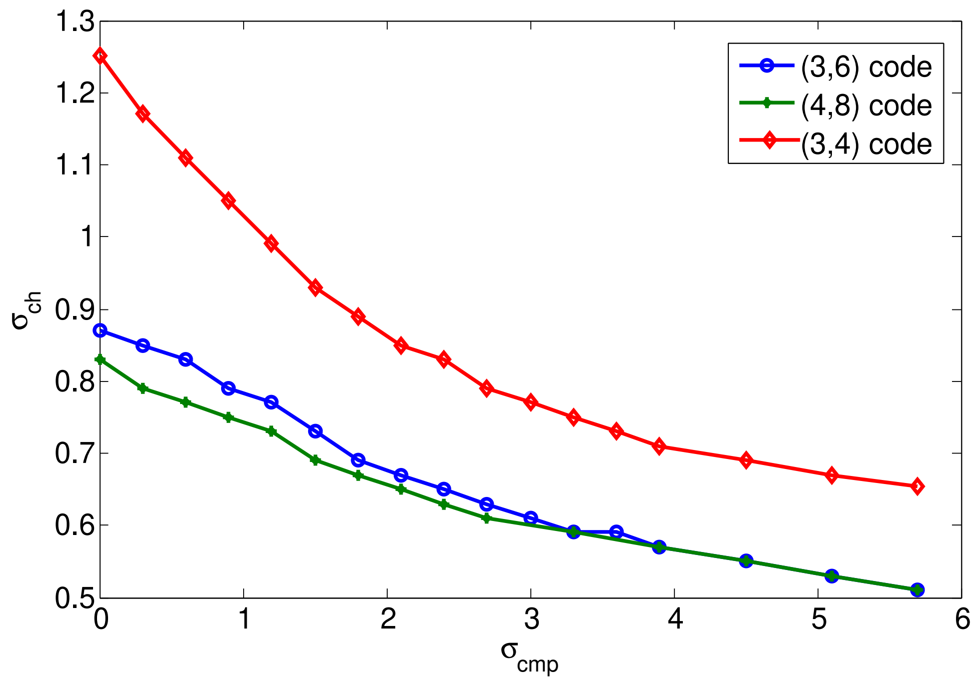


Figure 3.4: Decoding threshold pairs for different regular LDPC codes.

method can be found in [63]. For the noisy BP decoder under Gaussian computation noise and the infinite-precision hardware (i.e., messages of arbitrarily large dynamic range are represented with arbitrarily high precision), we observe from the density evolution analysis that the decoder approaches zero error probability if the communication channel noise and the computation noise are small. This observation was also made in [36], and is explained as follows. The iterative decoding process is able to raise the message values (log-likelihood ratios) to infinity when the communication channel noise is small enough. Since the magnitude of Gaussian computation noise is finite w.p.1, when the messages are large enough (i.e., close to infinity), the effects of the computation noise diminish.

Extending the definition of the decoding threshold for the noise-free case (see [39]), we define the decoding threshold for a noisy BP decoder as a curve $\sigma_{ch} = \sigma^*_{ch}(\sigma_{cmp})$ on the $\sigma_{ch} - \sigma_{cmp}$ plane such that perfect decoding is achievable whenever $\sigma_{ch} < \sigma^*_{ch}(\sigma_{cmp})$ and unachievable whenever $\sigma_{ch} > \sigma^*_{ch}(\sigma_{cmp})$.

Reversely, the curve can also be expressed as $\sigma_{cmp} = \sigma^*_{cmp}(\sigma_{ch})$, and perfect decoding is achievable whenever $\sigma_{cmp} < \sigma^*_{cmp}(\sigma_{ch})$ and unachievable whenever $\sigma_{cmp} > \sigma^*_{cmp}(\sigma_{ch})$.

In Fig. 3.4, we plot the decoding threshold curve derived from the Gaussian approximate density evolution for different (d_v, d_c)-regular LDPC codes. It is clear that σ^*_{ch} decreases as σ_{cmp} increases. Therefore, we conclude from the decoding threshold analysis that the communication channel noise level must be lowered if the computation noise is large, so that the joint effect of the two types of noises does not prohibit perfect decoding.

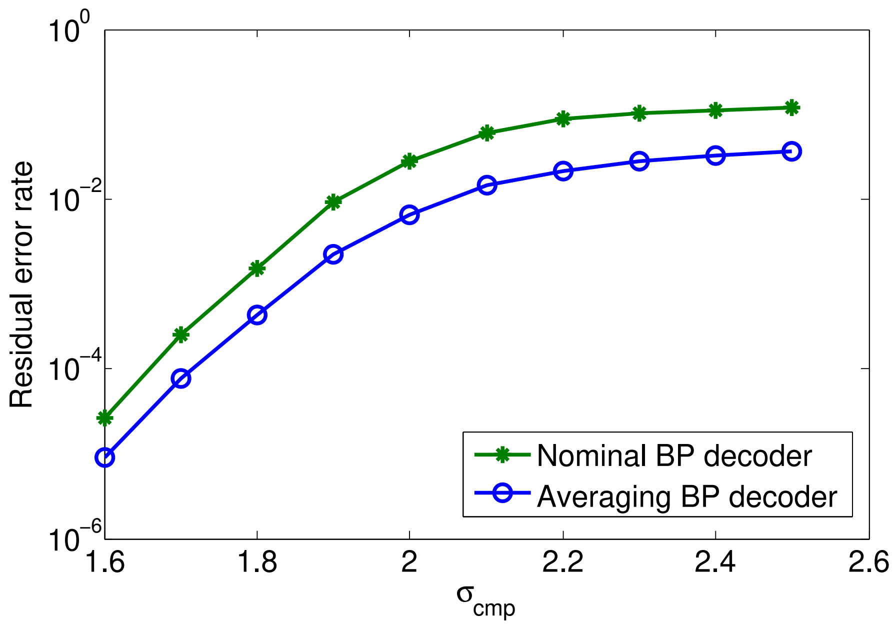
3.4.3 Averaging BP Decoder

Under noise of moderate variance values, the residual error rate is bounded away from zero (i.e., when the $(\sigma_{cmp}, \sigma_{ch})$ pair is above and to the right of the curve in Fig. 3.4). In this section, we focus on the BP decoder operating under noise of moderate variance values and propose a robust decoder implementation based on averaging BP proposed in Section 3.2.3. To be more specific, when we terminate the decoding process at the L'th iteration, we compute the average message, $\bar{m}_v^{(L)}$, as

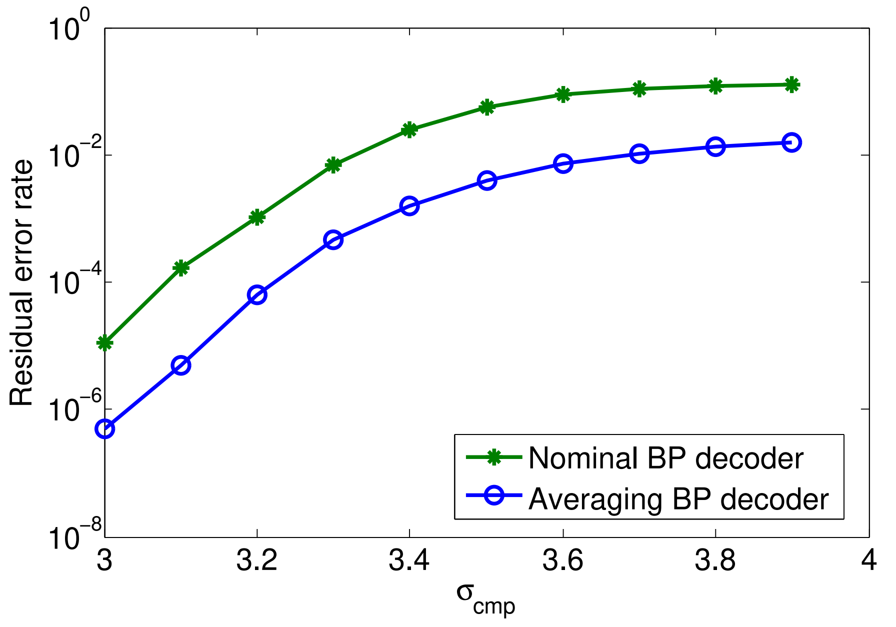
$$\bar{m}_v^{(L)} = \frac{1}{\ell} \sum_{i=1}^{L} m_v^{(i)}. \tag{3.40}$$

The decoder output is then computed by $\bar{m}_v^{(L)}$. We call this decoder the averaging BP decoder. In fact, since we consider binary LDPC codes in this section, the decoded bits are determined only by the signs of $\bar{m}_v^{(L)}$'s. We therefore can eliminate $\frac{1}{\ell}$ from (3.40); the decoded bits are specified only by $\sum_{i=1}^{\ell} m_v^{(i)}$.

We compare the residual error rates of the averaging BP decoder with the nominal BP decoder under the noisy decoder model (3.39) by simulations to demonstrate the advantages of the averaging BP decoder. We use a $(3, 6)$-regular LDPC code with code length 2640 [41]. The Gaussian communication channel noise has



(a)



(b)

Figure 3.5: Residual error rate comparison (a) $\sigma_{ch} = 0.7$. (b) $\sigma_{ch} = 0.6$.

zero-mean and variance σ_{ch}^2. As an example, we consider $\sigma_{ch} = 0.6$ and $\sigma_{ch} = 0.7$. The range of σ_{cmp} is chosen to be an interval starting near $\sigma_{cmp}^*(\sigma_{ch})$ to show the effects of increasing computation noise on the residual error rates. The number of iterations in our simulations is 30. Note that for finite length LDPC codes, the bipartite graphs may have cycles. The contraction mapping condition is not satisfied.[4]

Simulation results are shown in Fig. 3.5a and Fig. 3.5b. We observe that the averaging BP decoder achieves lower residual error rates than the nominal BP decoder for the whole σ_{cmp} interval we simulated and for both communication channel noise cases. The residual error rate reduction reaches 20X in the low computation noise region with $\sigma_{ch} = 0.6$. We then conclude that averaging BP significantly reduces the residual error rate of the noisy BP decoder.

When σ_{cmp} drops below σ_{cmp}^* (with a given σ_{ch}), with enough iterations, we observe that the residual error rates of both the nominal and the averaging BP decoders quickly approach zero. Interestingly, with small σ_{cmp}, the residual error rate of the nominal BP decoder converges faster to zero than the residual error rate of the averaging BP decoder. For example, when $\sigma_{cmp} = 1.4$ and $\sigma_{ch} = 0.7$, the nominal BP decoder residual error rate drops below 10^{-8} at iteration 19, while the averaging BP decoder residual error rate goes under 10^{-8} after iteration 25. In fact, when σ_{cmp} is small enough (below σ_{cmp}^* with a given σ_{ch}), the noisy BP decoder behavior is similar to that of the noise-free BP decoder (all noisy messages approach infinity as $L \to \infty$). In this case, the averaging operation slows down the increase in message magnitudes, and in turn decoder convergence.

We additionally performed simulations under fixed computation noise ($\sigma_{cmp} =$

[4]Since the derivative of $\tanh^{-1} x$ (part of the check node message update) is $\frac{1}{1-x^2}$, which approaches infinity when x is close to 1, the contraction mapping condition ($\sup_{u \in U} \|f'(u)\| < 1$) is not satisfied for BP decoders. Therefore, in the finite length case, the averaging BP decoder does not necessarily converge to the noise-free BP decoder. However, in the following simulations, we demonstrate that even when the contraction mapping condition is not satisfied, applying averaging BP can still improve the noisy decoder performance by canceling the effects of computation noise via the averaging operation.

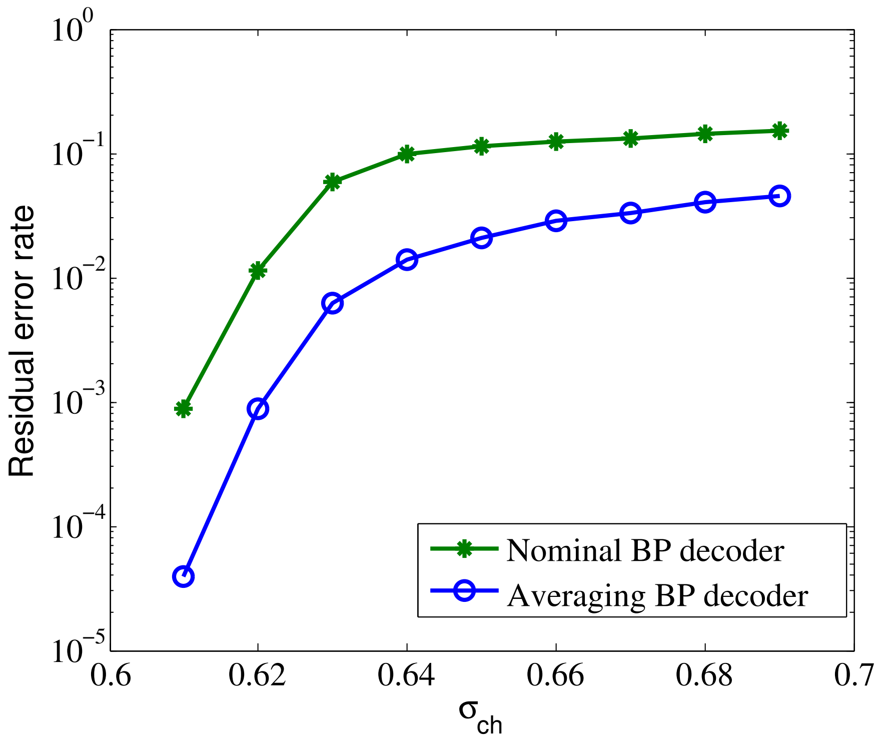
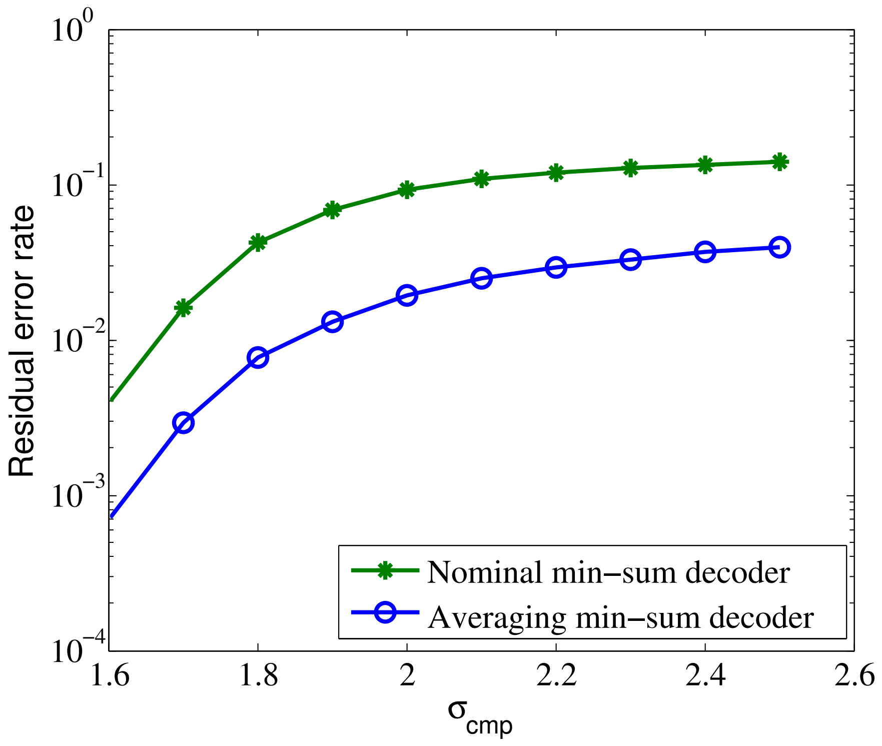


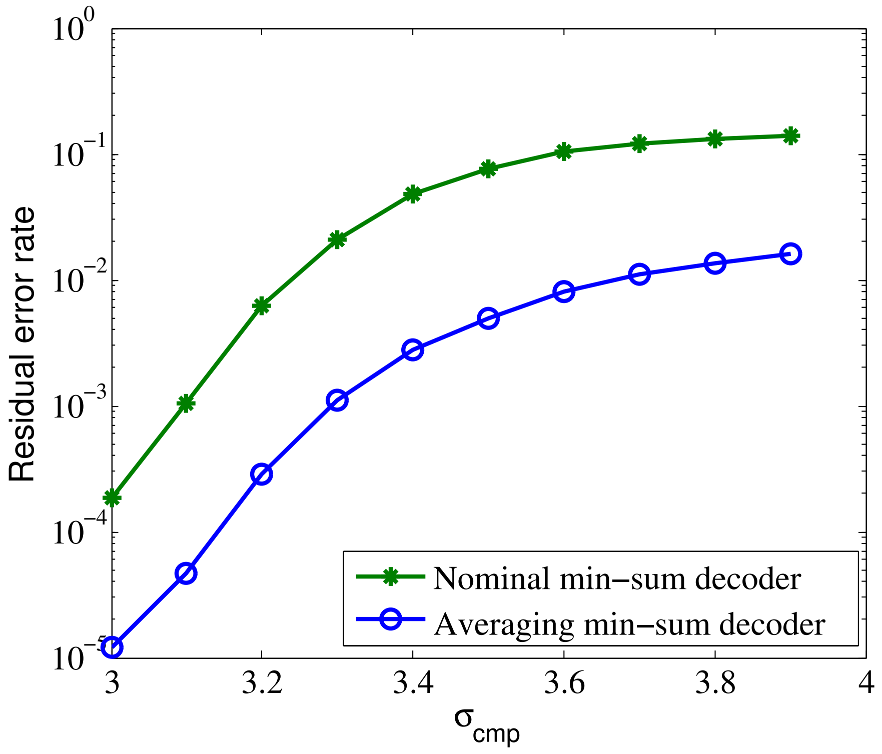
Figure 3.6: Residual error rate comparison with $\sigma_{cmp} = 3$.

3) as the variance of communication channel noise varies. The results are shown in Fig. 3.6. We observe that larger communication channel noise results in larger residual error rates, and that the averaging BP decoder outperforms the nominal BP decoder in the entire range of σ_{ch} we considered (similar to what we observed in the cases with fixed σ_{ch} and varying σ_{cmp}).

Note that although we consider the BP decoder in this section, we also observe from simulations that applying averaging BP can also reduce the residual error rates of the lower-complexity LDPC decoders, such as the min-sum decoder. Simulations results of min-sum decoders are shown in Fig. 3.7a and 3.7b.



(a)



(b)

Figure 3.7: Residual error rate comparison for min-sum decoders (a) $\sigma_{ch} = 0.7$. (b) $\sigma_{ch} = 0.6$.

CHAPTER 4

Adaptive Coding for Approximate Computing on Faulty Memories

In this chapter, we propose the *A*daptive *Co*ding for approximate *Co*mputing (ACOCO) framework, which provides us with a theory-guided design methodology to develop adaptive codes for different computations on the data read from faulty memories. The robust algorithms developed in Chapter 2 and 3 aim at achieving the performance of inference algorithms on noise-free hardware. Following the approximate computing design methodology, in this chapter, we consider the applications that can tolerate small errors and develop robust algorithms (code design methods) with better implementation efficiency by relaxing the computation accuracy constraint.

In ACOCO, we first compress the data by introducing distortion in the source encoder, and then add redundant bits to protect the data against memory errors in the channel encoder; thus we are able to protect the data against memory errors without additional memory overhead. We develop adaptive codes for two types of systems under ACOCO. The first type of systems we consider, which includes many machine learning and graph-based inference systems, are the systems dominated by product operations. Next, we consider another type of systems: iterative decoders with min operation and sign-bit decision, which are widely applied in wireless communication systems.

4.1 Background and Previous Work

Approximate computing is a class of techniques that relax the requirement of exact numerical or Boolean equivalence between the specification and implementation of a computing platform, in order to achieve improvements in performance or energy efficiency [20, 64]. Approximate computing techniques can be applied to the systems that are able to produce acceptable outputs despite some of its underlying computations being executed incorrectly in the underlying hardware. Those systems usually have the following properties: (i) the input data set is collected in real-world and has significant redundancy, e.g., camera collected data for image and video processing applications, (ii) the systems produce outputs which are eventually consumed by humans and we, humans, have our own perceptual limitations and small deviations in the output cannot be perceived by users [22]. Approximate computing has been studied in the context of traditional CPUs and proposed new programming models, compiler systems, and run-time systems to manage approximation [65, 66]. Approximate computing for GPUs was recently studied in [21]. Besides developing approximate computing techniques, an analysis methodology for approximate computing was proposed in [20].

As mentioned in Chapter 1, error correcting codes for faulty memory were widely studied in many previous works [9, 14, 15]. In a recent work [67], a novel memory protection scheme discarding several least significant bits (LSBs) before applying error-correcting codes was proposed, and this scheme is able to protect memory errors without the requirement of additional memory cells.

4.2 The Adaptive Coding for Approximate Computing Framework

4.2.1 Memory Error Model

In this section, we assume that the computation units in the unreliable hardware inference system model (Fig. 1.2) are error-free and focus on the errors in unreliable memory units. We use the bit-flipping model for faulty memories. The bit-flipping model is also considered in [55, 67, 68]. Unlike Section 2.3.1, in this Chapter we assume that the memory cells are subject to transient errors. We model the effect of transient errors in each memory cell as passing a binary input through a BSC with cross-over probability σ_{me}. To be more specific, suppose a bit is stored in a memory cell. The value retrieved at read time differs from the original value with probability σ_{me}, as illustrated in Fig. 4.1. Such memory errors may be caused by voltage over-scaling, semiconductor-process variation, electromagnetic interference, etc., [67,69,70][1]. We assume that the BSCs associated with different memory cells are independent but have the same cross-over (bit-flipping) probability σ_{me}. The logic gates performing the computations (computation units) are assumed to be noise-free. The noise-free computation units fetch their input (input data or the previously computed data) from the memory which is subject to hardware errors.

4.2.2 Adaptive Code Design Methodology

Approximate computing in hardware is based on designs of components whose output does not exactly match the prescribed values, due to the impact of either timing-induced errors or functional approximation. We can apply approximate computing techniques in certain applications (e.g., image processing) in which

[1]For example, the SRAM bit error rate is 10^{-3} when the supply voltage is 0.6V [67], and 5.2×10^{-3} under the radiation intensity of 2.8×10^{-4}rad [70].

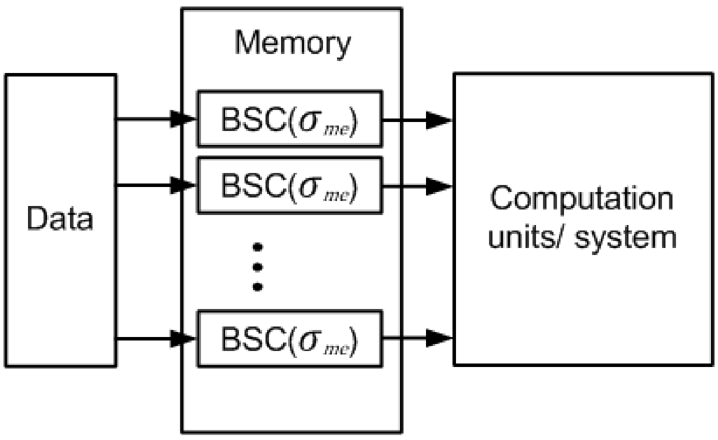


Figure 4.1: The bit-flipping memory error model.

variations in hardware implementation and output quality degradation are tolerable to a certain degree with a careful system design [20, 21]. Inspired by approximate computing techniques and [67], we propose the ACOCO framework. We first reduce the number of bits used to represent each input (to a computation unit) being stored in the memory cells, allowing some distortion in the quantity. Then, a (systematic) channel code is applied to protect the slightly distorted representation against possibly harmful errors (e.g., the errors in the sign bit or MSB) by storing redundant bits in memory cells that would otherwise be used to store less important bits of the uncoded quantity. More specifically, consider each computation unit input (from the memory) as a source message. We first encode the source message using a (lossy) source encoder so as to represent the message with fewer bits. We then append redundant bits using a channel encoder. The system block diagram for ACOCO is shown in Fig. 4.2. The source message, i.e., the original data or the previously computed data to be written into the memory cells, is denoted by x_s. The source encoder generates source codeword x_{se}, the compressed version of x_s, which is fed into the channel encoder. The output x_{ce} of the channel encoder represents the binary representation stored in the memory cells. The output x_{ch} of the hardware error channel represents the binary repre-

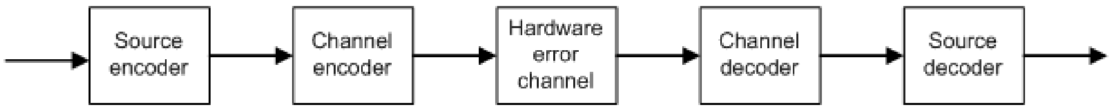


Figure 4.2: A system diagram of the adaptive coding for approximate computing framework.

sentation read from memory cells. A channel decoder followed by a source decoder decodes x_{ch}, and the output x_o is the input to the computation units performing operations on the data. Note that we choose the number of bits reduced during the source coding stage to be equal to the number of redundant bits introduced during the channel coding stage. Hence, jointly, our adaptive code is of rate 1.

Since we adopt BSC as our memory error model, binary linear block codes, which are commonly used in communication systems, are promising candidates for the channel code part of our adaptive codes. For the source coding stage, given the absence of a generally applicable assumption on the probability distribution of computation unit inputs, and given the short lengths of the binary representations of the inputs, lossless source coding schemes such as Hoffman coding or Lempel-Ziv coding, whose efficiency hinges on knowledge of the input distribution and long block-lengths, are not suitable for compressing the inputs. Therefore, we adopt the truncation-based scalar quantization as the source coding scheme. Here we propose a general methodology for minimizing the impact of the distortion introduced in the source encoder on the system output. An ideal source encoder is able to minimize the difference between the suitably quantified difference in the output of the system with noise-free memory and the system subject to memory errors but protected by the proposed adaptive codes. However, for most systems, especially those with iterative data processing, the input-output relationships are too complicated to allow for a tractable analysis to evaluate the effects of the distortion. Therefore, we design a cost function $f(x_s, x_o)$ to (approximately) measure the effects of the distortion introduced in the source encoder on the system output, and then construct the source code based on the cost function. Different cost functions are used for systems with different operations on the data read from faulty memory to capture the severity of the distortion. For example, the quadratic loss function, $f(x_s, x_o) = (x_o - x_s)^2$, is a cost function suitable for a system with addition operations.

We first consider a simple code design under the ACOCO framework. Assume that we use a (n, k) channel code, with k information bits and n coded bits. When we consider the cost function $f(x_s, x_o) = (x_o - x_s)^2$, one candidate source coding scheme is to discard the $n-k$ LSBs in the binary representation of x_s. This source coding scheme is equivalent to quantizing x_s by setting the $n - k$ LSBs to zero and discarding them in its binary representation, thereby reducing the resolution from u (the original resolution) to $u \times 2^{n-k}$. Next, we encode the k MSBs by the (n, k) channel code. The additional $n - k$ bits generated by the channel code are stored in the memory cells originally storing the $n-k$ LSBs. We choose to protect the MSBs because errors occurring in those bits results in a larger distortion than errors in other bits, as evaluated by the cost function $f(x_s, x_o) = (x_o - x_s)^2$. This simple coding scheme is the same as the one proposed in [67]. We refer to this simple coding scheme as the quantization code.

In the following sections, we consider two important types of information processing systems that require more sophisticated adaptive coding schemes: (a) systems dominated by product operations and (b) iterative decoders with min operation and sign-bit decision. We design appropriate cost functions and develop adaptive codes for these systems. Then, we compare the performance of the systems implementing proposed adaptive codes with the systems implementing simple quantization code and the nominal systems (without any codes), and demonstrate the advantage of the theory-guided code design that takes the cost

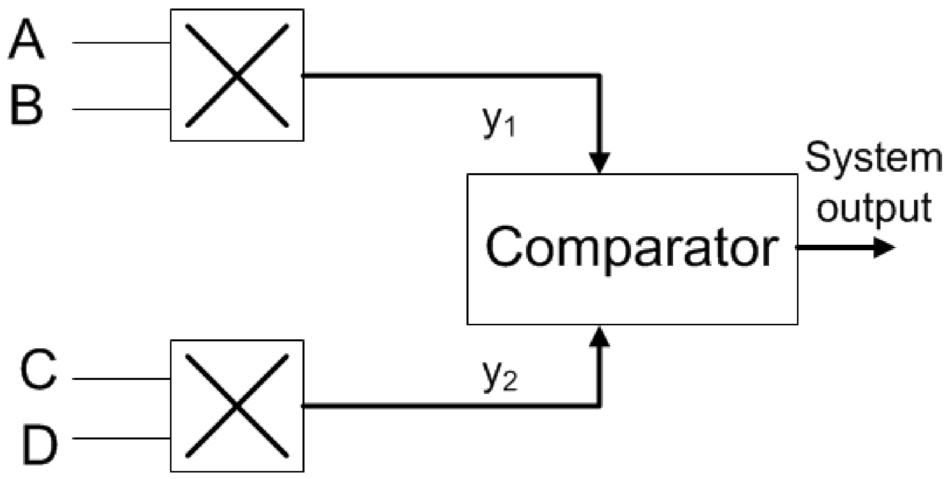


Figure 4.3: Block diagram for the system in Table 4.1 and 4.2.

function into consideration and the error correction capability of our codes.

4.3 Adaptive Codes for Product Operations

We first demonstrate by an exemplary system that for systems dominated by product operations, the normalized quadratic loss function, $f(x_s, x_o) = \frac{(x_o - x_s)^2}{x_s^2}$, is better suited to measure the effects of the introduced distortion induced on the system output than the quadratic loss function, $f(x_s, x_o) = (x_o - x_s)^2$. Then, we design an adaptive coding scheme based on the normalized quadratic loss function. The comparison of the proposed adaptive codes and the quantization code (mentioned in the last section) shows the performance improvement by the adaptive codes designed according to the suitable cost function for the system.

In the exemplary system (Fig. 4.3), we compare the outputs of two multipliers, and the system output is the larger one. Two input sets (and the corresponding system outputs) are shown in Table 4.1 and Table 4.2. As illustrated in Fig. 4.2, x_o is the deteriorated version of x_s. We compare the cost functions and system outputs corresponding to x_s (as input) and x_o (as input) for the two input sets.

	A	B	C	D	y_1	y_2	Output
x_s	10	1	14	2	10	28	y_2
x_o	10	2	14	1	20	14	y_1
$(x_o - x_s)^2$	0	1	0	1			
$\frac{(x_o-x_s)^2}{x_s^2}$	0	1	0	0.25			

Table 4.1: Input set (i) and its corresponding output.

	A	B	C	D	y_1	y_2	Output
x_s	10	7	14	8	70	112	y_2
x_o	10	8	14	7	80	98	y_2
$(x_o - x_s)^2$	0	1	0	1			
$\frac{(x_o-x_s)^2}{x_s^2}$	0	$\frac{1}{49}$	0	$\frac{1}{64}$			

Table 4.2: Input set (ii) and its corresponding output.

A good cost function $f(x_s, x_o)$ should be able to capture the effect of replacing the input x_s with x_o on the system output.

We observe that the effects of distortions applied to the input sets (i) and (ii) are the same when evaluated by the quadratic loss function, but the effects of the distortion on the comparator output is clearly different: the distortion in the input set (i) results in an error in the comparator output (the output is y_1 instead of y_2), while the distortion in the input set (ii) still leaves the comparator output correct. On the other hand, the normalized quadratic loss function evaluates the distortion in the input set (ii) as significantly smaller than the distortion in the input set (i), thus correctly capturing the different severity levels of the distortions in the two cases.

In this section, we design an adaptive coding scheme with much smaller cost function statistics than the quantization code mentioned in the last section, when we use normalized quadratic loss function as the cost function. We consider unsigned numbers to simplify the presentation; the adaptive code design and the analysis can be easily extended to the general case by considering the sign bit as the MSB. The data, when stored in the memory cells, is in the form of (unsigned) binary representation. The binary representation of a number is determined by the quantization resolution and the number of bits B used to represent the number. For example, when $B = 4$ and when the quantization resolution is 1 (i.e., the LSB stands for 1), the binary representation of 12 is $\overline{1100}$. We will use $\overline{a_1 a_2 \dots a_B}$ to denote the binary representation of a number x in the following sections.

4.3.1 Code Design

Since we have x_s in the denominator of the normalized quadratic loss function, the quantization code (which non-adaptively discards the LSBs) results in a small value of normalized quadratic loss when the magnitude of x_s is large, but it has a large value of normalized quadratic loss when the magnitude of x_s is small. This observation inspires us to propose the following adaptive coding scheme. When the magnitude of x_s is large, just as with the quantization code, we discard the LSBs and protect the MSBs by the channel code. The additional bits introduced by the channel code are stored in the memory cells originally storing the LSBs. When the magnitude of x_s is small, however, we keep the LSBs and use a single bit to indicate that the magnitude of x_s is small, identified by the fact that a given number of MSBs are all zero. This bit is stored in the memory cell originally storing the first MSB (which is 0), and the remaining cells storing the zero MSBs are used to store the additional bits generated by the channel code. Assume that we use a (n, k) systematic linear code as the channel code. Following the above design guideline, we propose two adaptive codes with different channel code parameters n and k in the following.

Adaptive code I: When $2k - n - 2 \geq 0$, we use the following source code. Denote the binary representation of x_s, the input to the source encoder, as $\overline{b_1 b_2 \ldots b_B}$ (recall that B is the length of the binary representation), and the binary representation of x_{se}, the output of source code encoder, as $\overline{c_1 c_2 \ldots c_{B-n+k}}$. When at least one of the $k-1$ MSBs is 1, we say that the x_s is "large", otherwise x_s is "small". We then use the MSB in x_{se}, c_1, to indicate whether this x_s is large (set c_1 to 1 when x_s is large), and the remaining $B-n+k-1$ bits in x_{se} are equal to the $B-n+k-1$ MSBs in the binary representation of x_s. The $n-k+1$ LSBs in the binary representation of x_s are discarded. When all the $k-1$ MSBs are zero (i.e., x_s is small), we set the MSB in x_{se}, c_1, to zero, set the $B-k+1$ bits in the left of the MSB in x_{se} to be the $B-k+1$ LSBs in binary representation of x_s, and the remaining bits in x_{se} are set to be 0. Note that we do not discard any of the $n-k+1$ LSBs when x_s is small. We summarize this source code as follows (depicted in Fig. 4.4).

Case 1: If $\sum_{i=1}^{k-1} b_i > 0$, then $c_1 = 1$ and $c_{i+1} = b_i$ for $1 \le i \le B-n+k-1$.

Case 2: If $\sum_{i=1}^{k-1} b_i = 0$, then $c_1 = 0$, $c_{i+1} = b_{k-1+i}$ for $1 \le i \le B-k+1$, and $c_i = 0$ for $i \ge B-k+2$.

The (n,k) channel code protects the first k bits (denoted by the dashed box in Fig. 4.4) in x_{se}.

Adaptive code II: When $2k-n-2 < 0$, the number of available bits in x_{se} may not be enough to represent all the non-zero bits in the binary representation of x_s (see Fig. 4.4 case 2). Hence we design the following new source code for this case. Here we follow the notation in the previous paragraph describing Adaptive code I. When at least one of the $k-1$ MSBs is 1, we use the same code as Adaptive code I. When all the $k-1$ MSBs are zeros, we have the following two cases. When all b_i, $1 \le i \le B-k+2$, are zero, we set $c_1 = 0$ and $c_2 = 0$. The third to the kth bits in x_{se} are equal to the $k-2$ LSBs in the binary representation of x_s. The remaining $B-n$ bits in x_{se} are set to be 0. When at least one of b_i, $k \le i \le B-k+2$, is

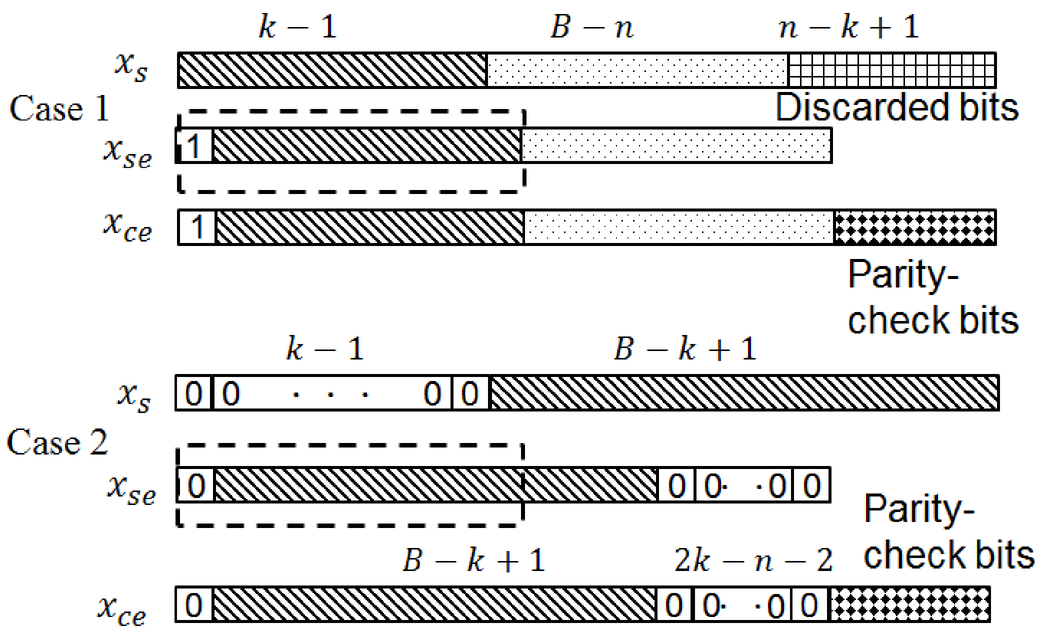


Figure 4.4: Adaptive code I. Channel code protects the part of x_{se} in the dashed box.

non-zero, we set $c_1 = 0$, $c_2 = 1$, and $c_{i+2} = b_{k-1+i}$ for $1 \leq i \leq B-n+k-2$. The $n-2k+3$ LSBs in x_s are discarded. We summarize this source code as follows (depicted in Fig. 4.5).

Case 1: If $\sum_{i=1}^{k-1} b_i > 0$, then $c_1 = 1$ and $c_{i+1} = b_i$ for $1 \leq i \leq B-n+k-1$.

Case 2: If $\sum_{i=1}^{B-k+2} b_i = 0$, then $c_1 = 0$, $c_2 = 0$, and $c_{i+1} = b_{B-k+1+i}$ for $2 \leq i \leq k-1$, $c_i = b_i$ for $k \leq i \leq B-n+k-1$.

Case 3: If $\sum_{i=1}^{k-1} b_i = 0$ and $\sum_{i=k}^{B-k+2} b_i > 0$, then $c_1 = 0$, $c_2 = 1$, and $c_{i+2} = b_{k-1+i}$ for $1 \leq i \leq B-n+k-2$.

The (n,k) channel code protects the first k bits (denoted by the dashed box in Fig. 4.5) in x_{se}.

The source decoder recovers the binary representation of x_s by reversing the above encoding rules. The source decoder further replaces the bits discarded

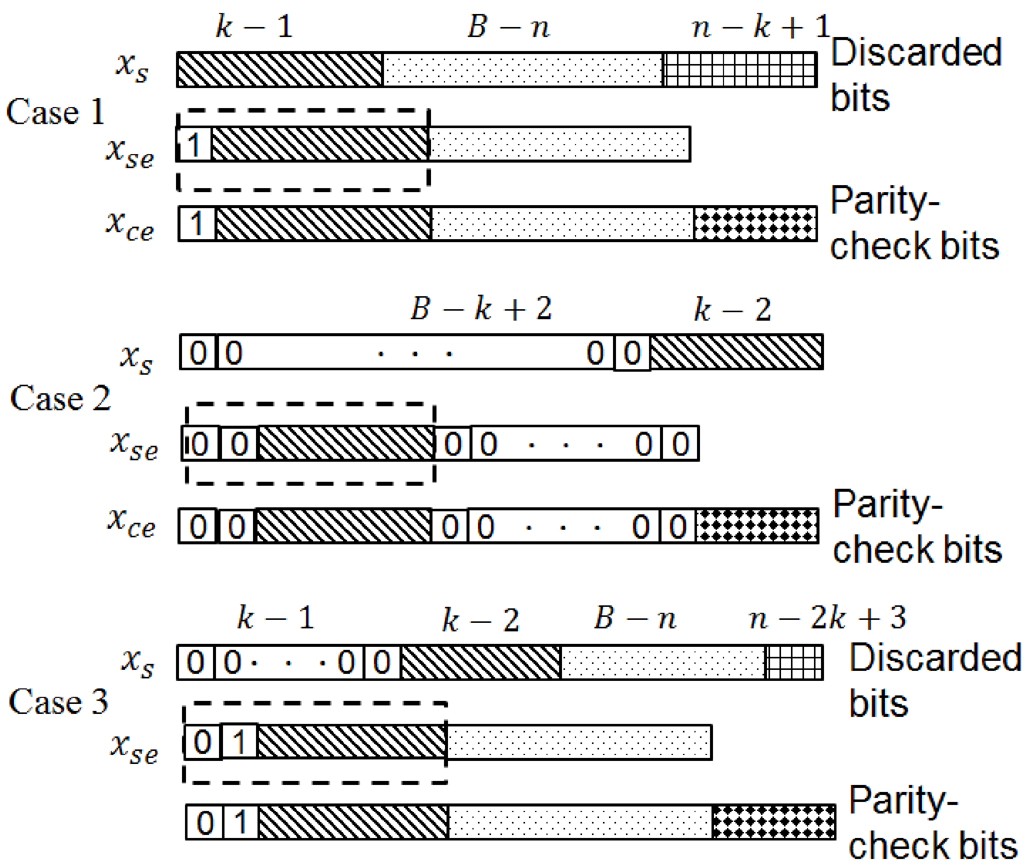


Figure 4.5: Adaptive code II. Channel code protects the part of x_{se} in the dashed box.

during encoding with values randomly selected from 0 and 1.

4.3.2 Analysis of Cost Function Statistics and Numerical Examples

In this section, we analyze (the linear approximation of) the statistics of the cost functions and compare the cost function statistics of the quantization code with the proposed adaptive codes via numerical examples.

There are two random mappings in the ACOCO system (Fig. 4.2): the hardware error channel and the (random) source decoder. The remaining of the mappings are deterministic. Let $x_s, x_o \in X$, and $|X| = 2^B$ since we use B bits to represent x_s (and x_o). We use $|\cdot|$ to denote the set size. We then have the expression for the (general) statistics of the cost function $\mathbb{S}[f(x_s, x_o)]$, as follows,

$$\begin{aligned}&\mathbb{S}[f(x_s, x_o)]\\ &= \sum_{x_s \in X} \sum_{u} \sum_{x_o \in X} g\big(f(x_s, x_o)\big) P(x_o|x_{cd}) P(x_{cho}|x_{ce}) P(x_s),\end{aligned} \tag{4.1}$$

where $u = x_{ce} \oplus x_{cho}$, $\oplus$ denotes the XOR operation and g is determined by the statistics we want to derive. We have (4.1) because x_{cd} is a deterministic function of x_{cho}, and x_{ce} is also a deterministic function of x_s. To simplify the analysis, we assume that x_s is uniformly distributed on X, i.e., we randomly draw an element in X as our source x_s. We then have $P(x_s) = \frac{1}{|X|}$ for all $x_s \in X$. We further assume that we choose the channel code capable of correcting one bit-flipping error (e.g., $(7,4)$ Hamming code).

Since we have different encoding rules for different elements in X, we divide X into M subsets, Φ_i's, $1 \le i \le M$, for grouping x_s's corresponding to different cases in the previous section. Note that $M = 2$ for Adaptive code I and $M = 3$ for Adaptive code II. Let $x_{cd} \in X_{cd}$, and let Ψ_j's, $1 \le j \le |X_{cd}|$, be subsets of X. We map every x_{cd} to an index, denoted by $I(x_{cd})$. We define $\Psi_{I(x_{cd})}$ to be the set of elements (in X) that x_{cd} maps to (by the source decoder). Since the source decoder is a one-to-many random mapping and every $x_o \in X$ corresponds to exactly one x_{cd}, we have $\cup_{j=1}^{|X_{cd}|}\Psi_j = X$ and $\Psi_{j_1} \cap \Psi_{j_2} = \emptyset$ for $1 \leq j_1, j_2 \leq |X_{cd}|$. Since the source decoder randomly decides 0 or 1 for the discarded bits, we have $P(x_o|x_{cd}) = \frac{1}{|\Psi_{I(x_{cd})}|}$ when $x_o \in \Psi_{I(x_{cd})}$. We say that the binary string u of length B belongs to the set U_1 if among the first k entries in u, there is exactly one entry with the value 1, and the remaining entries in u are 0. Likewise, we say that the binary string u of length B belongs to the set U_2 if among the last $B-k$ entries in u, there is exactly one entry with the value 1, and the remaining entries in u are 0. Denote the all-zero string of length B by u_0.

Based on the sets defined above, we derive the probability mass functions in (4.1) for Adaptive code I and II as follows. To simplify the analysis, we assume that the bit-flipping probability σ_{me} in the hardware error channel is small enough to ignore the quadratic and higher order terms of σ_{me} in our analysis. Since the sets for x_s's and x_o's are different for Adaptive codes I and II, we use $\Phi_{1,i}$ $(i = 1, 2)$ and $\Psi_{1,I(x_{cd})}$ for Adaptive code I, and $\Phi_{2,i}$ $(i = 1, 2, 3)$ and $\Psi_{2,I(x_{cd})}$ for Adaptive code II.

For Adaptive code I, we have two sets for x_s, $\Phi_{1,1}$ and $\Phi_{1,2}$, for Case 1 and Case 2 given in the previous section, respectively. Since the elements in $\Phi_{1,1}$ have at least one 1 on the first $k-1$ bits in their binary representations, $\Phi_{1,1}$ has $2^B - 2^{B-k+1}$ elements, and $\Phi_{1,2}$ has 2^{B-k+1} elements. For $x_s \in \Phi_{1,1}$, the corresponding x_{cd} has $|\Psi_{1,I(x_{cd})}| = 2^{n-k+1}$, and, for $x_s \in \Phi_{1,1}$,

$$P(x_o|x_{cd}) = \begin{cases} \frac{1}{2^{n-k+1}} & \text{if } x_o \in \Psi_{1,I(x_{cd})}, \\ 0 & \text{otherwise.} \end{cases} \tag{4.2}$$

On the other hand, when $x_s \in \Phi_{1,2}$, the corresponding x_{cd} has $|\Psi_{1,I(x_{cd})}| = 1$, and

$$P(x_o|x_{cd}) = \begin{cases} 1 & \text{if } x_o \in \Psi_{1,I(x_{cd})}, \\ 0 & \text{otherwise.} \end{cases} \tag{4.3}$$

Recall that the BSCs are independent across different bits in x_{ce}. We then have

$$P(x_{cho}|x_{ce}) \approx_1 \begin{cases} 1 - B\sigma_{me} & \text{if } x_{cho} \oplus x_{ce} = u_0, \\ \sigma_{me} & \text{if } x_{cho} \oplus x_{ce} \in U_1 \cup U_2, \\ 0 & \text{otherwise.} \end{cases} \tag{4.4}$$

We use $\approx_1$ to denote the linear approximation[2] (discarding the second and higher order terms) in (4.4). Note that our channel code is capable of correcting one bit error, hence $x_{se} = x_{cd}$ when $x_{cho} \oplus x_{ce} \in U_1$. By plugging (4.2), (4.3), and (4.4) into (4.1), we then have the statistics of any given cost function $f(x_s, x_o)$ under Adaptive code I. Following similar procedures, we can derive the statistics of any given cost function $f(x_s, x_o)$ under Adaptive code II.

We compute and compare the statistics of cost functions with the proposed adaptive codes (AD), the quantization code mentioned in Section 4.2.2 (QC), and without any error-correcting code (the nominal case, refer to as NC). We consider two cost functions: $f_1(x_s, x_o) = (x_o - x_s)^2$, which is designed for addition operations, and $f_2(x_s, x_o) = \frac{(x_o - x_s)^2}{x_s^2}$, which is designed for product operations. Recall that depending on whether $2k - n - 2 \geq 0$ or $2k - n - 2 < 0$ holds(see Section 4.3.1), we choose either Adaptive code I or Adaptive code II. Therefore, for $(n, k) = (15, 11)$, we choose Adaptive code I scheme, and we use $(15, 11)$ Hamming code; for $(n, k) = (7, 4)$, we choose Adaptive code II scheme, and we use $(7, 4)$ Hamming code. For both cases, the quantization resolution is 1 (i.e., the LSB stands for 1), and the bit-flipping probability is $\sigma_{me} = 10^{-3}$. We compare the mean, variance and maximum of the cost functions in Table 4.3 and 4.4. In applications where approximate computing is used, a small output deviation is usually acceptable while a large one is not. Therefore, the variance and the maximum of the cost function are more interesting statistics than the mean.

[2]For $x_{cho} \oplus x_{ce} = u_0$ case, $P(x_{cho}|x_{ce}) = 1 - \sum_{i=1}^{B} \binom{B}{i} \sigma_{me}^i (1 - \sigma_{me})^{B-i}$. We get (4.4) by discarding quadratic and higher order terms of σ_{me}.

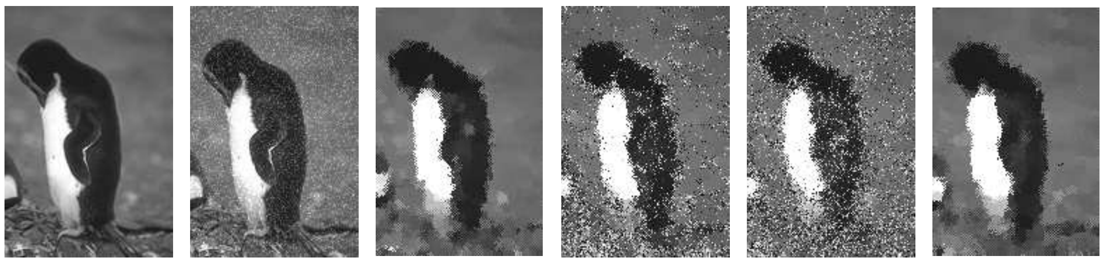
	NC		QC		AD	
	f_1	f_2	f_1	f_2	f_1	f_2
Var	7.58×10^{13}	17.72	4320	0.0028	6101	1.38×10^{-7}
Max	2.68×10^{8}	1.64×10^{5}	225	14	961	0.0037

Table 4.3: Adaptive code I, $(n, k) = (15, 11)$. The smallest variance and max among NC, QC, and AD are marked in red.

	NC		QC		AD	
	f_1	f_2	f_1	f_2	f_1	f_2
Var	1.76×10^{4}	0.0694	257.9	0.1302	3787	0.0238
Max	4096	64	49	6	225	0.9375

Table 4.4: Adaptive code II, $(n, k) = (7, 4)$. The smallest variance and max among NC, QC, and AD are marked in red.

For f_1, NC has large variances and maximums. On the other hand, QC has much smaller variances and much smaller maximums. AD is worse than QC when f_1 is considered. For f_2, NC still has large variances and large maximums. However, AD is better than QC, especially when we compare the variances and the maximums of the cost function. From the numerical examples, we conclude that the proposed adaptive coding scheme (AD) works better when the targeted cost function (f_2) is considered. As discussed previously (c.f. Fig. 4.3 and Table 4.1 and 4.2), for the type of the systems we consider in this section, f_2 is a better choice of the cost function (for these systems) than f_1. Therefore, we expect that the proposed adaptive codes perform better than the quantization code. In the following sections, we demonstrate the improvement brought about by the proposed adaptive coding scheme in the two application examples: image denoising by max-product algorithm and naïve Bayes classification.



(a) (b) (c) (d) (e) (f)

Figure 4.6: Image denoising via MP: (a) original image, (b) contaminated image, (c) recovered image by noise-free MP, (d) recovered image by noisy MP without ECC, (e) recovered image by noisy MP with QC, (f) recovered image by noisy MP with AD.

4.3.3 Application Example I: Max-Production Algorithm for Image Denoising

In this section, we apply the proposed adaptive coding scheme to image denoising by the max-production (MP) algorithm implemented on noisy hardware (with faulty memories), and compare to the cases with quantized coding and with no error-correcting code. An averaging method was proposed to mitigate the effects of hardware noise on the belief propagation algorithm [71]. Instead of modifying the algorithm, we use adaptive codes developed from the ACOCO framework to combat the memory errors. We quickly summarize the MP algorithm as follows.

The MP algorithm can be used to compute the approximate or exact maximum *a posteriori* (MAP) solution to inference problems on factor graphs associated with the random variables of interest, $Z_1, Z_2, \ldots, Z_N$. The random variables take values in a discrete space $\mathcal{Z} = \{1, 2, \ldots, d\}$. A factor graph is a bipartite graph with vertex set $\mathcal{V} \cup \mathcal{F}$, where $\mathcal{F}$ is the set of factors and $\mathcal{V}$ is the set of variable nodes. We denote a factor by its associated set I, where I is the set of indices of the variable nodes connected to this factor, and we denote a variable node by its index $i, i \in \{1, 2, \ldots, N\}$. Each variable node $i \in \mathcal{V}$ is connected with the factors

$I \in \mathcal{F}$ iff $i \in I$, and the variable node i is associated with the random variable Z_i. We denote the neighbors of variable node i by $\mathcal{N}_i$, thus we have $\mathcal{N}_i = \{I : i \in I\}$.

The computation of the MAP solution is conducted by iterative message passing on the factor graph: each node updates its messages according to the messages received from its neighbors, and sends its updated messages back to its neighbors. MP message update rule is

$$m_{I \to i'}^{(\ell)}(z_{i'}) \propto \max_{z_{I \setminus i'} \in \mathcal{Z}^{|I|-1}} \psi_I(z_I) \prod_{i \in I \setminus i'} \prod_{J \in \mathcal{N}_i \setminus I} m_{J \to i}^{(\ell-1)}(z_i), \tag{4.5}$$

where $\psi_I(z_I)$ is the *potential function* associated with the factor I, and ℓ denotes the current iteration. When we terminate the iterative message passing process at iteration L, the *beliefs* of the random variable Z_i is computed as $b_i^{(L)}(z_i) = C_i \prod_{I \in \mathcal{N}_i} m_{I \to i}^{(L)}(z_i)$, where C_i is a normalizing constant chosen such that $\sum_{x_i \in \mathcal{Z}} b_i(z_i) = 1$. The random variable with maximum belief is selected as the MAP solution to the inference problem.

MP can be applied to image denoising, where we seek to recover the original image from the noise-contaminated image. We briefly explain a simple MP-based image denoising algorithm as follows. Consider a 2-D image. We assign a unique number to every color on the image. The value of each pixel on the image is then the number corresponding to the pixel's color. In the 2-D image, the observation of each color (pixel value) is contaminated by additive noise. Assume that the noises added to different pixels are independent. To recover the image from the pixel value observations contaminated by noise, we construct a 2-D grid factor graph with an appropriate choice of potential functions (e.g., Potts model). We associate each pixel in the image to a variable node on the factor graph. The probabilities of all the possible colors of the associated pixel are presented by the message vector. To be more specific, consider the image with a 256-level grayscale; a message associated with a variable node is a vector with 256 entries. Each entry is the probability that the pixel associated with the variable node is at a darkness level corresponding to the entry. We run the MP algorithm on the factor graph to obtain the most likely value of every pixel based on the contaminated observations.

In this example, we associate the 2-D grid factor graph for the "penguin" image (with size 179×122 and 256 gray-scale levels) to a Potts model [61] with coefficient $\xi = 0.05$. In a Potts model, the potential function ψ_I of a factor connecting two variable nodes is 1 for the two variables with the same value, and ξ for the two variables with different values. We assign the integers 0 to 255 to the 256 grayscale levels. Every observation of the pixel value in the image is contaminated with an independent Gaussian random variable with zero-mean and standard deviation 30. The original image and the contaminated image are shown in Fig. 4.6a and Fig. 4.6b. The MP messages are stored in the faulty memories with bit-flipping rate 2.5×10^{-3}, as described in Section 4.2. We consider the following cases: memory without error-correcting code (NC), with quantization code (QC), and with the proposed adaptive code (AD). We use $B = 9$ bits for each message, and therefore we choose $(7, 4)$ Hamming code as our channel code. We run MP for 10 iterations to recover the original image from the contaminated one. Images recovered by noise-free MP, NC, QC and AD are shown in Fig.s 4.6c, 4.6d, 4.6e, and 4.6f, respectively.

Comparing Fig.s 4.6e, 4.6d, and 4.6f, we observe that for QC and NC, a large portion of the pixels are determined incorrectly and the recovered image is very different from the image recovered using noise-free MP (Fig. 4.6c). On the other hand, for AD, the recovered image closely resembles the image recovered by noise-free MP. This observation agrees with the theoretical analysis: without applying an error-correcting code, the memory errors degrade the algorithm performance. Although the error-correcting code can correct memory errors, the effect of distortion introduced in QC is large in product operations and the quality of the recovered image is still significantly deteriorated. However, by using our adaptive code, we correct the harmful memory errors as well as keep the effect of the distortion introduced by the source encoder small. The recovered image is therefore almost indistinguishable relative to the one recovered by noise-free MP (Fig. 4.6c and 4.6f).

4.3.4 Application Example II: Naïve Bayes Classification

In this section, we consider the naïve Bayes classification system, which is widely used in statistical learning systems, apply the proposed adaptive code to this system, and evaluate its performance.

A classifier is a function $\mathcal{C}$ that maps input feature vectors $\theta \in \times$ to output class labels $\phi \in \{1, 2, \ldots, M\}$, where $\times$ is the feature space. Bayesian classifiers assign the most likely class label ϕ to a given sample described by its feature vector θ based on $P(\theta|\phi)$. Learning such classifiers can be greatly simplified by assuming that the features are independent given the class label, i.e., $P(\theta|\phi) = \prod_{i=1}^{n} P(\theta_i|\phi)$, where θ_i is the ith entry of ϕ. The resulting classifier known as naïve Bayes has been proven effective in many practical applications, including text classification, medical diagnosis, and systems performance management [72].

Here we consider the Gaussian naïve Bayes classifier (GBC). In GBC, we assume that the distribution $P(\theta|\phi)$ is Gaussian. Learning of GBC is done by computing the mean and variance of each feature for each class from the training data. The likelihood of a sample with the feature vector θ belonging to class label ϕ is

$$P(\theta|\phi) = \prod_{i=1}^{n} P(\theta_i|\phi) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma_{i,\phi}^2}} \exp\left(-\frac{(\theta_i - \mu_{i,\phi})^2}{2\sigma_{i,\phi}^2}\right), \tag{4.6}$$

where $\mu_{i,\phi}$ and $\sigma_{i,\phi}$ are the mean and the variance of the Gaussian distribution for the feature θ_i conditioned on the class label ϕ, respectively.

In this application example, we apply the GBC implemented on noisy hardware with faulty memories to classify the data (sample) generated from (4.6). For every experiment, we generate 150 samples, each sample has four features, and there are three classes in the the 150 samples. We randomly select half of the samples as training samples, and the other half as testing samples. We compare the average classification error rates (over 1000 experiments) of the GBC without error-correcting code (NC), the GBC with the quantization code (QC), the GBC with the proposed adaptive code (AD), and the noise-free GBC. We use $B = 7$ bits for each message, and therefore we choose $(7, 4)$ Hamming code as our channel code. The GBC reads the likelihoods of each feature for each class from the faulty memory cells, decodes and multiplies them to derive the aggregate likelihood. We decide which class the sample belongs to by comparing the aggregate likelihood of each class.

In Fig. 4.7, we observe that the increase in bit-flipping rate σ_{me} has very little effect on the classification error rates under QC and AD, while the classification error rate under NC increases a lot as σ_{me} increases. This is due to the ability of the Hamming code to correct most of the harmful memory errors. We also find that the classification error rate under QC is much larger than under AD due to (non-adaptively) discarding the LSBs. AD is very close to the noise-free case (less than 0.5% difference) in the entire σ_{me} region we considered, due to the better choice of cost function and code design.

Both application examples demonstrate that by applying the proposed adaptive code developed under ACOCO, we successfully correct most of the harmful memory errors as well as keep the effects of the distortion introduced in the source encoder small.

4.4 Adaptive Codes for Min-Sum LDPC Decoder

In this section, we consider another type of systems: iterative decoders with min operations and sign-bit decision. We consider the min-sum decoder, which is widely applied in communication/storage systems, and develop an adaptive

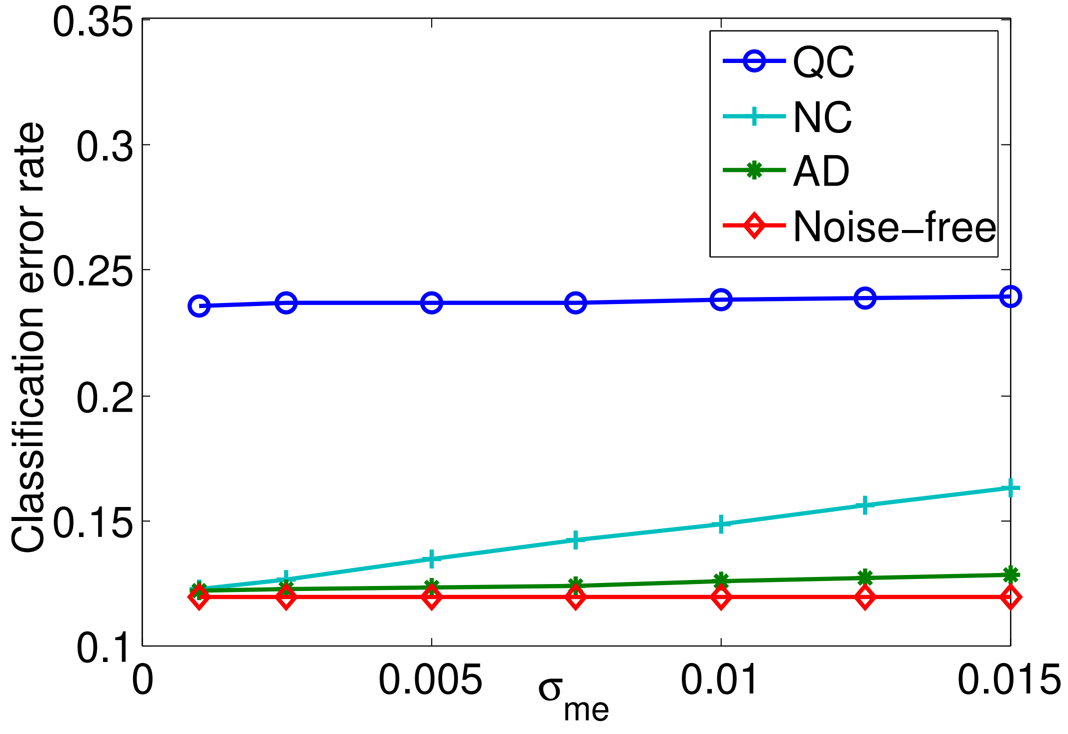


Figure 4.7: Gaussian naïve Bayes classifiers comparison.

coding scheme for the min-sum decoder subject to memory errors.[3]

Recently, noisy decoders subject to memory errors attracted much attention. Important topics including density evolution, equivalent noise modeling, and unequal error protection were studied in [12, 55, 74]. In this section, under the ACOCO framework, we develop adaptive error-correcting codes tailored for the noisy min-sum decoder for LDPC codes subject to memory errors. Unlike the previously proposed error protection methods, the proposed adaptive codes based on ACOCO do not require any additional memory cells while offering the same error protection capability. We start with a brief review of LDPC codes and the noise-free min-sum decoder, and then develop the adaptive codes for the noisy min-sum decoder.

[3]Since some of min-sum decoder's variants have similar message distribution characteristics [73], our adaptive coding scheme can also be applied to these decoders.

4.4.1 Min-Sum Decoder

On the bipartite graph associated with an LDPC code, let $\mathcal{N}_v$ and $\mathcal{N}_c$ denote the nodes connected to the variable node v and to the check node c, respectively. Let y_v be the decoder input at the variable node v, and $x_v \in \{+1, -1\}$ be the transmitted bit corresponding to the variable node v. Denote the noise-free messages sent in the ℓth iteration by $\tilde{m}^{(\ell)}$. More specifically, $\tilde{m}_{v,c}^{(\ell)}$ denotes the message sent from variable node v to its incident check node c, while $\tilde{m}_{c,v}^{(\ell)}$ denotes the message passed from check node c to its incident variable node v. For completeness, let us quickly summarize the main steps of the noise-free min-sum decoder.

- (Initialization) At iteration $\ell = 0$, each variable node v sends the message $\tilde{m}_{v,c}^{(0)} = \ln \frac{P(y_v|x_v=1)}{P(y_v|x_v=-1)}$ to each check node c, $c \in \mathcal{N}_v$.
- (Check node) At each iteration ℓ, $\ell \geq 0$, each check node c sends a message $\tilde{m}_{c,v}^{(\ell)}$ to each variable node v, $v \in \mathcal{N}_c$:

$$\tilde{m}_{c,v}^{(\ell)} = \Big(\prod_{v' \in \mathcal{N}_{c \setminus \{v\}}} \operatorname{sign}(\tilde{m}_{v',c}^{(\ell)}) \min_{v' \in \mathcal{N}_{c \setminus \{v\}}} |\tilde{m}_{v',c}^{(\ell)}| \Big).$$

- (Variable node) At each iteration ℓ, $\ell \geq 1$, each variable node v sends a message $\tilde{m}_{v,c}^{(\ell)}$ to each check node c, $c \in \mathcal{N}_v$:

$$\tilde{m}_{v,c}^{(\ell)} = \tilde{m}_{v,c}^{(0)} + \sum_{c' \in \mathcal{N}_v \setminus \{c\}} \tilde{m}_{c',v}^{(\ell-1)}.$$

Note that for min-sum decoders, a message is the proxy of the log-likelihood ratio (or say belief) of the transmitted bit corresponding to a variable node v.

Here we consider finite precision decoders, broadly deployed in practice. In such decoders, the messages are quantized and stored in the memory cells using a (binary) sign-magnitude representation. We use sign-magnitude representation in this section because the messages in the min-sum decoder are the proxies of likelihoods, which can be either positive or negative. The sign-magnitude representation of each message is determined by the quantization resolution and the number of bits B used to represent the message. For example, when $B = 5$ and the quantization resolution is 1 (the LSB stands for 1), the sign-magnitude representation of 12 is $\overline{01100}$, where the first bit, 0, stands for the positive sign, and the remaining bits represent the magnitude. The value of B is typically between 4 and 7 in practical LDPC decoders [75].

4.4.2 Noisy Min-Sum Decoder and Density Evolution

4.4.2.1 Noisy Decoder Model

We use the bit-flipping model introduced in Section 4.2.1 for the faulty memories storing the messages. This faulty memory model was also considered for the noisy decoders studied in [55] and [76]. As shown in [38] and [76], errors in check nodes do not have much impact on the residual error rates of the LDPC decoders. We therefore assume noise-free check node memory cells to simplify the analysis. For the variable nodes in the noisy decoder at iteration ℓ, the noise-free variable messages are quantized and stored in the faulty memory cells using a sign-magnitude representation. Denote the quantized variable messages before the memory errors are applied by $\hat{m}_{v,c}^{(\ell)}$. With the faulty memory cells described above, the variable messages retrieved from the memory cells, denoted by $m_{v,c}^{(\ell)}$, might be different from $\hat{m}_{v,c}^{(\ell)}$.

4.4.2.2 Density Evolution Analysis

Density evolution of the noisy finite precision min-sum decoder with the above hardware error model was derived in [55] under Gaussian communication channel with noise variance σ. The major difference between the noise-free and the noisy min-sum decoder density evolution analysis is the "distribution perturbation" due to the BSCs modeling memory cell errors. Based on the analysis in [55], we express the "distribution perturbation" as a linear transformation

$$p^{(\ell)} = A(\sigma_{me})\hat{p}^{(\ell)}, \tag{4.7}$$

where $p^{(\ell)}$ is a length-$2^B - 1$ vector representing the probability mass function (PMF) of $m_{v,c}^{(\ell)}$, $\hat{p}^{(\ell)}$ is a length-$2^B - 1$ vector representing the PMF of $\hat{m}_{v,c}^{(\ell)}$, and $A(\sigma_{me})$ is a $2^B - 1 \times 2^B - 1$ matrix. Recall that as defined in Sec. 4.4.1, B is the number of bits in the sign-magnitude representation of each message. Detailed derivation of $\hat{p}^{(\ell)}$ can be found in [55]. The matrix $A(\sigma_{me})$ is derived as follows. Let $\{\alpha_i\}_{i=1}^{2^B-1}$ be the set of possible values of messages $\hat{m}_{v,c}^{(\ell)}$ and $m_{v,c}^{(\ell)}$. Let the ith entry in $p^{(\ell)}$ denote the probability of the event $m_{v,c}^{(\ell)} = \alpha_i$, $1 \le i \le 2^B - 1$. Likewise, let the ith entry in $\hat{p}^{(\ell)}$ denote the probability of the event $\hat{m}_{v,c}^{(\ell)} = \alpha_i$. The (i, j) entry in $A(\sigma_{me})$, $A_{i,j}(\sigma_{me})$, denotes the conditional probability of the event $m_{v,c}^{(\ell)} = \alpha_i$ given $\hat{m}_{v,c}^{(\ell)} = \alpha_j$, is calculated as

$$A_{i,j}(\sigma_{me}) = P(m_{v,c}^{(\ell)} = \alpha_i | \hat{m}_{v,c}^{(\ell)} = \alpha_j) = (1 - \sigma_{me})^{B-d_{i,j}} \sigma_{me}^{d_{i,j}}, \tag{4.8}$$

where $d_{i,j}$ is the Hamming distance between the sign-magnitude representation of α_i and α_j.

The residual error rate $p_r^{(\ell)}$ at iteration ℓ is then derived as

$$p_r^{(\ell)} = \sum_{\alpha_i < 0} p_i^{(\ell)} + \frac{p_\zeta^{(\ell)}}{2}, \tag{4.9}$$

where $p_i^{(\ell)}$ is the ith entry in $p^{(\ell)}$, and ζ is the index of the entry in $p^{(\ell)}$ representing the probability of the message being zero, i.e. $\alpha_\zeta = 0$. When the message is equal to zero, we pick uniformly at random either $+1$ or -1 as the transmitted signal.

4.4.3 Adaptive Codes for the Noisy Min-Sum Decoder

We begin with some key observations from the message update rules and the density evolution analysis for the noisy min-sum decoder. Then, we design an adaptive coding scheme for the variable node memory and develop density evolution analysis for the noisy min-sum decoder using the proposed codes.

4.4.3.1 Code Design

From the density evolution analysis in [55], we observe that, when the communication channel noise and hardware error rate are sufficiently small and when ℓ is sufficiently large, $\hat{p}_i^{(\ell)}$ with $i = \arg\max_k \alpha_k$ (recall that α_i is the value of $m_{v,c}^{(\ell)}$ corresponding to the ith entry in $p^{(\ell)}$) is very close to one, and $p_i^{(\ell)}$ with $i = \arg\min_k \alpha_k$ is very close to σ_{me}. That is, when the communication channel noise and hardware error rate are sufficiently small and when ℓ is sufficiently large, the PMF of $\hat{m}_{v,c}^{(\ell)}$ concentrates at the largest positive number (as with the noise-free min-sum decoder), and most of the residual errors come from the bit flips in the memory cells storing sign bits (converting the largest representable positive value, i.e. the correct value, to the largest representable negative value, i.e. the incorrect value). Bit flips in the other direction, namely, converting the largest representable negative value to the largest representable positive value, are fortuitously possible, and will be taken into consideration when we perform density evolution analysis in Section 4.4.3.2.

Due to the min operation in check nodes, the decoding process is sensitive to the precision level of small-magnitude messages. However, for the large-magnitude messages, the precision level is less important.

Based on the above observations, we specify a cost function and design an adaptive coding scheme to protect memory cells in the noisy min-sum decoder according to the cost function. The cost function for the noisy min-sum decoder is

$$f(x_s, x_o) = \mathbf{1}(|x_s| \geq \tau)(x_o - x_s)^2 + \mathbf{1}(|x_s| < \tau)\mathbf{1}(x_s \neq x_o), \tag{4.10}$$

where $\mathbf{1}(\cdot)$ is the indicator function that is set to 1 if the condition in the parenthesis holds, and 0 otherwise, and τ is the threshold for "large magnitude." We choose τ to be a number whose sign-magnitude representation is in the form of $\overline{010\ldots0}$.

Since when the message magnitude is smaller than τ, the cost function is 1 for every possible x_o except for when $x_o = x_s$, we choose to keep the message as it is when stored in the memory cells (because the cost of every possible error is evaluated to be the same). When the message magnitude is larger than or equal to τ, since the cost function is a quadratic loss function, we quantize the message and apply the $(3,1)$ repetition code (since we have $B = 5$) to protect the sign bit. More specifically, when the MSB is 1, the source encoder discards the two LSBs, and the channel encoder uses the $(3,1)$ repetition code to protect the sign bit. When the MSB is 0, both source and channel encoder become an identity function. When we retrieve the bits from the memory cells, if the MSB reads as 1, the channel decoder decodes the $(3,1)$ repetition code to determine the sign bit. The source decoder randomly selects 0 or 1 for the two discarded LSBs; if the MSB reads as 0, the message is the same as what we retrieve from the memory cells.

4.4.3.2 Density Evolution Analysis of the Noisy Min-Sum Decoder With the Proposed Adaptive Code

The major difference between the density evolution analysis of the nominal min-sum decoder and that of the noisy min-sum decoder with the proposed adaptive code is in the linear transformation (4.7). Given the coding scheme introduced in Sec. 4.4.3.1, the linear transformation becomes

$$p^{(\ell)} = HA(\sigma_{me})G\hat{p}^{(\ell)}, \tag{4.11}$$

where both G and H are $2^B - 1 \times 2^B - 1$ matrices. The matrix G corresponds to the encoding process (including the source and channel encoder, we combine the two encoders in the following discussion to simplify the presentation), and the matrix H corresponds to the decoding process (including the source and channel decoder, we combine the two decoders in the following discussion to simplify the presentation).

We construct G as follows. Like the analysis in Section 4.3.2, we group messages into several classes to analyze the mappings in ACOCO (Fig. 4.2). The set of 2^{B-1} B-bit representations whose MSB is 1 is divided into 2^{B-3} equivalence classes Φ_0, Φ_1, $\ldots$ $\Phi_{2^{B-3}-1}$. Each equivalence class Φ_j has four elements, $\beta_{j,q}$, $q \in \{0,1,2,3\}$. The element $\beta_{j,q}$ has the sign-magnitude representation $\overline{s_j 1 a_{j,1} a_{j,2} \ldots a_{j,B-4} z_1 z_2}$, where the s_j and $a_{j,k}$'s ($k = 1, 2, \ldots, B-4$) are either 0 or 1, and $\overline{0 z_1 z_2}$ is the sign-magnitude representation of length 3 of the index q. Let $\phi(j)$ map the index j of the equivalence class Φ_j to the index of the message value $\alpha_{\phi(j)}$ which belongs to Φ_j and has the sign-magnitude representation $\overline{s_j 1 a_{j,1} a_{j,2} \ldots a_{j,B-4} s_j s_j}$. The equivalence class Φ_j is represented by $\alpha_{\phi(j)}$.

In the encoding process, we copy the sign bit to the two LSBs when the MSB is equal to 1. Hence, all elements in Φ_j are mapped to $\alpha_{\phi(j)}$. Note that the probability of $\alpha_{\phi(j)}$ is represented as the $\phi(j)$th entry in $\hat{p}^{(\ell)}$. Let the probabilities of the other three elements in Φ_j be represented as the $v_1(j)$th, $v_2(j)$th, and $v_3(j)$th entries, respectively, in $\hat{p}^{(\ell)}$ ($v_1(j), v_2(j), v_3(j) \in \{1, 2, 3, \ldots, 2^B - 1\}$). In G, for $j = 0, 1, \ldots, 2^{B-3} - 1$, the row $\phi(j)$ has 1's in columns $\phi(j)$, $v_1(j)$, $v_2(j)$, and $v_3(j)$, 0's elsewhere, and the rows $v_1(j)$, $v_2(j)$, and $v_3(j)$ have all-zero entries. In the remainder of the rows of G, all entries are 0 except for the ones on the diagonal. For example, when $B = 5$, the following entries in G are 1: $(1,1)$, $(1,2)$, $(1,3)$, $(1,4)$, $(5,5)$, $(5,6)$, $(5,7)$, $(5,8)$, (γ,γ), $(24,24)$, $(24,25)$, $(24,26)$, $(24,27)$, $(28,28)$, $(28,29)$, $(28,30)$, $(28,31)$, $\gamma = 9, 10, \ldots, 23$. Other entries are 0. Note that $\alpha_1 = -15, \alpha_2 = -14, \ldots, \alpha_{31} = 15$ in this example.

In the decoding process, when the MSB reads as 1, we set the sign bit to the value that occurs at least twice among the two LSBs and the sign bit. The two LSBs are randomly decided. The set of 2^{B-1} B-bit representations whose MSB is 1 is divided into 2^{B-3} equivalence classes Ψ_0, Ψ_1, $\ldots$ $\Psi_{2^{B-3}-1}$. Denote the majority function (of three binary inputs), which takes the majority of its inputs z_1, z_2, z_3 as its output, by $g_{ma}(z_1, z_2, z_3)$. Each equivalence class Ψ_j has four elements, $\theta_{j,q}$, $q \in \{0, 1, 2, 3\}$. The element $\theta_{j,q}$ has the sign-magnitude representation $\overline{u_{j,q,1}1a_{j,1}a_{j,2}\ldots a_{j,B-4}u_{j,q,2}u_{j,q,3}}$, where the $c_{j,k}$'s ($k = 1, 2, \ldots, B-4$) are either 0 or 1, and $u_{j,q,1}, u_{j,q,2}, u_{j,q,3}$ satisfy $g_{ma}(u_{j,q,1}, u_{j,q,2}, u_{j,q,3}) = s_j$, $s_j \in \{0, 1\}$.

Following the construction of G, here we derive H. When the MSB is 1, the decoding process, in which we randomly decide the two LBSs, is equivalent to the following operation: when we have $x_{cho} = z_1$, z_1 in Ψ_j, we randomly pick an element z_2 from Φ_j (including z_1 if $z_1 \in \Phi_j$), and $x_o = z_2$. Let the probabilities of the four elements in Ψ_j be represented as the $w_1(j)$th, $w_2(j)$th, $w_3(j)$th, and $w_4(j)$th entries, respectively, in $\hat{p}^{(\ell)}$ ($w_1(j), w_2(j), w_3(j), w_4(j) \in \{1, 2, 3, \ldots, 2^B - 1\}$). In H, for $j = 0, 1, \ldots, 2^{B-3} - 1$, the row $v_i(j)$ ($i = 1, 2, 3$) has $\frac{1}{4}$ in columns $w_k(j)$ ($k = 1, 2, 3, 4$), and 0's elsewhere. The row $\alpha_{\phi(j)}$ also has $\frac{1}{4}$'s in columns $w_k(j)$ ($k = 1, 2, 3, 4$), and 0's elsewhere. In the remaining of the rows in H, all entries are 0 except for the ones on the diagonal. Following the example we have for G, the following entries in H are $\frac{1}{4}$: $(\eta_1, 1)$, $(\eta_1, 2)$, $(\eta_1, 3)$, $(\eta_1, 31)$, $(\eta_2, 5)$, $(\eta_2, 6)$, $(\eta_2, 7)$, $(\eta_2, 27)$, $(\eta_3, 24)$, $(\eta_3, 25)$, $(\eta_3, 26)$, $(\eta_3, 8)$, $(\eta_4, 28)$, $(\eta_4, 29)$, $(\eta_4, 30)$, $(\eta_4, 4)$, for $\eta_1 \in \{1, 2, 3, 4\}$, $\eta_2 \in \{5, 6, 7, 8\}$, $\eta_3 \in \{24, 25, 26, 27\}$, $\eta_4 \in \{28, 29, 30, 31\}$. The entries (γ, γ), $\gamma = 9, 10, \ldots, 23$, are 1, and the remaining of the entries are 0. Note that $\alpha_1 = -15, \alpha_2 = -14, \ldots, \alpha_{31} = 15$ in this example.

We compute the residual error rate at the 200th iteration, $p_r^{(200)}$, of the noisy min-sum decoders without error-correcting code (the nominal case, we call it NC), the noisy min-sum with the quantization code (QC), the noisy min-sum decoder with the proposed adaptive code (AD) and the noise-free min-sum decoder via density evolution. We consider the LDPC code with $(d_v, d_c) = (3, 6)$, $B = 5$, bit-flipping probability $\sigma_{me} = 5 \times 10^{-4}$, and Gaussian communication channel noise with zero mean and different standard deviation σ.

The numerical results are shown in Fig. 4.8. Note that the residual error rate of the noise-free min-sum decoder under small σ and large enough number of iterations is 0. In Fig. 4.8, we observe that as in the noise-free min-sum decoder, NC, QC, and AD all exhibit a sharp drop in the residual error rates as σ decreases below certain values. A similar phenomenon was also observed in [77]. As in [77], we refer to the σ value corresponding to the sharp drop on the curve as the phase transition point. Before all the phase transition points, we observe that the residual error rates under QC and AD are roughly on the order of the square of the residual error rate under NC. The phase transition points are 0.781, 0.798, and 0.803 for QC, NC, and AD, respectively. The decoding threshold of the noise-free min-sum decoder is 0.811. It is also worth noting that the proposed adaptive coding scheme not only lowers the residual error rate, but also moves the phase transition point closer to the decoding threshold of the noise-free min-sum decoder (comparing to both QC and NC cases). The low phase transition point under QC (even lower than NC) is due to the distortion introduced in the source encoder by dropping the two LSBs regardless of the message magnitude.

The improvement in the residual error rate in the low σ region is attributed to our adaptive coding scheme: in NC, a single memory cell error in the sign bit results in a decoded bit error, while in QC and AD, at least 2 memory cell errors are required to result in a decoded bit error. In fact, this observation holds more generally. Let $i^* = \arg\max_i \alpha_i$, and the i^*th entry in $\hat{p}^{(\ell)}$ for AD (or QC) be $1-\epsilon_{ad}^{(\ell)}$, and NC be $1-\epsilon_{nc}^{(\ell)}$. That is, the probabilities of the noise-free variable node messages having the largest representable positive value at iteration ℓ are $1-\epsilon_{ad}^{(\ell)}$ and $1-\epsilon_{nc}^{(\ell)}$ for AD (or QC) and NC, respectively. Denote the approximation of the residual error rate of AD up to the second order terms of σ_{me} by $\breve{p}_{r,ad}^{(\ell)}$, and denote the approximation of the residual error rate of NC up to the second order terms of σ_{me} by $\breve{p}_{r,nc}^{(\ell)}$. Then, we have the following lemma:

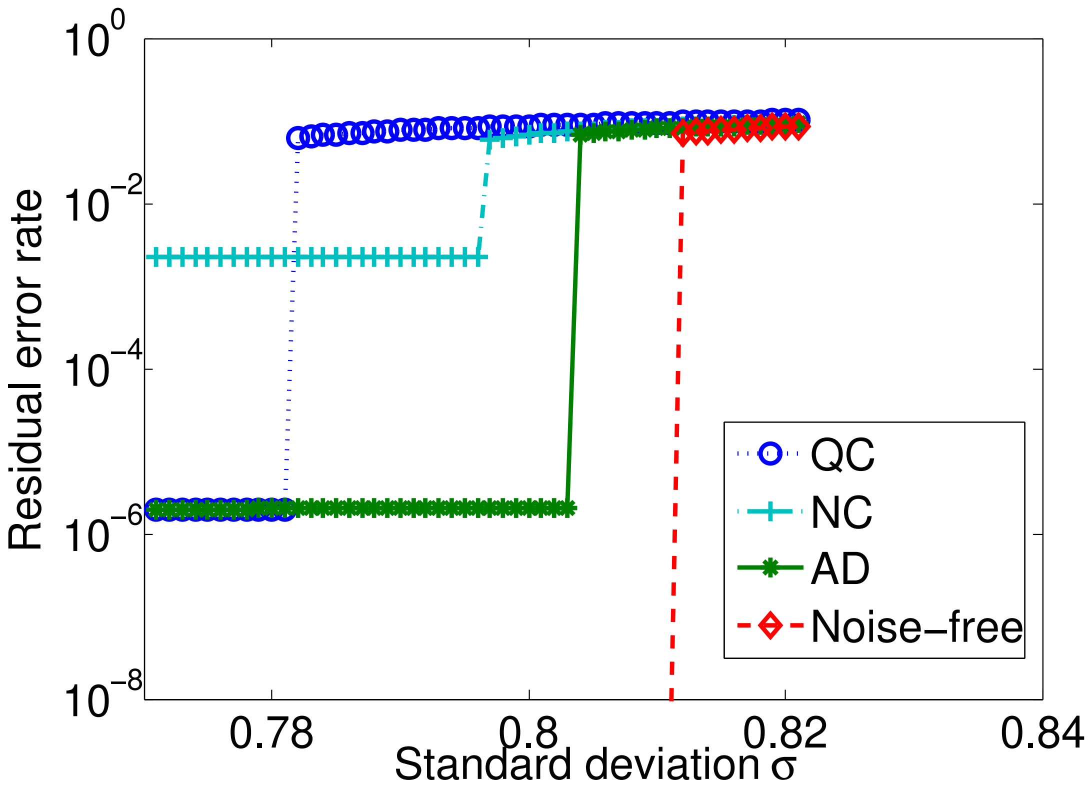


Figure 4.8: Comparison of residual error rates under NC, QC, AD and noise-free decoder for different σ via density evolution.

Lemma 5. *If the communication noise σ and bit-flipping rate σ_{me} are sufficiently small and ℓ is sufficiently large so that $\epsilon_{ad}^{(\ell)}, \epsilon_{nc}^{(\ell)} < \sigma_{me}^3$, then $\breve{p}_{r,ad}^{(\ell)} = (\breve{p}_{r,nc}^{(\ell)})^2$.*

Proof. Since the sign bits are flipped with probability σ_{me}, $\epsilon_{nc}^{(\ell)} < \sigma_{me}^3$, and $\epsilon_{nc}^{(\ell)}\sigma_{me} < \sigma_{me}^4$, we have

$$p_{r,nc}^{(\ell)} \approx_2 (1 - \epsilon_{nc}^{(\ell)})\sigma_{me} \approx_2 \sigma_{me} = \breve{p}_{r,nc}^{(\ell)}, \tag{4.12}$$

where $\approx_2$ denotes the approximation up to the second order terms of σ_{me}. Since with the proposed adaptive coding scheme, at least two bit flips are required to cause an error in the decoded sign bit, $\epsilon_{ad}^{(\ell)} < \sigma_{me}^3$, and $\epsilon_{ad}^{(\ell)}\sigma_{me}^2 < \sigma_{me}^5$, we have

$$p_{r,ad}^{(\ell)} \approx_2 (1 - \epsilon_{ad}^{(\ell)})\sigma_{me}^2 \approx_2 \sigma_{me}^2 = \breve{p}_{r,ad}^{(\ell)}. \tag{4.13}$$

Therefore, $\breve{p}_{r,ad}^{(\ell)} = (\breve{p}_{r,nc}^{(\ell)})^2$. $\square$

We thus conclude that AD achieves significant error rate reduction by adaptively duplicating the sign bit in 2 memory cells storing LSBs without adding any redundant memory cells. In fact, by using a length n code capable of correcting m errors, we are able to reduce $p_{r,ad}^{(\ell)}$ to the order of $(p_{r,nc}^{(\ell)})^{m+1}$ when the communication noise and σ_{me} are sufficiently small.

Next, in Fig.4.9, we compare the trajectory of the residual error rate $p_r^{(\ell)}$ w.r.t. iteration number ℓ under AD and QC when communication noise standard deviation σ is small. Note that in this σ region, the residual error rate under NC is much higher than under AD and QC. In this figure, we plot $p_r^{(\ell)}$ as a function of ℓ (up to 30) for the bit-flipping probability $\sigma_{me} = 5 \times 10^{-4}$ and communication noise standard deviation $\sigma = 0.7$. We observe that the residual error rate under AD is lower than that under QC in every iteration before they reach the lowest achievable value (this lowest value is the same for AD and QC, see also Lemma 5). Moreover, the rate of convergence for this lowest value is higher for AD than QC.

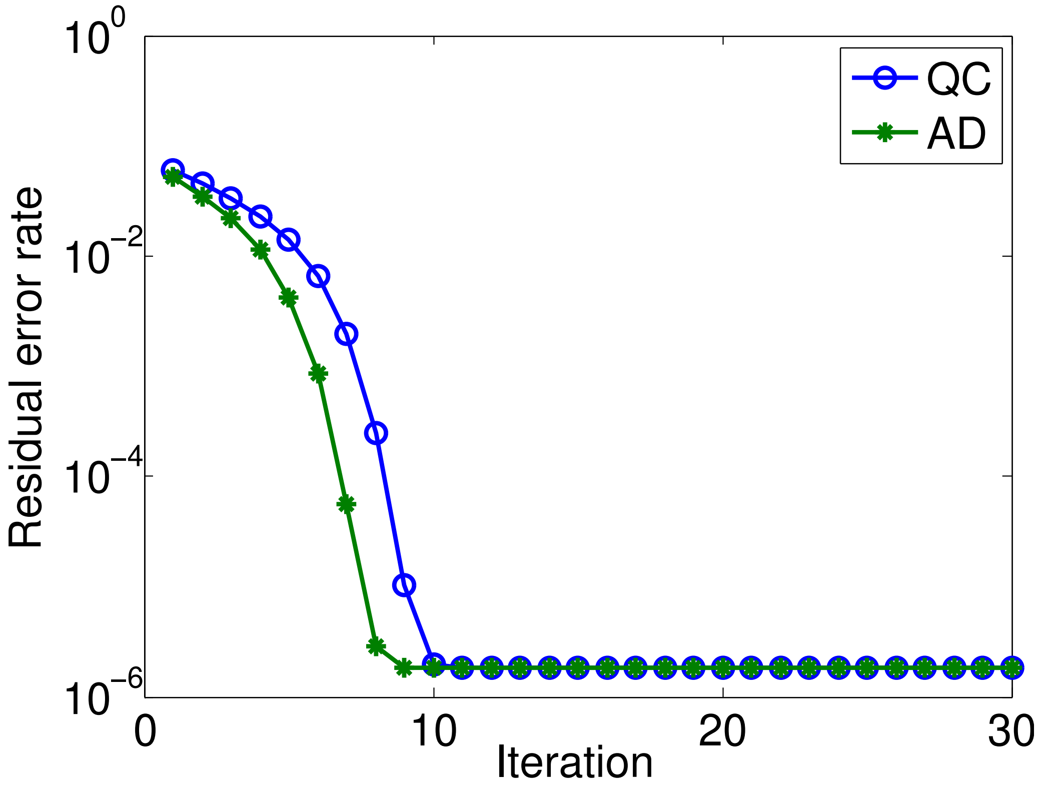


Figure 4.9: Comparison of the residual error rate under QC and AD at the end of each iteration via density evolution.

The effect of the precision level in the min-sum decoder on the residual error rate is independent of the choice of code parameters, hence the above observations are expected to hold more generally.

4.4.4 Simulation Results

We now compare via simulations the residual error rates under NC, QC, and AD for finite length LDPC codes under the faulty memory model. We use a $(3, 6)$-regular LDPC code with code length 2640 [41]. The communication channel noise is modeled as a zero-mean Gaussian random variable with variance σ^2. The bit-flipping probability of noisy memory is σ_{me}. The maximum number of iteration is 30.

We first fix $\sigma_{me} = 5 \times 10^{-4}$ and compare the residual error rate under NC, QC, and AD under different σ's (Fig. 4.10). It is clear that when σ is large (corresponding to the region to the right of the phase transition points shown in Section 4.4.3.2), the residual error rates of all the decoders are approximately the same, while for small σ (corresponding to the region to the left of the phase transition points shown in Section 4.4.3.2), the residual error rates under AD (and QC) are roughly the square of the residual error rate under NC, as predicted in Lemma 5. In the transition region between small and large σ, AD still achieves a lower residual error rate than QC and NC. Interestingly, although QC has a lower residual error rate than NC in the small σ region, it has larger residual error rate than NC in part of the transition region. The larger residual error rates in the transition region reflect the observation from density evolution analysis: QC has a lower phase transition point than NC due to the distortion introduced in the source encoder.

Next, we fix $\sigma = 0.7, \sigma_{me} = 5\times10^{-4}$ and compare the residual error rates under QC and AD in each iteration (up to 30). The residual error curves shown in Fig.

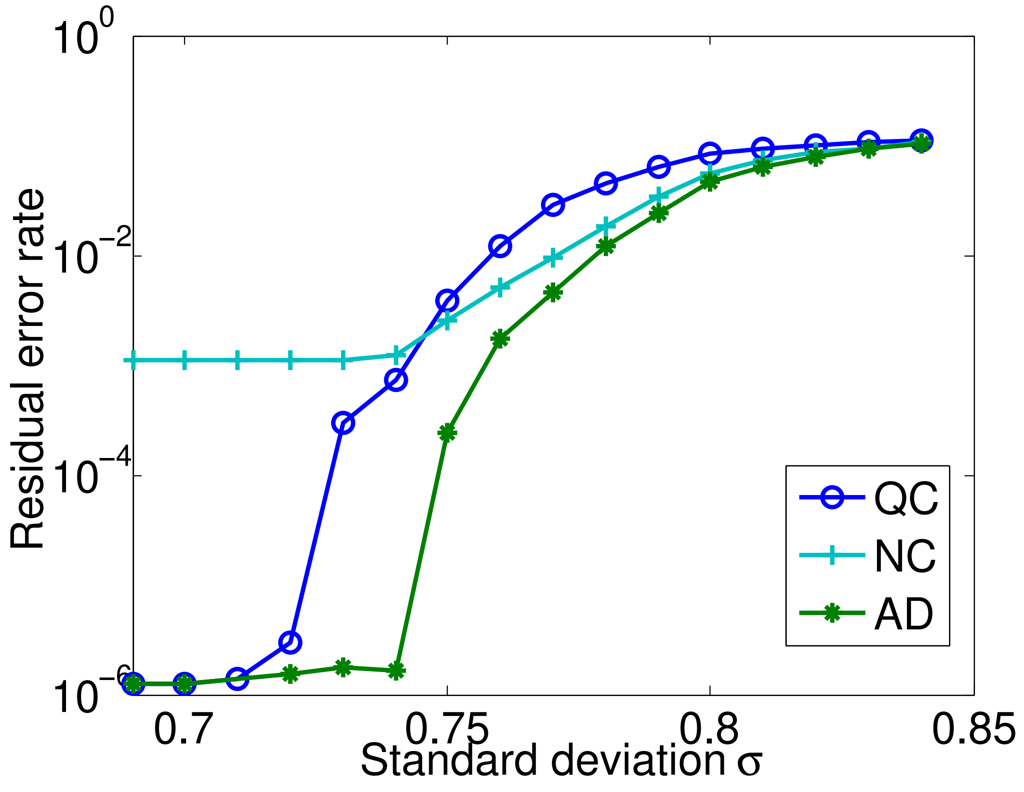


Figure 4.10: Comparison of residual error rates under NC, QC, and AD for different σ in finite length code simulations.

4.11 exhibit the same trend as predicted by the density evolution analysis (Fig. 4.9): the residual error rate under AD converges faster to its lowest achievable error rate than QC.

To sum up, NC has the highest residual error rate in the small σ region; QC achieves a lower residual error rate in the small σ region at the expense of slower convergence and a higher residual error rate in the transition region; on the other hand, AD achieves a lower residual error rate in the small σ region and maintains fast convergence and a low residual error rate in the transition region. The superior performance of AD is attributed to the code design under ACOCO with an appropriately chosen cost function.

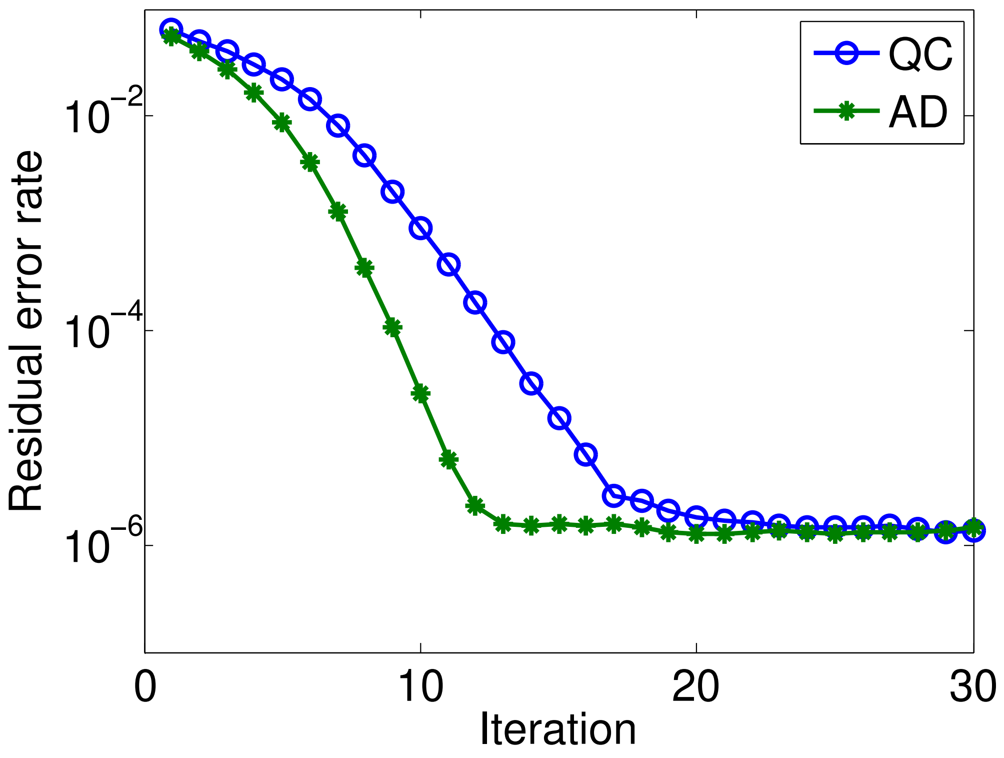


Figure 4.11: Comparison of residual error rates under QC and AD at the end of each iteration in finite length code simulations.

CHAPTER 5

Conclusion

5.1 Summary of the Results

In this dissertation, we investigated the iterative inference systems implemented on unreliable hardware. We started from studying a noisy finite-alphabet iterative decoder implemented on hardware built out of processors with different error rates. We derived an iterative expression for the error rates as a function of both transmission noise and processing noise. As an example, we formulated the optimal processor assignment for a noisy Gallager E decoder, and showed improvement in the residual error rate when processors were optimally assigned. We then extend to study a noisy Gallager B decoder under processor transient errors and memory permanent errors. We derived both the exact and approximate density evolution expressions for the decoder output bit error rate as a function of channel noise (and transient and permanent decoder errors). Additionally, we proposed a scheme for detecting permanent errors and we illustrated the method with an accompanying example.

Next, we studied a more general algorithm, the BP algorithm implemented on noisy hardware. We proposed two low-overhead robust implementations of the BP algorithm targeting computation noise with different characteristics. Under the contraction mapping condition, we mathematically derived the sufficient conditions for computation noise to guarantee the convergence of the two proposed algorithms, namely, censoring BP and averaging BP. We investigated two important applications of BP: image denoising and a BP decoder for LDPC codes for communication systems. In the image denoising application, the image recovered by averaging BP has a much better quality than the image recovered by nominal BP. For the BP decoder, we proposed an averaging BP decoder by applying averaging BP. We demonstrated that the averaging BP decoder achieves lower residual error rates than the nominal BP decoder under various computation noise conditions.

Inspired by approximate computing, we proposed the ACOCO framework to develop adaptive codes for a variety of computations performed on data read from faulty memories. By introducing distortion with negligible effects in the source encoder, the proposed codes protect the data against memory errors without requiring additional memory cells, i.e., the codes have coding rate 1. Under the ACOCO framework, we first developed adaptive codes for systems dominated by product operations. By applying the proposed code to max-sum image denoising and naïve Bayes classification, we demonstrated that our codes successfully correct harmful memory errors, and that the distortion introduced by the source encoder has negligible effects on the system performance. Then, we further developed adaptive codes for min-sum decoders with faulty memories. Via density evolution analysis, we showed that application of the proposed codes indeed lowers the residual error rate. Higher phase transition points are observed when the proposed codes are applied to a noisy min-sum decoder compared with when the proposed codes are not applied and when only quantization codes are applied.

5.2 Future Directions

In this dissertation, we provided a novel information-theoretic approach for concrete fundamental performance limit characterizations and theory-guided system designs with mathematical guarantees. The same methodology can be applied to the study of a wide range of systems and designs that can achieve reliable performance when the system components are unreliable. By establishing mathematical foundations for the analysis of the tradeoff between design overhead/energy consumption and quality of inference, we can successfully explore the new design dimension introduced by considering unreliable hardware to develop systems with better efficiency and performance. We briefly mention some potential directions for research specifically related to this dissertation in the following.

We have provided a detailed density evolution analysis for infinite precision and finite precision LDPC decoders subject to different kinds of hardware errors. A promising research direction is a robust code design against hardware errors based on our density evolution analysis. An ideal robust code against noisy hardware will have better threshold and lower error floor than the code design for a noise-free decoder.

The BP algorithm has been applied in various problems with different message updating rules. For example, approximate message passing for compressive sensing [78] and max-product algorithm for weighted matching [79]. Applying the proposed robust implementation or developing better suited algorithm design for those applications of BP is an important future research direction.

In ACOCO, the most important characteristic of the proposed adaptive codes is its computation-awareness. In other words, we design the adaptive codes according to the computation we are going to perform on the data to be protected by the adaptive code. Therefore, for inference system implemented on unreliable hardware, the error-correcting code design (or more general, algorithm and system design) has to be aware of both hardware error characteristics and the computations/operations performing on the data. A closer look and detailed study of how the two factors interact with each other will lead to better and more efficient design for inference algorithms on unreliable hardware.

References

[1] J. von Neumann, "Probabilistic logic and the synthesis of reliable organisms from unreliable components," *Automata Studies*, pp. 43–98, 1956.

[2] S. Mitra and E. McCluskey, "Word-voter: a new voter design for triple modular redundant systems," in *IEEE VTS*, 2000.

[3] F. Kastensmidt, L. Sterpone, L. Carro, and M. Reorda, "On the optimal design of triple modular redundancy logic for SRAM-based FPGAs," in *DATE*, 2005.

[4] C. Hadjicostis and G. Verghese, "Coding approaches to fault tolerance in linear dynamic systems," *IEEE Trans. Inf. Theory*, vol. 51, no. 1, pp. 210 – 228, Jan. 2005.

[5] K.-H. Huang and J. A. Abraham, "Algorithm-based fault tolerance for matrix operations," *IEEE Trans. Comput.*, vol. 100, no. 6, pp. 518–528, Jun. 1984.

[6] G. Varatkar, S. Narayanan, N. Shanbhag, and D. Jones, "Stochastic networked computation," *IEEE Trans. VLSI Syst.*, vol. 18, no. 10, pp. 1421–1432, 2010.

[7] R. Hegde and N. R. Shanbhag, "Soft digital signal processing," *IEEE Trans. VLSI Syst.*, vol. 9, no. 6, pp. 813–823, Dec. 2001.

[8] J. Han, J. Gao, P. Jonker, Y. Qi, and J. A. Fortes, "Toward hardware-redundant, fault-tolerant logic for nanoelectronics," *IEEE Des. Test. Comput.*, vol. 22, no. 4, pp. 328–339, Jul. 2005.

[9] Y. Emre and C. Chakrabarti, "Memory error compensation techniques for JPEG2000," in *IEEE SIPS*, 2010.

[10] A. Chandrakasan *et al.*, "Technologies for ultradynamic voltage scaling," *Proceedings of the IEEE*, vol. 98, no. 2, pp. 191–214, Feb 2010.

[11] C. Roth, C. Benkeser, C. Studer, G. Karakonstantis, and A. Burg, "Data mapping for unreliable memories," in *ACCCC*, 2012.

[12] M. Khairy, A. Khajeh, A. Eltawil, and F. Kurdahi, "Equi-noise: A statistical model that combines embedded memory failures and channel noise," *IEEE Trans. Circuits Syst. I*, vol. 61, no. 2, pp. 407–419, Feb 2014.

[13] B. Vasic and S. Chilappagari, "An information theoretical framework for analysis and design of nanoscale fault-tolerant memories based on low-density parity-check codes," *IEEE Trans. Circuits Syst. I*, vol. 54, no. 11, pp. 2438 –2446, Nov. 2007.

[14] D. Rossi, N. Timoncini, M. Spica, and C. Metra, “Error correcting code analysis for cache memory high reliability and performance,” in *DATE*, 2011.

[15] M. Jayarani and M. Jagadeeswari, “A novel fault detection and correction technique for memory applications,” in *ICCCI*, 2013.

[16] N. R. Shanbhag, R. A. Abdallah, R. Kumar, and D. L. Jones, “Stochastic computation,” in *IEEE/ACM DAC*, 2010.

[17] J. Han, H. Chen, J. Liang, P. Zhu, Z. Yang, and F. Lombardi, “A stochastic computational approach for accurate and efficient reliability evaluation,” *IEEE Trans. Comput.*, vol. 63, no. 6, Jun. 2014.

[18] S. Narayanan, G. Varatkar, D. Jones, and N. Shanbhag, “Computation as estimation: Estimation-theoretic IC design improves robustness and reduces power consumption,” in *IEEE ICASSP*, 2008.

[19] E. Kim and N. Shanbhag, “Soft N-modular redundancy,” *IEEE Trans. Comput.*, vol. 61, no. 3, pp. 323 –336, Mar. 2012.

[20] R. Venkatesan, A. Agarwal, K. Roy, and A. Raghunathan, “MACACO: Modeling and analysis of circuits for approximate computing,” in *ICCAD*, 2011.

[21] M. Samadi, J. Lee, D. A. Jamshidi, A. Hormati, and S. Mahlke, “Sage: Self-tuning approximation for graphics engines,” in *IEEE/ACM MICRO*, 2013.

[22] A. K. Mishra, R. Barik, and S. Paul, “iACT: A software-hardware framework for understanding the scope of approximate computing,” in *WACAS*, 2014.

[23] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, “Analysis and characterization of inherent application resilience for approximate computing,” in *IEEE/ACM DAC*, 2013.

[24] T. Richardson and R. Urbanke, “The capacity of low-density parity-check codes under message-passing decoding,” *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 599–618, Feb. 2001.

[25] M. G. Taylor, “Reliable information storage in memories designed from unreliable components,” *Bell System Technical Journal*, vol. 47, no. 10, pp. 2299–2337, Dec. 1968.

[26] A. V. Kuznetsov, “Information storage in a memory assembled from unreliable components,” *Problemy Peredachi Informatsii*, vol. 9, no. 3, pp. 100–114, Jul.-Sep. 1973.

[27] S. K. Chilappagari and B. Vasic, “Fault tolerant memories based on expander graphs,” in *IEEE ITA*, 2007.

[28] ——, “Reliable memories built from ureliable components based on expander graphs,” 2007. [Online]. Available: http://arxiv.org/abs/0705.0044

[29] M. Sipser and D. Spielman, “Expander codes,” *IEEE Trans. Inf. Theory*, vol. 42, no. 6, pp. 1710 –1722, Nov. 1996.

[30] S. Chilappagari, M. Ivkovic, and B. Vasic, “Analysis of one step majority logic decoders constructed from faulty gates,” in *IEEE ISIT*, 2006.

[31] R. Radhakrishnan, S. Sankaranarayanan, and B. Vasic, “Analytical performance of one-step majority logic decoding of regular LDPC codes,” in *IEEE ISIT*, 2007.

[32] O. W. Yeung and K. M. Chugg, “On the error tolerance of iterative decoder circuitry,” in *ITA*, 2008.

[33] C. Winstead, Y. Tang, E. Boutillon, C. Jego, and M. Jezequel, “A space-time redundancy technique for embedded stochastic error correction,” in *IEEE ISTC*, 2012.

[34] Y. Tang, C. Winstead, E. Boutillon, C. Jego, and M. Jezequel, “An LDPC decoding method for fault-tolerant digital logic,” in *IEEE ISCS*, 2012.

[35] F. Leduc-Primeau and W. J. Gross, “Faulty Gallager-B decoding with optimal message repetition,” in *ACCCC*, 2012.

[36] L. R. Varshney, “Performance of LDPC codes under faulty iterative decoding,” *IEEE Trans. Inf. Theory*, vol. 57, no. 7, pp. 4427–4444, Jul. 2011.

[37] A. Tarighati, H. Farhadi, and F. Lahouti, “Performance analysis of noisy message-passing decoding of low-density parity-check codes,” in *ISTC*, 2010.

[38] S. M. S. Tabatabaei Yazdi, H. Cho, and L. Dolecek, “Gallager B decoder on noisy hardware,” *IEEE Trans. Commun.*, vol. 61, no. 5, pp. 1660–1673, May 2013.

[39] T. J. Richardson and R. L. Urbanke, “The capacity of low-density parity-check codes under message-passing decoding,” *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 599 –618, Feb. 2001.

[40] S. M. S. Tabatabaei Yazdi, C.-H. Huang, and L. Dolecek, “Optimal design of a Gallager B noisy decoder for irregular LDPC codes,” *IEEE Commun. Lett.*, vol. 16, no. 12, pp. 2052–2055, Dec. 2012.

[41] D. MacKay, “Encyclopedia of sparse graph codes.” [Online]. Available: http://www.inference.phy.cam.ac.uk/mackay/codes/data.html

[42] C.-C. Wang, S. R. Kulkarni, and H. V. Poor, "Density evolution for asymmetric memoryless channels," *IEEE Trans. Inf. Theory*, vol. 51, no. 12, pp. 4216 –4236, Dec. 2005.

[43] I. Kim, Y. Zorian, G. Komoriya, H. Pham, F. Higgins, and J. Lewandowski, "Built in self repair for embedded high density SRAM," in *ITC*, 1998.

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

[45] A. J. Van de Goor, "Using march tests to test SRAMs," *IEEE Des. Test. Comput.*, vol. 10, no. 1, pp. 8–14, Mar. 1993.

[46] J. S. Yedidia, W. T. Freeman, and Y. Weiss, "Constructing free-energy approximations and generalized belief propagation algorithms," *IEEE Trans. Inf. Theory*, vol. 51, no. 7, pp. 2282–2312, Jul. 2005.

[47] A. T. Ihler, J. Fisher, and A. S. Willsky, "Loopy belief propagation: convergence and effects of message errors," *J. Mach. Learn. Res.*, vol. 6, no. 5, pp. 905–936, May 2005.

[48] J. M. Mooij and H. J. Kappen, "Sufficient conditions for convergence of the sum–product algorithm," *IEEE Trans. Inf. Theory*, vol. 53, no. 12, pp. 4422–4437, Dec. 2007.

[49] S. C. Tatikonda and M. I. Jordan, "Loopy belief propagation and gibbs measures," in *UAI*, 2002.

[50] M. I. Jordan, "Graphical models," *Statistical Science*, vol. 19, no. 1, pp. 140–155, Feb. 2004.

[51] T. G. Roosta, M. J. Wainwright, and S. S. Sastry, "Convergence analysis of reweighted sum-product algorithms," *IEEE Trans. Signal Process.*, vol. 56, no. 9, pp. 4293–4305, Sep. 2008.

[52] N. Noorshams and M. J. Wainwright, "Stochastic belief propagation: A low-complexity alternative to the sum-product algorithm," *IEEE Trans. Inf. Theory*, vol. 59, no. 4, pp. 1981–2000, Apr. 2013.

[53] Y. Li and L. Dolecek, "Effects of approximate representation in belief propagation for inference in wireless sensor networks," in *IEEE ACSSC*, 2013.

[54] C. K. Ngassa, V. Savin, and D. Declercq, "Min-sum-based decoders running on noisy hardware," in *IEEE Globecom*, 2013.

[55] A. Balatsoukas-Stimming and A. Burg, "Density evolution for min-sum decoding of LDPC codes under unreliable message storage," *IEEE Commun. Lett.*, vol. 18, no. 5, pp. 849–852, May 2014.

[56] J. Sloan, D. Kesler, R. Kumar, and A. Rahimi, “A numerical optimization-based methodology for application robustification: Transforming applications for error tolerance,” in *IEEE/IFIP ICDSN*, 2010.

[57] D. Williams, *Probability with martingales.* Cambridge university press, 1991.

[58] B. T. Polyak and A. B. Juditsky, “Acceleration of stochastic approximation by averaging,” *SIAM J. Control Optim.*, vol. 30, no. 4, pp. 838–855, 1992.

[59] A. Willsky, “Relationships between digital signal processing and control and estimation theory,” *Proceedings of the IEEE*, vol. 66, no. 9, pp. 996–1017, Sep. 1978.

[60] F. Chung and L. Lu, “Concentration inequalities and martingale inequalities: a survey,” *Internet Math.*, vol. 3, no. 1, pp. 79–127, Jan. 2006.

[61] P. F. Felzenszwalb and D. P. Huttenlocher, “Efficient belief propagation for early vision,” *Int. J. Comput. Vision*, vol. 70, no. 1, pp. 41–54, Oct. 2006.

[62] 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. Inf. Theory*, vol. 47, no. 2, pp. 657–670, Feb. 2001.

[63] M. Ardakani and F. R. Kschischang, “A more accurate one-dimensional analysis and design of irregular LDPC codes,” *IEEE Trans. Commun.*, vol. 52, no. 12, pp. 2106–2114, Dec. 2004.

[64] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, “Analysis and characterization of inherent application resilience for approximate computing,” in *IEEE/ACM DAC*, 2013.

[65] J. Ansel, Y. L. Wong, C. Chan, M. Olszewski, A. Edelman, and S. Amarasinghe, “Language and compiler support for auto-tuning variable-accuracy algorithms,” in *IEEE/ACM ISCGO*, 2011.

[66] M. Rinard, “Probabilistic accuracy bounds for fault-tolerant computations that discard tasks,” in *ACM ICS*, 2006.

[67] F. Frustaci, M. Khayatzadeh, D. Blaauw, D. Sylvester, and M. Alioto, “A 32kb SRAM for error-free and error-tolerant applications with dynamic energy-quality management in 28nm CMOS,” in *IEEE ISSCC*, 2014.

[68] U. Schlichtmann *et al.*, “Connecting different worlds: Technology abstraction for reliability-aware design and test,” in *DATE*, 2014.

[69] S. Ganapathy, G. Karakonstantis, R. Canal, and A. P. Burg, “Variability-aware design space exploration of embedded memories,” in *IEEE IEEEI*, 2014.

[70] C. I. Allen, J. Petrosky, and P. L. Orlando, “Effects of non-ionizing radiation on a 130 nm CMOS SRAM for low earth orbit applications,” in *IEEE NAECON*, 2014.

[71] C.-H. Huang, Y. Li, and L. Dolecek, “Belief propagation algorithms on noisy hardware,” *IEEE Trans. Commun.*, vol. 63, no. 1, pp. 11–24, Jan. 2015.

[72] I. Rish, “An empirical study of the naïve Bayes classifier,” in *IJCAI*, 2001.

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

[74] C. Condo, G. Masera, and P. Montuschi, “Unequal error protection of memories in LDPC decoders,” *IEEE Trans. Comput.*, to be published, 2014.

[75] C. Studer, N. Preyss, C. Roth, and A. Burg, “Configurable high-throughput decoder architecture for quasi-cyclic LDPC codes,” in *IEEE ACSSC*, 2008.

[76] M. May, M. Alles, and N. Wehn, “A case study in reliability-aware design: a resilient LDPC code decoder,” in *DATE*, 2008.

[77] C. Kameni Ngassa, V. Savin, E. Dupraz, and D. Declercq, “Density evolution and functional threshold for the noisy min-sum decoder,” *arXiv preprint*, 2014. [Online]. Available: http://arxiv.org/abs/1405.6594

[78] D. L. Donoho, A. Maleki, and A. Montanari, “Message-passing algorithms for compressed sensing,” *Proc. of the NAS*, vol. 106, no. 45, pp. 18 914–18 919, Sep. 2009.

[79] S. Sanghavi, D. Malioutov, and A. Willsky, “Belief propagation and lp relaxation for weighted matching in general graphs,” *IEEE Trans. Inf. Theory*, vol. 57, no. 4, pp. 2203–2212, 2011.