WHEN MACHINE LEARNING MEETS INFORMATION THEORY: SOME PRACTICAL APPLICATIONS TO DATA STORAGE

A Dissertation

by

PULAKESH UPADHYAYA

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Chair of Committee,	Anxiao (Andrew) Jiang
Committee Members,	Andreas Klappenecker
	Yoonsuck Choe
	Tie Liu
Head of Department,	Scott Schaefer

August 2020

Major Subject: Computer Engineering

ABSTRACT

Machine learning and information theory are closely inter-related areas. In this dissertation, we explore topics in their intersection with some practical applications to data storage.

Firstly, we explore how machine learning techniques can be used to improve data reliability in non-volatile memories (NVMs). NVMs, such as flash memories, store large volumes of data. However, as devices scale down towards small feature sizes, they suffer from various kinds of noise and disturbances, thus significantly reducing their reliability. This dissertation explores machine learning techniques to design decoders that make use of natural redundancy (NR) in data for error correction. By NR, we mean redundancy inherent in data, which is not added artificially for error correction. This work studies two different schemes for NR-based error-correcting decoders. In the first scheme, the NR-based decoding algorithm is aware of the data representation scheme (e.g., compression, mapping of symbols to bits, meta-data, *etc.*), and uses that information for error correction. In the second scenario, the NR-decoder is oblivious of the representation scheme and uses deep neural networks (DNNs) to recognize the file type as well as perform soft decoding on it based on NR. In both cases, these NR-based decoders can be combined with traditional error correction codes (ECCs) to substantially improve their performance.

Secondly, we use concepts from ECCs for designing robust DNNs in hardware. Non-volatile memory devices like memristors and phase-change memories are used to store the weights of hardware-implemented DNNs. Errors and faults in these devices (e.g., random noise, stuck-at faults, cell-level drifting *etc.*) might degrade the performance of such DNNs in hardware. We use concepts from analog error-correcting codes to protect the weights of noisy neural networks and to design robust neural networks in hardware.

To summarize, this dissertation explores two important directions in the intersection of information theory and machine learning. We explore how machine learning techniques can be useful in improving the performance of ECCs. Conversely, we show how information-theoretic concepts can be used to design robust neural networks in hardware.

DEDICATION

To my parents and sister Manjari.

ACKNOWLEDGMENTS

Firstly, I would like to thank my advisor Dr. Anxiao (Andrew) Jiang for his continued guidance and help. As a student who was new to the domain of research, his smiling attitude towards complicated research problems was enough to dispel my initial doubts. I consider myself extremely lucky to be one of his students. Without his help, this dissertation would have remained an elusive dream.

I would also like to thank the members of my dissertation committee, Dr. Andreas Klappenecker, Dr. Tie Liu, and Dr. Yoonsuck Choe for their valuable advice. I am deeply grateful to Dr. Krishna Narayanan for his inputs and his excellent lectures on coding theory, which made my journey easier. I was lucky to have collaborated with Dr. Jehoshua Bruck, Dr. Netanel Raviv, Dr. Hongchang Zhao, Dr. Erich Haratsch, Dr. Siddharth Jain, and Jin Sima. I would also like to express my gratitude to other collaborators from Texas A&M University: Dr. Ying Wang, Kunping Huang, Xiaojing Yu, Palash Parmar, Jacob Mink, Jeffrey Cordero, who helped me with their excellent insights into various research problems. I would also like to thank Dr. Tom V. Mathew at I.I.T. Bombay for instilling in me the initial spark for research.

I was lucky to have the constant support of my friends and family. They were kind and understanding during the most difficult times and continued inspiring me. I would especially like to thank my sister Manjari for her smiling reassurances. I would also like to thank my friends I made in College Station: Rajarshi, Arijit, Partha, Neha, Dharanidhar, Pranami, Shriti, Amlan, Prachee, Prithvi, Ranbir, Pradipta, Shantanu, Suman, Chinmoy, Mriganav, Nilkamal, Bobby, Ishan, Abhishek Das, Abhishek Sarmah, Jyotikrishna, Jay, Kaustubh, and Anubhav. I would also like to thank my friends from CSEGSA: Andrew, Scott, Seth, Chris, Dennis, Raniero, and Adam. I am deeply grateful to my friends outside College Station: Rakesh, Satam, Sushanta, Tanmay, Tanudeep, Ashutosh, and Yashwanth, who played an important role in making me believe in myself as a person.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a dissertation committee consisting of Dr. Anxiao Jiang, Dr. Andreas Klappenecker, Dr. Yoonsuck Choe of the Department of Computer Science and Engineering and Dr. Tie Liu of the Department of Electrical Engineering.

All other work conducted for the dissertation was completed by the student independently.

Funding Sources

Graduate study was supported by graduate research and teaching assistantships from Texas A&M University and funding from National Science Foundation.

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION

A lot of interesting research questions lie at the intersection of information theory and machine learning. Information-theoretic concepts have been used in developing theories of deep learning, designing robust training and inference algorithms, finding efficient compressed models, studying deep generative models, *etc.* Conversely, concepts from machine learning have been used in modulation, source coding, channel coding, and so on. This dissertation looks at both directions of the interplay between information theory and machine learning, with a focus on solving problems related to data storage in non-volatile memory (NVM) devices. Fig. 1.1 shows the broad topics covered by this dissertation in either direction. Firstly, we show how machine learning techniques can be used to correct errors in data stored in NVMs. Secondly, we also show how concepts from error-correcting codes can be used to protect stored weights and to design robust neural networks in hardware.

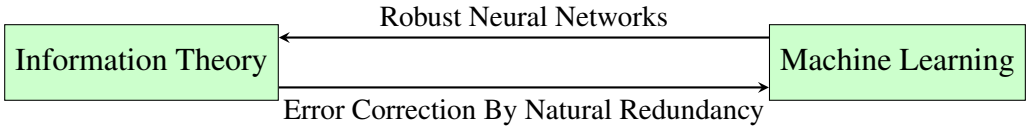

Figure 1.1: Interplay between machine learning and information theory.

1.1 Motivation

The first direction we explore in our research is to use machine learning for error correction in data. We live in the age of the big data revolution, and it has been estimated that over 90% of the total volume of existing data was produced in the last two years. Emerging data storage technologies are an integral part of this revolution. Non-volatile memories, such as flash memories, have been used almost ubiquitously in such data storage systems. Their popularity is due to the fact they are not only faster to read and write than traditional hard-disk drives, but also robust and less

prone to physical wear and tear. In addition, they are of smaller size and are more energy-efficient, which makes them good candidates for big-data storage.

However, these devices have some of their own limitations. As devices scale down, various types of noise might accumulate in these devices over time, making their long-term reliability a critical challenge [2, 3]. For example, inter-cell interference is a significant source of errors in flash memory cells. Additionally, flash memory cells wear out after a limited number of reads and writes. Other devices such as phase-change memories (PCM) also suffer from write endurance issues [4].

The traditional approach to improve data reliability is to use error-correcting codes (ECC), which add external redundancy (e.g., parity bits/symbols) to correct errors in data [5, 6]. Usually, data is first compressed by a source coding scheme before external redundancy is added by an ECC encoder [7]. However, there is a need to improve the long-term reliability of data, while at the same time reducing the storage overhead.

In this dissertation, we present a novel approach to improve data reliability, in case traditional ECC decoding fails. A lot of internal redundancy remains in data even after compression (because of practical issues such as sub-optimal compression based on local features). We call such redundancy *natural redundancy* (NR). For example, let us consider data compression for the English language. The compression ratio of the well-known character-based Huffman code is 4.59 bits per character, which is significantly higher than Shannon's estimate of the entropy of printed English (1.34 bits per character) [8]. If we consider Shannon's prediction as a standard, 71% of character-based Huffman-compressed data consists of NR.

An information system can utilize such NR in two distinct ways. The first way is to develop enhanced compression schemes, which often use deep learning to further compress the data by removing leftover redundancy [9, 10]. This is a popular and active research area, and better compression ratios have been achieved for high distortion regimes. A second way, which we explore in this dissertation, uses NR for error correction. New error-correcting decoders are designed to mine NR in data, use it for error correction, and eventually combine it with ECC decoders for

enhanced performance. The main motivation behind using NR for error correction comes from the fact that data reprocessing might prove costly for storage systems, which already store large volumes of data. This dissertation explores schemes that make most of the information stored in devices (which already contains NR) and uses it as a rich resource for error correction.

We explore two distinct schemes for error correction using NR. In the first scheme, which we cover in Chapters 3, 4, and 5 of this dissertation, the NR-decoder is aware of the compression schemes and the type of the source. The NR-decoder then uses machine learning and algorithmic techniques to correct errors. We call such a scheme a *representation-aware* scheme. In the second scheme, which we explore in Chapter 6 of this dissertation, we do not need any prior knowledge of how data compression and representation in files. We call such a scheme a *representation-oblivious* scheme. This approach is particularly useful for storage systems where error correction happens at a lower level in device controllers. These controllers do not have access to data compression schemes or file types, which makes this approach more practical for use in such storage systems.

However, some key challenges exist in using NR for error correction. NR is usually complex and unstructured. In addition, the nature of such residual redundancy varies from source to source. (For example, language-based data has a different type of NR as compared to images.) To overcome these challenges, the representation-aware schemes explored in this dissertation use both algorithmic and machine learning techniques for error correction. We show analytically how such schemes improve the erasure thresholds of ECCs. On the other hand, our representation-oblivious scheme makes use of deep learning techniques to identify the file type and then perform soft decoding on that file type. We show experimentally how the soft decoding result is then combined with an ECC decoder for improved performance (especially when the bit error rate exceeds the threshold of the ECC decoder).

To summarize, in the first direction, we explore machine learning techniques to improve the performance of traditional ECC decoders. In the second direction in our research, we explore ideas from information theory for building robust hardware-implemented neural networks. Deep neural networks (DNNs) are an important tool in artificial intelligence (AI) and have been used in a wide

range of applications, such as computer vision, robotics, recommendation systems, healthcare, environmental monitoring, *etc.* In recent times, the implementation of DNNs in hardware has become a popular area of research [11, 12, 13, 14, 15, 16, 17, 18]. Hardware-implemented DNNs are used in sensors, phones, security cameras, wearable devices, and so on. These techniques are advancing rapidly and will make artificial intelligence (AI) systems more pervasive in the future [19, 20].

However, several reliability issues might affect the performance of hardware-implemented DNNs. As integrated circuits (ICs) move towards smaller feature sizes and DNNs are used in extreme environments for long periods of time, they might suffer severe stability and endurance issues [21, 22, 23, 24]. For example, the non-volatile memories that are used to store analog weights of such DNNs, such as phase-change memories or memristors, have well-known challenges in cell-level drifting, data retention, sneak-path interferences [25, 26, 27]. Such issues can cause significant performance issues and data loss. If these issues are not resolved, the application of hardware-implemented DNNs in critical areas (e.g, healthcare, public safety, industrial production *etc.*) will remain less convenient.

In this dissertation, we propose two approaches to make neural networks robust based on analog error-correcting codes. In the first approach, which we term *external*, analog codes are used to protect the analog weights of hardware-implemented neural networks. In the second approach, which we term *internal*, we use concepts from analog codes to convert neural networks to an inherently robust form.

1.2 Research Contributions

Table 1.1 shows the topics covered by this dissertation in two broader areas, (a) error correction by natural redundancy, and (b) robust neural networks. These topics are summarized below:

1.2.1 Learning Techniques for Correcting Errors by Natural Redundancy

Digital storage systems store huge volumes of data. However, as devices scale down to smaller sizes, there emerge significant challenges with respect to long time reliability of data. The usual

Error Correction by NR	Robust Neural Networks
(a) Learning Techniques for Error Correction Using NR (b) LDPC decoding with NR (c) Stopping set elimination of LDPC codes using NR (d) Representation Oblivious Error Correction by NR	(a) Externally Coded Neural Networks (b) Internally Coded Neural Networks

Table 1.1: We cover two main topics:(a) Error Correction by Natural Redundancy, (b) Robust Neural Networks.

approach in correcting errors in storage systems is the use of error-correcting codes, which add structured redundancy to data and increase the overall reliability. In our work, we try a different approach [28, 29]. A lot of redundancy remains in data even after compression. The primary reason for it is the fact that compression algorithms have to take into account computational complexity, and therefore do not completely remove the redundancy from data. We design algorithms and learning techniques to use this residual NR in data for error correction. We then combine our approach with error-correcting codes to further improve the performance. In this work, we explore several aspects of natural redundancy in data. The first contribution is the development of effective algorithms that discover NR in language-based compressed data. We also discuss the efficient decoding of codes with random structures and the capacity of error-correcting codes in the presence of natural redundancy. We then examine the trade-off between the time complexity of source and channel decoding.

1.2.2 LDPC Decoding with Natural Redundancy

The previous work shows how NR is a significant resource that can be utilized for error-correction. In this work [30], we examine the problem more theoretically. We study the combination of NR-based decoders with LDPC codes and show that the error-correction capability of the combined decoder is significantly enhanced. We derive analytical equations for the density evolution of LDPC codes when side information is available from the NR-decoder. We also propose a theoretical model for compressed languages and study the performance of an iterative scheme, where the NR-decoder and the LDPC decoder perform decoding in multiple iterations. We also

present theoretical results which show the upper bound to the code sizes of error-correcting codes in the event that they are assisted by NR-based decoders.

1.2.3 Stopping Set Elimination of LDPC Codes by Language-based Natural Redundancy

In the previous work, we consider a decoding scheme which consists of two decoders: ECC-Decoder and NR-Decoder, which work collaboratively to correct errors in data. In this work [31], we explore another generic decoding model, which is motivated by language-based NR decoders. In this case, we look at fixed-length LDPC codes and the Stopping-Set Elimination Problem for these codes. Given erasures in data, decoding in LDPC codes fails if every parity-check equation involves at least two erasures. Such a set of erasures is called a Stopping Set. NR-based decoders can be used to recover such erasures. However, such decoders are usually of higher complexity, therefore we want to use them to remove the fewest number of erasures from the stopping set so that the belief propagation decoding algorithm of LDPC codes (which has lower complexity) can decode the remaining erasures in k iterations. We study cases where $k = \infty$ and prove that such a problem is NP-hard. We also present an approximation algorithm when $k = 1$, and design efficient exact algorithms for general k when the stopping graphs of these stopping sets form trees.

1.2.4 Deep Learning for Representation-Oblivious Error Correction by Natural Redundancy

In the previous works, we studied representation-aware schemes, where the NR-decoder is aware of the compression algorithms and uses that knowledge to perform decoding accordingly. In this work [32], we present a new scheme which is representation-oblivious. In such a scheme, the decoder does not have any prior knowledge of data representation (e.g., data compression algorithms, mappings from symbols to bits, metadata, and so on). Such an approach makes it convenient to use in storage systems, where error correction is a low-level operation, usually executed by the controller. We show how such a scheme can use deep learning to identify the file type, and perform soft decoding on noisy file segments based on the natural redundancy of the recognized file type. We then combine the soft decoding results with belief propagation decoder of a high-rate

fixed-length LDPC code (typical for storage systems). The results of our experiments demonstrate the efficiency of the scheme in correcting errors in data, especially when the bit error rates in these file segments are significantly higher than the thresholds of the given LDPC code.

1.2.5 Externally Coded Neural Networks

In recent times, there has been a lot of interest in the implementation of neural networks in hardware. When the analog weights of a neural network are stored in hardware devices (e.g., memristors), various types of noise will accumulate, leading to degradation in their classification (or regression) performance. In our work [33], we study the use of analog codes for correcting the noisy weights of a neural network. Such a scheme is external, where the internal structure of the neural network remains the same. Specifically, we study the performance of linear analog codes in systematic forms. The results show that analog codes not only improve performance but also allow graceful degradation of performance under Additive White Gaussian Noise (AWGN).

1.2.6 Internally Coded Neural Networks

Deep Neural Networks (DNNs) have been used ubiquitously in artificial intelligence, resulting in revolutionary impact in many applications, including mission-critical ones. However, their intrinsic properties are not easily explained. In recent times, it has been shown how DNNs are sensitive to various kinds of noise, whether adversarial or random. This makes it important to address the issue of their robustness, especially if they are to be deployed in critical applications like autonomous driving and under extreme conditions. With an eye on solving these problems, we propose the construction of robust DNNs with the help of concepts from coding theory. In such applications, either the data or the internal DNN layers are coded using error-correcting codes, which guarantee robust computation in the presence of noise. In this work [34, 35], we focus on linear analog codes. In contrast to many existing solutions, we do not alter the training algorithms to ensure robustness. We only transform an already trained neural network into a coded form.

1.3 Organization of the Dissertation

The rest of the dissertation is organized as follows. In Section 2, we review two fundamental approaches to solving problems in the intersection of information theory and machine learning. In Section 3, we study error correction schemes that use natural redundancy in data to correct errors. In Section 4, we study the sequential and iterative combination of natural redundancy decoders with LDPC codes, and present results of density evolution. In Section 5, we study the problem of interaction of language-based natural redundancy decoders with fixed length LDPC codes, and study the problem of stopping set elimination is such codes. In Section 6, we study a deep learning-based representation-oblivious scheme for error correction using natural redundancy. In Section 7, we study error correction in hardware-implemented neural networks by using error-correcting codes. In Section 8, we study robust neural networks for coded classification. In Section 9, we present the conclusions and future work.

2. TWO FUNDAMENTAL APPORACHES: INTERPLAY BETWEEN INFORMATION THEORY AND MACHINE LEARNING

2.1 Background

The journeys of information theory and machine learning have always been closely intertwined. David Mackay, in his seminal book describes these fields as “two sides of the same coin" [36]. These fields, though apparently independent, have influenced each other in more ways than one. However, these similarities should not be too surprising either, as both these disciplines have been fundamentally influenced by statistics. For instance, both machine learning algorithms and channel decoders fundamentally perform statistical inference tasks.

Historically, information theorists were interested in the problem of making computers perform tasks that were normally performed by human beings. In the 1960s, when the foundations of learning and information theory were both emerging, people were interested in fundamental problems that were common to both fields. Claude Shannon, who is known as the “father of information theory", looked into some interesting problems, such as making a mouse-like device solve a maze, or programming a computer how to play chess [37], and believed that these seemingly insignificant problems would provide meaningful insights into making machines perform more complicated tasks, such as making a melody, or performing mathematical operations. Norbert Weiner [38] believed areas such as control, communication, and learning should be considered a part of a larger field of study known as *cybernetics*.

In the ’80s and the ’90s there were also works from the information-theoretic community on using neural networks for solving problems in information theory. For instance, Bruck *et al.* studied how maximum-likelihood decoding of linear block codes and neural network optimization were related problems [39]. Another work proposed a generalized convergence theorem for neural networks [40]. The capacity of densely-connected associative neural networks was studied in [41]. Bichsel and Seitz viewed neural networks as multistage encoders and suggested

information-theoretic concepts like conditional class entropy to find the optimal number of neurons in hidden layers [42]. Another information-theoretic approach to finding the optimal number of hidden layer neurons in stochastic feed-forward neural networks was also studied in [43]. Kanter suggests an information-theoretic method of estimating the maximal capacity per weight for a two-layered discrete neural network when the magnitude of the weights was bounded [44]. The use of information theory in the analysis of neural coding of brains has also been studied extensively [45]. Hoffman, with the help of algorithmic information theory, predicted how the task of learning would require either complex programs or providing programs with a lot of data [46].

In the present age, when machine learning has played a central role in the information revolution, concepts like entropy, cross-entropy, mutual information, or Kullback-Leibler Divergence (KLD) have been widely used in machine learning literature. Cross entropy is a widely-used loss function for classification models like logistic regression [47]. Estimates of mutual information have been used in feature selection problems [48]. Divergence measures such as KLD are useful in unsupervised learning methods like variational autoencoders [49].

2.2 Recent Work

In addition to the work in the past, there has been a renewed interest in exploring ideas at the intersection of machine learning and information theory. Some of the most recent works in the intersection of information theory and machine learning are reviewed in this sub-section. We explore recent work in both directions; (1) machine learning for information theory, and (2) information theory for machine learning.

2.2.1 Machine Learning for Information Theory

Machine learning techniques have become widely popular among information theorists in recent times. This is because machine learning can be used to solve complex problems like channel estimation, and the design of trainable communication systems holds promise for future applications in 6G/7G systems. Neural networks can be used to perform iterative quantization and optimization to build better and smarter communication system models[50]. Trainable communication

systems have been explored to design deep learning-based decoders in receivers that decode signals with in-phase and quadrature imbalance [51]. Machine learning has also been used in performing channel estimation in orthogonal frequency division multiplexing systems with lower complexity and storage overhead [52]. Online meta-learning techniques can be used to perform fast end-to-end training of encoders and decoders over fading noisy channels [53]. Recent works have also proposed training for autoencoders based on Wasserstein General Adversarial Networks(GANs) in practical over-the-air setups [54].

Learning techniques also improve the performance of error-correcting codes, such as LDPC codes, Polar codes, Reed Solomon codes, Turbo codes, Convolutional codes, and so on. In [55], neural decoders are built for both linear and non-linear block codes, and a good generalization property is obtained, wherein training at a particular SNR (0 dB) works well across a wide range of SNRs (-5 dB to 7.5 dB). Iterative soft decoding of Reed-Solomon codes can also be improved using deep learning algorithms [56].

Deep learning has been shown to improve the performance of alternating direction method of multipliers (ADMM) decoders over traditional LDPC decoding schemes [57]. Recurrent quantized neural networks have also been shown to design low-precision linear finite alphabet iterative decoders for LDPC codes, with better performance, lower complexity, and faster convergence than floating-point BP algorithms. [58]. Channel decoding based on a combination of complex-valued CNNs with traditional BP algorithms has also been suggested to reduce correlated noise in a channel. Such a scheme improves upon the performance of the traditional BP algorithm [59]. Reinforcement-learning based methods have also been suggested to improve sequential decoding decisions in channel decodings. Such iterative decoders can become adaptive to their current state [60].

An attention-based one-shot decoding scheme for Polar codes has been studied, which uses deep learning to be able to decode shorter codes with low computational complexity [61]. A convolutional neural network for decoding polar codes is proposed in [62], and the decoding process is shown to be faster for shorter codes, whose block length is 32. The work [63] shows how polar

neural decoders usually work well on the binary symmetric channel and binary erasure channels, however, they are less robust to the mismatch in training/validation statistics for binary asymmetric channel and suggests alternative constructions of such neural decoders using domain adaptation techniques. Syndrome-enabled unsupervised learning at the receiver has also been shown to be useful in improving the performance of a Polar decoder [64]. Novel deep learning techniques such as transfer learning have also been used to train neural network decoders. Such a scheme seeks to address the problem of underfitting in neural Polar decoders [65].

A coding scheme for 1-bit receivers is developed using deep learning and outperforms Turbo codes for finite block lengths [66]. Feedback Auto Turbo Encoders with CNN architectures have been used to combine interleaver and iterative decoding to show a significantly improved performance over traditional turbo codes [67]. Autoencoders have also been used for joint coding and modulation schemes for codes with very short block lengths, with a focus on Bernoulli-Gaussian impulsive noise channels [68].

A convolutional decoder based on convolutional neural networks (CNNs) that performs comparably to the Viterbi soft decoding algorithms is proposed [69]. Farsad *et al.* [70] consider the problem of symbol detection in different channels and implement well-known algorithms by replacing channel state information with data-driven machine learning methods. These methods work in a hybrid manner with known decoding algorithms like Viterbi and Bahl-Cocke-Jelinek-Raviv (BCJR) algorithms. Neural architectures such as fully convolutional U-nets have been used to design efficient decoders for convolutional codes over AWGN channels [71]. Deep neural networks have also been proposed to design a posteriori probability detector to replace trellis-based BCJR or Viterbi algorithms for two-dimensional magnetic recording. This scheme also has a lower per-bit latency [72].

Autoencoding deep neural networks have also been used for symbol demapping and decoding and such schemes have been shown to produce lower error rates when compared with traditional schemes based on constellation demapping and LDPC decoding [73]. Deep learning-based end-to-end systems have also been suggested in [74], where the neural encoder performs joint modulation

and encoding, and the neural decoder performs joint decoding and demodulation.

Joint source-channel coding has been suggested for images using auto-encoders and such schemes have several advantages like graceful degradation and adaptiveness to different channels [75]. For bandwidth-limited channels, the problem of learning an optimal joint compression and error correction scheme facilitated by a neural network has been studied, which leads to lower distortion compared to a separate source-channel scheme, for a wide range of SNRs [76].Neural architectures have also been proposed for bit-interleaved coded modulation systems, and new joint probabilistic and geometric shaping using neural networks significantly outperform traditional schemes [77].

In optical communication for unmanned aerial vehicles (UAVs), where channel estimation is a complex task, conditional Generative Adversarial Networks can be used to simulate real channels [78]. The system is end-to-end and the transmitter and receiver are implemented using DNNs. For visible light communications, a deep learning-based run-length limited decoder performs comparably to existing schemes, while reducing computational complexity [79]. A deep learning decoder has also been suggested for free space optical (FSO) channels, and these decoders have been shown to be stable in the presence of turbulence [80]. Satorras *et al.* introduce graph neural networks for factor graphs, and suggest improved algorithms that combine the advantages provided by belief propagation and graph neural networks [81]. Such a scheme outperforms the traditional belief propagation scheme for LDPC codes over bursty channels.

Deep learning has also been used in areas like lattices. A DNN model has been proposed which takes the lattice generator matrix and sphere radius as input to count the number of lattice points in a sphere with 80% accuracy [82]. A neural estimator has also been suggested to estimate information-theoretic quantities like the transfer entropy metric [83].

The use of machine learning techniques can provide important tools to solve many problems in communication systems in the future. Such techniques will not only be useful not only in the application layer but also in medium access layers and application layers [84]. Such techniques can also be explored in the context of storage systems, which store big data. In addition to the

work cited here, there have been various works over time, which have looked at the idea of using machine learning techniques for solving traditional information-theoretic problems.

2.2.2 Information Theory for Machine Learning

Information theorists have also been interested in using concepts from information theory to understand the black box of deep learning. One interesting approach is the use of information bottleneck theory in [85, 86]. These works claim that the training process in deep learning can be seen as preserving mutual information about the target output from the input. Learning is supposed to proceed in two stages, a fitting phase followed by a compression phase [86]. However, a recent work [87] shows how such notion of compression holds only for the *tanh* activation function and not for *ReLU*, which is the most widely used activation function in recent times. This work outlines the fact that noise assumptions are important in the application of information theory in explaining the generalization performance of deep learning. A tractable method of estimating entropy and mutual information between different layers of DNNs has also been proposed [88].

An approximate Fisher information-based method for characterizing stochastic gradient descent training for DNNs has also been explored [89]. Ideas such as Huffman codes have been used in algorithms designed for model compression and quantization, for efficient implementation of DNNs in hardware [13]. Information theory has been useful in improving the performance of general adversarial networks (GANs) in using unsupervised learning to learn disentangled representations [90]. A mutual information-based approach to ensure fair and robust training in the presence of noise and poisoning has been explored in [91].

Neural network weights are generally analog values. High-density information storage devices can be used to store the weights of neural networks implemented in hardware. Analog codes have been shown to be useful for such devices. It achieves a higher capacity than discrete coding schemes and also has other added benefits like lower complexity and energy efficiency [92]. When analog weights of a neural network are stored in devices such as memristors, analog codes could be a viable option for error correction. Coded computation can be used to build robust distributed systems, which are resilient to failures and delays. Machine learning can be used within the coded

computation framework to exhibit resilience to non-linear computations [93]. There have also been other notable works in this area, and we skip the details for our convenience.

2.3 Importance of this Dissertation

Our review shows how a wide range of topics in the intersection of machine learning and information theory have been explored recently. However, this dissertation looks at some unique, yet practical problems, which have not been deeply explored in the past. The importance of this dissertation lies in the fact that we look at some practical problems from the point of view of data storage, and suggest novel solutions. Data storage systems are going to be an integral part of new information systems in the future to not only store big data but also as part of devices that perform artificial intelligence tasks. The use of natural redundancy for error correction has been explored in innovative ways, with ideas from several areas, such as algorithmic techniques, denoising, convolutional neural networks, statistical language processing, autoencoding neural networks, joint source-channel coding, *etc.* This makes our approach easily generalizable to other related areas, such as distributed storage and satellite communication systems. Additionally, we look into the problem of building robust neural networks which perform reliable computation in the presence of noise.

3. CORRECTING ERRORS BY NATURAL REDUNDANCY USING LEARNING TECHNIQUES [1]

3.1 Introduction

The storage of big data has become increasingly important. Every day a large amount of data – 2.5 billion GB – is generated. However, its long-term reliability has significant challenges. For example, non-volatile memories (NVMs), such as flash memories and phase-change memories, store a substantial portion of big data due to their fast speed, physical robustness and large storage capacity. However they have data retention problems, where charge leakage or cell-level drifting makes data more noisy over time. Operations such as reads and writes cause accumulative disturbance in NVM data. Furthermore, erasures of NVM cells degrade cell quality and make cells more prone to errors over time. There is a strong motivation in elevating the long-term reliability of big-data storage to the next level.

The most effective way to protect data has been error-correcting codes (ECCs). By adding redundancy to data in a disciplined way, errors can be effectively corrected. We call such redundancy *artificial redundancy*. The recent advancement in learning and the availability of big data for study have offered a new opportunity for error correction: *to use the natural redundancy in data for error correction*. By natural redundancy (NR), we refer to the inherent redundancy in data that is not artificially added for error correction, such as various features in languages and images, structural features in databases, etc. Due to practical reasons (e.g., high complexity for compression, and lack of precise models for data), even after compression, lots of redundancy often still exist.

This section studies how to use the natural redundancy in data for error correction, with a focus on languages and images. It is a topic related to joint source-channel coding and denoising. The idea of using the leftover redundancy at a source encoder to improve the performance of ECCs has been studied within the field of joint source-channel coding (JSCC) [94, 95, 96, 97, 98,

[1] ©IEEE 2017. Parts of this section are reprinted, with permission, from A. Jiang, P. Upadhyaya, E. F. Haratsch and J. Bruck, "Correcting errors by natural redundancy," 2017 Information Theory and Applications Workshop (ITA), San Diego, CA, 2017.

99, 100, 101, 102]. However, not many works have considered JSCC specifically for language-based sources, and exploiting the redundancy in the language structure via an efficient decoding algorithm remains as a significant challenge. Related to JSCC, denoising is also an interesting and well studied technique [103, 104, 105, 106, 107, 108, 109, 110, 111]. A denoiser can use the statistics and features of input data to reduce its noise level for further processing.

This section explores several new topics on NR: the efficient list decoding of random codes; the error-correction capability of ECCs with NR; and the computational-complexity tradeoff between source and channel coding.

3.2 Sampling-based Decoding for Random Codes

3.2.1 Sliding-Window Decoder for Prefix-free Codes

Prefix-free codes are another important choice for compression. In [1, 112, 113, 114], Huffman codes for English-text characters were used for the study of NR, where every character (letter or punctuation mark) is represented by a variable-length Huffman codeword. A significant challenge for NR-decoding is that the compressed file does not specify the boundaries of Huffman codewords, making it difficult to recognize words/phrases, especially for high BER. To address the problem, we propose a *sliding-window decoding* technique: use a sliding window (of a variable size) to check different segments of the noisy compressed file; and if by flipping at most a few bits, the bits in the window can be decompressed as a long yet relatively common word/phrase (such as "information"), then this solution is highly likely to be correct, because long words/phrases are extremely sparse. The latter point is shown in the following example.

Example 1. *Consider lower-case words. Assume there are M_n words whose Huffman codewords have n bits. Then the* density *of such words is $D_n = \frac{M_n}{2^n} = 10^{-x_n}$. We show M_n and x_n (collected from Wikipedia, a very large text corpus) in Fig. 3.1. It can be seen that the word density decreases exponentially fast for large n. So long words are very sparse.* □

The sliding-window technique can enhance existing NR-decoders, such as the Word-Recognition NR-decoding algorithm in [1]. The key is how to correct errors inside a window both efficiently

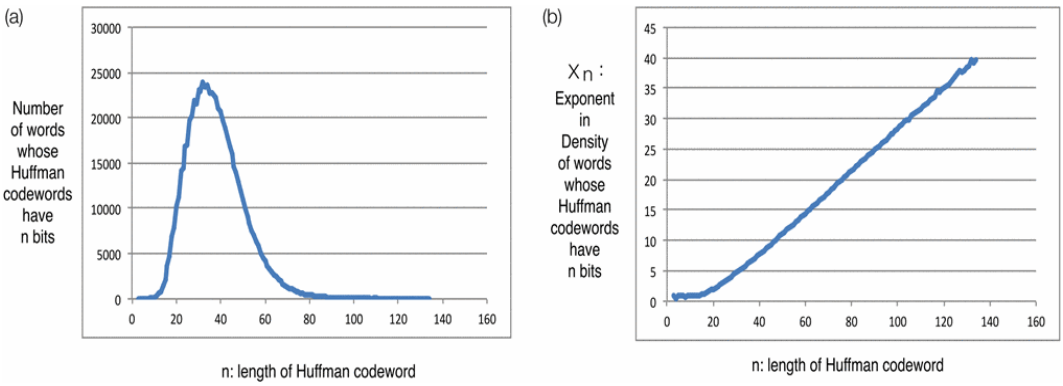

Figure 3.1: (a) The number of words M_n whose Huffman codewords have n bits. (b) The exponent x_n in the density 10^{-x_n} for words whose Huffman codewords have n bits.

and reliably, without exhaustive search. That leads to the random-code decoding algorithm below.

3.2.2 Sampling-based Decoder for Random Codes

Consider a window of n bits. It can have 2^n possible values; however, only a small subset of them $\mathcal{C} \subset \{0,1\}^n$ correspond to valid words/phrases. Let $\mathbf{y} = (y_1, y_2, \cdots, y_n) \in \{0,1\}^n$ be the bit-string we receive in the window, and let $t < n$ be an integer parameter. Assuming that there exists a codeword $\mathbf{x} \in \mathcal{C}$ with Hamming distance $d_H(\mathbf{x}, \mathbf{y}) = t$, we would like to find $\mathbf{x}$ with high probability and low time complexity. Note that an exhaustive search will have complexity $\binom{n}{t} = O(n^t)$, which is exponential in t and inefficient in practice.

We start by defining a sampling function. Given a set $P \subseteq \{1, 2, \cdots, n\}$, the *sampling function* $f_{n,P} : \{0,1\}^n \to \{0,1,?\}^n$ is: $\forall\ \mathbf{b} = (b_1, b_2, \cdots, b_n) \in \{0,1\}^n$, we have $f_{n,P}(\mathbf{b}) = (b'_1, b'_2, \cdots, b'_n)$ where each $b'_i = b_i$ if $i \in P$ and $b'_i = ?$ otherwise. (Here "?" represents an unsampled bit.) Define the *sample-value set* as $V_{n,P} \triangleq \{(b_1, b_2, \cdots, b_n) \in \{0,1,?\}^n | \text{for } 1 \leq i \leq n, b_i = ? \text{ if } i \notin P\}$. We have $|V_{n,P}| = 2^{|P|}$. For any sample value $\mathbf{s} \in V_{n,P}$, let $E_{n,P}(\mathcal{C}, \mathbf{s}) \subseteq \mathcal{C}$ be the subset of codewords that, when sampled by $f_{n,P}$, have the sample value $\mathbf{s}$; that is, $E_{n,P}(\mathcal{C}, \mathbf{s}) \triangleq \{\mathbf{b} \in \mathcal{C} \mid f_{n,P}(\mathbf{b}) = \mathbf{s}\}$. $E_{n,P}(\mathcal{C}, \mathbf{s})$ may contain zero, one or more codewords of $\mathcal{C}$.

We build a data structure called *sample dictionary* $Dic_{n,P}(\mathcal{C}) \triangleq \{(\mathbf{s}, E_{n,P}(\mathcal{C}, \mathbf{s})) \mid \mathbf{s} \in V_{n,P}\}$. It is a set of $(key, value)$ pairs, where each key is a sample value $\mathbf{s} \in V_{n,P}$, and its corresponding

value is the set $E_{n,P}(\mathcal{C}, \mathbf{s})$. Given $\mathbf{y}$, if its sample value $f_{n,P}(\mathbf{y})$ matches that of $\mathbf{x}$, we can use it to look up the dictionary, and find $\mathbf{x}$ in the set $E_{n,P}(\mathcal{C}, f_{n,P}(\mathbf{y}))$.

We now generalize the above discussion to k sampling functions f_{n,P_1}, f_{n,P_2}, $\cdots$, f_{n,P_k}. By choosing a suitable k, a good balance between decoding complexity and decoding-success probability can be achieved. Let P_1, P_2, $\cdots$, P_k be k subsets of $\{1, 2, \cdots, n\}$. We can use the k corresponding sampling functions $f_{n,P_1}, f_{n,P_2}, \cdots, f_{n,P_k}$ to sample the code $\mathcal{C}$, and get an *aggregated sample dictionary* $Dic_{n,P_1,\cdots,P_k}(\mathcal{C}) \triangleq \{(i, \mathbf{s}, E_{n,P_i}(\mathcal{C}, \mathbf{s})) \mid 1 \leq i \leq k, \mathbf{s} \in V_{n,P_i}\}$. We build the dictionary before decoding as *preprocessing*.

Define the *candidate codewords* for $\mathbf{y}$ as $Cand_{n,P_1,\cdots,P_k}(\mathcal{C}, \mathbf{y}) \triangleq \bigcup_{i=1}^{k} E_{n,P_i}(\mathcal{C}, f_{n,P_i}(\mathbf{y}))$. They are codewords that match $\mathbf{y}$ for at least one of the k sampling functions. So we can also define it as $Cand_{n,P_1,\cdots,P_k}(\mathcal{C}, \mathbf{y}) \triangleq \{\mathbf{b} \in \mathcal{C} \mid \exists 1 \leq i \leq k$ such that $f_{n,P_i}(\mathbf{b}) = f_{n,P_i}(\mathbf{y})\}$.

We now describe our decoding strategy: given a received string $\mathbf{y} \in \{0,1\}^n$, use the k sample values $f_{n,P_1}(\mathbf{y})$, $f_{n,P_2}(\mathbf{y})$, $\cdots$, $f_{n,P_k}(\mathbf{y})$ as keys to look up values in the dictionary $Dic_{n,P_1,\cdots,P_k}(\mathcal{C})$, and return a set of candidate codewords $Cand_{n,P_1,\cdots,P_k}(\mathcal{C}, \mathbf{y})$. (The candidate codewords will be filtered further based on their Hamming distance to $\mathbf{y}$, the frequencies of its words/phrases in training texts, co-location relationship with phrases elsewhere, etc., and be combined with an existing NR-decoder such as [1]. Our objective here is to include $\mathbf{x}$ as a candidate codeword.)

We now analyze two aspects of the decoding strategy's performance: its probability of including $\mathbf{x}$ as a candidate codeword, and its expected time complexity. Given any $\mathbf{a} = (a_1, \cdots, a_n) \in \{0,1\}^n$ and $\mathbf{b} = (b_1, \cdots, b_n) \in \{0,1\}^n$, define $M(\mathbf{a}, \mathbf{b}) = \{i \mid 1 \leq i \leq n,\ a_i = b_i\}$. We have $|M(\mathbf{a}, \mathbf{b})| = n - d_H(\mathbf{a}, \mathbf{b})$. Let $P_{IN}(t)$ denote the probability that the target codeword $\mathbf{x}$ is among the candidate codewords, namely, $P_{IN}(t) \triangleq Pr\{\mathbf{x} \in Cand_{n,P_1,\cdots,P_k}(\mathcal{C}, \mathbf{y}) \mid \mathbf{x} \in \mathcal{C},\ d_H(\mathbf{x}, \mathbf{y}) = t\}$.

Lemma 2. *$\forall \mathbf{b} \in \mathcal{C}$, $\mathbf{b} \in Cand_{n,P_1,\cdots,P_k}(\mathcal{C}, \mathbf{y})$ if and only if there exists $i \in \{1, 2, \cdots, k\}$ such that $P_i \subseteq M(\mathbf{b}, \mathbf{y})$.*

$\mathcal{C}$ is an unstructured code. To facilitate analysis, we assume the following random model: Let $\mathcal{C}$ be a random code, whose codewords are chosen independently and uniformly at random from the vector space $\{0,1\}^n$. In addition, let $|P_1| = |P_2| = \cdots = |P_k| = m$ for some $m \leq n - t$, and

let each P_i independently choose its m elements uniformly at random from $\{1, 2, \cdots, n\}$ without replacement.

Theorem 3. $P_{IN}(t) = 1 - \left(1 - \frac{\binom{n-t}{m}}{\binom{n}{m}}\right)^k$.

The time complexity of decoding is determined by the number of candidate codewords we need to examine. For $i = 1, 2, \cdots, k$, the i-th sampling function f_{n,P_i} samples m bits of the received string $\mathbf{y}$, and uses the sample $f_{n,P_i}(\mathbf{y})$ to look up the set of candidate codewords $E_{n,P_i}(\mathcal{C}, f_{n,P_i}(\mathbf{y}))$ in the dictionary. So the total number of candidate codewords to examine (with possible overlapping for different sampling functions) is $\sum_{i=1}^{k} |E_{n,P_i}(\mathcal{C}, f_{n,P_i}(\mathbf{y}))|$. Let $\mu(t)$ denote the *expected* number of candidate codewords to examine given that there exists a codeword $\mathbf{x} \in \mathcal{C}$ with $d_H(\mathbf{x}, \mathbf{y}) = t$.

Theorem 4. $\mu(t) = k\left(\frac{\binom{n-t}{m}}{\binom{n}{m}} + (|\mathcal{C}| - 1)\left(\frac{1}{2}\right)^m\right)$.

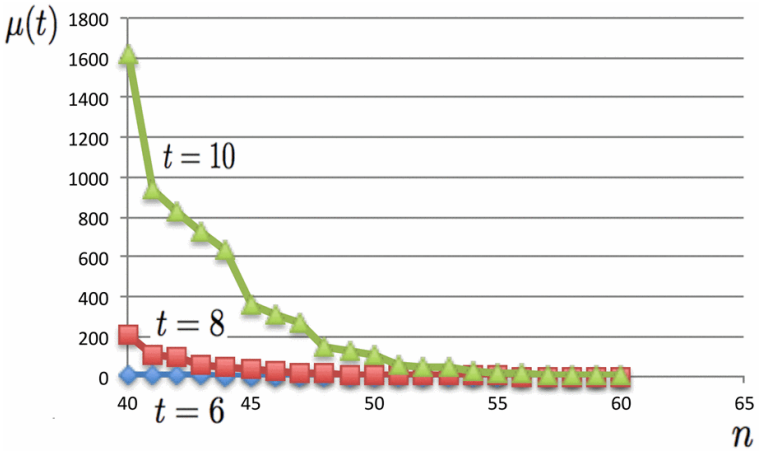

Figure 3.2: Performance of sampling-based decoder for random codes. Here the x-axis is n, and the y-axis is the minimum value of $\mu(t)$ for which there exists a feasible solution to k and m given the condition that $P_{IN}(t) \geq 0.99$ for $t = 6$, 8 and 10.

Since long words/phrases are very sparse, $\mu(t)$ can be quite small for large n. Given n, $|\mathcal{C}|$ and t, we choose the parameters k and m to achieve a good balance between decoding complexity

and success rate. For example, when $n = 48$ and $t = 10$, for n-bit English words compressed by Huffman coding (for example, "information" is such a 48-bit word, and there are $|\mathcal{C}| = 12,895$ such words), we can choose parameters m and k such that only less than 250 (instead of $\binom{48}{10} \gg 250$) candidate words need to be checked on average, and the correct word is included in the checked words with probability $P_{IN}(t) \geq 0.99$. We show more results in Fig. 3.2, where we let $n = 40$ to 60, $t = 6$, 8 and 10, and show the minimum value of $\mu(t)$ for which there exist values of k and m that make $P_{IN}(t) \geq 0.99$. We see that $\mu(t)$ varies between 1 and 1,617, which is much less than $\binom{n}{t}$. (The curve for $t = 6$ is between 1 and 4, so it looks almost flat.) Note that here the corresponding values of $\binom{n}{t}$ is approximately between 10^7 and 10^{12}. It can be seen that the decoding algorithm reduces the number of candidate codewords substantially.

We have combined the word-recognition algorithm in [1] with the sliding-window decoding technique here, for suitably chosen n and t. The new algorithm improves the error-correction performance substantially. Consider compressed texts protected by an $(4376, 4095)$ LDPC code designed by MacKay [115], which has rate 0.936 and is designed for BSC of error probability 0.2% (a typical parameter setting in storage systems). We compare the new algorithm with two known algorithms: using the BP decoding of the LDPC code alone, and the word-recognition algorithm in [1]. The results are shown in Table 3.1, where *success rate* is defined as the probability that an LDPC codeword is decoded correctly.

BER	0.2%	0.3%	0.4%	0.5%	0.6%	0.7%
P_{ldpc}	100%	98.2%	77.5%	27.4%	2.9%	0
P_{soft}	100%	99.9%	99.5%	97.9%	94.2%	84.6%
P_{slid}	100%	100%	99.9%	99.2%	98.4%	94.5%
BER	0.8%	0.9%	1.0%	1.1%	1.2%	1.3%
P_{ldpc}	0	0	0	0	0	0
P_{soft}	67.1%	47.8%	26.7%	12.4%	3.9%	1.4%
P_{slid}	84.8%	68.3%	47.9%	28.7%	14.2%	5.8%

Table 3.1: The success rate of decoding with LDPC code alone (P_{ldpc}), the word-recognition algorithm (P_{soft}) [1], and the enhanced algorithm using sliding-window decoding (P_{slid}), when the bit error probability (BER of a binary-symmetric channel) increases from 0.2% to 1.3%.

3.3 Capacity of ECC with Natural Redundancy

In this section, we study two closely related theoretical models for ECCs with NR. We first study the capacity for information transmission when NR-decoding is present before ECC-decoding. We then study ECCs with finite length, and present an upper bound to the code sizes given that the ECCs receive assistance from NR-decoding.

3.3.1 Channel Capacity with Natural Redundancy

Consider compressed data with NR protected as information bits by a systematic ECC. A decoding scheme is shown in Fig. 3.3, where an NR-decoder is followed by an ECC-decoder. For the ECC, the channel together with the NR-decoder can be seen as a *compound channel*, where the channel adds noise and the NR-decoder reduces noise. The *compound-channel capacity* is defined in the conventional way, namely, as the maximum rate at which the channel input $X_1^n = (x_1, x_2, \cdots, x_n)$ (with $n \to \infty$) can be transmitted reliably.

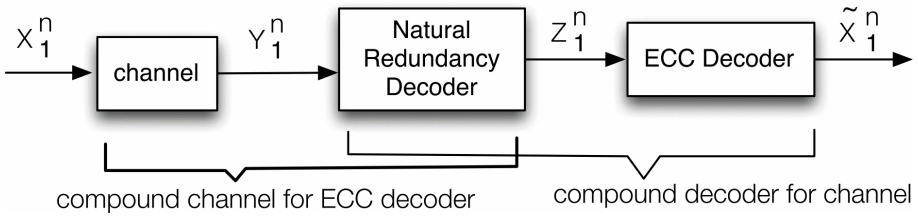

Figure 3.3: A decoding scheme that combines NR-decoding with ECC-decoding.

We have studied NR-decoding for both BEC and BSC in previous sections. Note that in principle, the NR-decoder can decode not only information bits, but also parity-check bits by using parity-check constraints. That motivates us to study the following two compound-channel models: (1) in the Compound-BEC model, the BEC erases each bit $x_i \in \{0, 1\}$ independently with probability p; then for each bit y_i, if it is an erasure, the NR-decoder marks it as "original erasure" (so that this marked information is known to the ECC decoder), then independently recovers its value

(correctly) as x_i with probability $(1-\delta)(1-\epsilon)$, recovers its value (incorrectly) as $1-x_i$ with probability $(1-\delta)\epsilon$, and let it remain as an erasure with probability δ; (2) in the Compound-BSC Model, the BSC flips each bit $x_i \in \{0,1\}$ independently with probability p; then for each bit y_i, the NR-decoder marks it as "NR-decoded bit" independently with probability r. If y_i is marked as an "NR-decoded bit", the NR-decoder independently sets its value (correctly) to x_i with probability $1-q$, and sets its value (incorrectly) to $1-x_i$ with probability $q < p$.

Theorem 5. *The capacity of Compound-BEC is* $C_{c-BEC} = 1-p+p(1-\delta)(1-H(\epsilon))$. *And the capacity of Compound-BSC is* $C_{c-BSC} = (1-r)(1-H(p)) + r(1-H(q))$.

3.3.2 Upper Bound to ECC Sizes with NR

The previous sections have presented analysis specifically for LDPC codes with belief-propagation decoding algorithms. Let us now consider general finite-length ECCs and their sizes. The NR-decoders for images and languages presented in Section II have a common feature: they both have very low error probabilities introduced by NR-decoding, namely, the corrections are made with high confidence by NR-decoders. That motivates us to study the following theoretical model for error correction.

Let $\mathcal{A} = \{0, 1, \cdots, q-1\}$ be an alphabet, where $q \geq 2$. Let $\mathcal{C} \subseteq \mathcal{A}^n$ be a code of length n. Let r and t be integer parameters with $r+t \leq n$. Let the decoding process be an NR-decoder followed by an ECC-decoder, as shown in Fig. 3.3. Given a noisy word $\mathbf{y} = (y_1, y_2, \cdots, y_n) \in \mathcal{A}^n$, assume that the NR-decoder can determine the correct values of at least r symbols with certainty, without introducing additional errors. (Note that in practice, the errors corrected by the NR-decoder are only a small portion of such bits (symbols with $q = 2$). Many more such bits are non-errors, and the NR-decoder can determine that they are error-free because they belong to highly likely patterns, such as long and common phrases. Also note that in general, the NR-decoder can decode both information bits and parity-check bits.) Let $P \subseteq \{1, 2, \cdots, n\}$ denote the indexes of such determined symbols (where $|P| \geq r$), and without loss of generality (WLOG), we may assume $|P| = r$ for code analysis (because having larger $|P|$ only helps more). WLOG, we may also

assume that the symbols of $\mathbf{y}$ with indexes in P are already correct symbols (because the NR-decoder determines their values anyway). After the NR-decoding, the ECC-decoder takes the pair $(\mathbf{y}, P)$ as input, and decodes it using maximum-likelihood (ML) decoding: the output is a codeword $\mathbf{x} = (x_1, x_2, \cdots, x_n) \in \mathcal{C}$ such that: (1) $\forall\ i \in P$, $x_i = y_i$; (2) the Hamming distance $d_H(\mathbf{x}, \mathbf{y}) \triangleq |\{i \mid 1 \leq i \leq n, x_i \neq y_i\}| = |\{i \mid 1 \leq i \leq n, i \notin P, x_i \neq y_i\}|$ is minimized.

$\forall\ \mathbf{x}, \mathbf{y} \in \mathcal{A}^n$ and $P \subseteq \{1, 2, \cdots, n\}$, if $x_i = y_i$ for every $i \in P$, we say $\mathbf{x} =_P \mathbf{y}$. We define $S_{t,P}(\mathbf{x}) \triangleq \{(\mathbf{y}, P) \mid \mathbf{x} =_P \mathbf{y},\ d_H(\mathbf{x}, \mathbf{y}) \leq t\}$. If $\forall\ \mathbf{x}_1, \mathbf{x}_2 \in \mathcal{C}$ and $P \subseteq \{1, 2, \cdots, n\}$ with $|P| = r$, we have $S_{t,P}(\mathbf{x}_1) \cap S_{t,P}(\mathbf{x}_2) = \emptyset$, we call $\mathcal{C}$ an (r, t)-ECC. An (r, t)-ECC is an error-correcting code that can correct t Hamming errors when the NR-decoder determines the values of any r symbols. It is an extension of t-error correcting codes. We have the following sphere packing bound.

Theorem 6. *For an (r, t)-ECC $\mathcal{C}$ with code length n, alphabet size q and $r + t \leq n$, the code's size*

$$|\mathcal{C}| \leq \frac{q^n}{\sum_{i=0}^{t} \binom{n-r}{i} (q-1)^i}.$$

3.4 Computational-Complexity Tradeoff

NR can be used for both compression and error correction. How to use it suitably depends on many factors, such as available coding techniques, hardware design, etc. In this work, we discuss one such tradeoff: the computational complexity of using NR for compression or error correction. Real NR is hard to model precisely, so we explore this topic from a theoretical point of view, and consider NR in general forms. We show that certain types of redundancy are computationally efficient for compression, while others are so for error correction. Note that there exist works on analyzing the hardness of certain types of source coding schemes [116, 117, 118] and channel coding schemes [119, 120, 121, 122, 123]. In contrast, here we focus on the tradeoff between the two.

Let $B = (b_1, b_2, \cdots, b_n) \in \{0, 1\}^n$ be an n-bit message with NR. Define $\mathcal{V} : \{0, 1\}^n \to \{0, 1\}$ as a *validity function*: B is a valid message if and only if $\mathcal{V}(B) = 1$. The set of all valid messages of n bits is $\mathcal{M} \triangleq \{B \in \{0, 1\}^n \mid \mathcal{V}(B) = 1\}$. For simplicity, for both source and channel coding,

assume that the valid messages in $\mathcal{M}$ are equally likely.

First, consider source coding. Let $k = \lceil \log_2 |\mathcal{M}| \rceil$. Define an *optimal lossless compression scheme* to be an injective function $C_{opt} : \mathcal{M} \to \{0,1\}^k$ that compresses any valid message $B \in \mathcal{M}$ to a distinct k-bit vector $C_{opt}(B)$. Define the *Data Compression Problem* as follows: Given a validity function $\mathcal{V}$, find an injective function $C_{opt} : \mathcal{M} \to \{0,1\}^k$.

Next, consider channel coding. Assume that a valid message $X = (x_1, x_2, \cdots, x_n) \in \mathcal{M}$ is transmitted through a binary-symmetric channel (BSC), and is received as a noisy message $Y = (y_1, y_2, \cdots, y_n) \in \{0,1\}^n$. Maximum likelihood (ML) decoding requires us to find a message $Z = (z_1, z_2, \cdots, z_n) \in \mathcal{M}$ that minimizes the Hamming distance $d_H(Y, Z)$. Define the *Error Correction Problem* as follows: Given a validity function $\mathcal{V}$ and a message $Y \in \{0,1\}^n$, find a valid message $Z \in \mathcal{M}$ that minimizes the Hamming distance $d_H(Y, Z)$.

Let $\mathcal{F}$ be the set of all functions from the domain $\{0,1\}^n$ to the codomain $\{0,1\}$. (We have $|\mathcal{F}| = 2^{2^n}$.) The function $\mathcal{V}$ represents NR in data. In practice, different types of data have different *types* of NR. Let us define the latter concept formally. For any subset $\mathcal{T} \subseteq \mathcal{F}$, let $\mathcal{T}$ be called a *type* of validity functions (which represents a type of NR). When $\mathcal{V}$ can only be a function in $\mathcal{T}$ (instead of $\mathcal{F}$), we denote the Data Compression Problem and the Error Correction Problem by $\mathcal{P}_{dc}^{\mathcal{T}}$ and $\mathcal{P}_{ec}^{\mathcal{T}}$, respectively. The hardness of the problems $\mathcal{P}_{dc}^{\mathcal{T}}$ and $\mathcal{P}_{ec}^{\mathcal{T}}$ depends on $\mathcal{T}$. Let $S_{dc=NP,ec=P}$ denote the set of types $\mathcal{T}$ (where each type is a subset of $\mathcal{F}$) for which the data compression problem $\mathcal{P}_{dc}^{\mathcal{T}}$ is NP-hard while the error correction problem $\mathcal{P}_{ec}^{\mathcal{T}}$ is polynomial-time solvable. Similarly, let $S_{dc=P,ec=NP}$ (or $S_{dc=P,ec=P}$, $S_{dc=NP,ec=NP}$, respectively) denote the set of types $\mathcal{T}$ for which $\mathcal{P}_{dc}^{\mathcal{T}}$ is polynomial-time solvable while $\mathcal{P}_{ec}^{\mathcal{T}}$ is NP-hard (or $\mathcal{P}_{dc}^{\mathcal{T}}$ and $\mathcal{P}_{ec}^{\mathcal{T}}$ are both polynomial-time solvable, or both NP-hard, respectively). The following theorem shows that there exist validity-function types for each of those four possible cases.

Theorem 7. *The four sets $S_{dc=NP,ec=P}$, $S_{dc=P,ec=NP}$, $S_{dc=P,ec=P}$ and $S_{dc=NP,ec=NP}$ are all non-empty.*

The above result shows a wide range of possibilities for the computational-complexity tradeoff between source and channel coding. In practice, it is worthwhile to study the properties of natural

redundancy (e.g., whether the redundancy is mainly local or global, which differs for different types of data), and choose appropriate coding schemes based on computational complexity along with other important factors.

4. COMBINATION OF LDPC AND MACHINE LEARNING-BASED NATURAL REDUNDANCY DECODING [1]

4.1 Introduction

Big-data storage is having increasingly wide applications. However, it faces a substantial challenge – how to recover data from errors as effectively as possible for reliable long-term storage – due to accumulative noise in storage media. For example, flash memories and other NVMs have noise mechanisms such as charge leakage, read/write disturbs, and cell-quality degradation due to P/E cycling. They make data more and more noisy over time. So there is a strong motivation in exploring new techniques for error correction.

In this work, we study how to correct errors using *natural redundancy* (NR) in compressed data, and how to combine it with error-correcting codes (ECCs). By natural redundancy, we refer to the redundancy in data that is not artificially added for error correction, such as features in languages/images and structures in databases. In comparison, the redundancy in an ECC (which we shall call *artificial redundancy*) is added in a disciplined way with the specific goal of effective error correction. NR is often a rich resource for error correction for data that are uncompressed or compressed imperfectly. There are various reasons for imperfect compression in practical systems, including high complexity of optimal compression, our limited understanding on the data models (e.g., for languages and images), etc. For data that are encoded as ECCs and later corrupted by errors, as our understanding on the data model improves, we can design better and better NR-decoders to correct the errors.

Example 8. *Consider texts compressed by an LZW algorithm that uses a fixed dictionary of size* 2^ℓ*. The dictionary has* 2^ℓ *text strings (called patterns) of variable lengths, where every pattern is encoded as an* ℓ*-bit codeword. Given a text to compress, the LZW algorithm scans it and partitions*

[1] ©IEEE 2017. Parts of this section are reprinted, with permission, from P. Upadhyaya and A. A. Jiang, "On LDPC decoding with natural redundancy," 55th Annual Allerton Conference on Communication, Control, and Computing (Allerton), Monticello, IL, 2017.

it into patterns, and maps them to codewords. For instance, if $\ell = 20$ and the text is "Flash memory is an electronic $\cdots$", the partitioning and LZW-codewords can be as illustrated in Fig. 4.1 (a).

Now suppose some bits in the LZW-codewords are erased. An NR-Decoder can check all the possible solutions, map each solution back to patterns, and use a dictionary of words to eliminate those solutions that contain invalid words. (Such a dictionary of words has been commonly used in spell checkers.) If all the remaining solutions agree on the value of an erased bit, then that erasure is decoded by the NR-Decoder. For instance, suppose each LZW-codeword in Fig. 4.1 (a) suffers from two erasures, which lead to four possible solutions/patterns (see Fig. 4.1 (b)). By combining the patterns for each codeword, we can rule out many solutions. For instance, the combination "should becnomially ars an ele" can be eliminated due to the invalid word "becnomially". In fact, the only combination without invalid words (without considering words on the boundary of the string, which might be part of a longer word) is "Flash memory is an ele", so the NR-Decoder can recover all six erasures in the three codewords. (In practice, it is also possible that we get more than one combination that contain only valid words. In that case, an erased bit can be corrected if all such combinations set the same value for that erasure.)

Suppose that the LZW-codewords, seen as information bits, are protected by a systematic ECC. Then the ECC-Decoder can correct erasures by parity-check constraints, and the NR-Decoder can correct erasures by NR. They can work collaboratively to maximize the number of correctable erasures. □

4.2 Efficient Natural Redundancy Discovery

4.2.1 Discovery of Natural Redundancy in Languages

As this work is largely motivated by language-based NR, it is worthwhile to note that an LZW algorithm with a dictionary of 2^{20} patterns (as in the above example) can compress the English language to 2.94 bits per character. The UNIX Compress command uses LZW with a smaller dictionary and so achieves a lower compression ratio. There are compression algorithms for languages with higher compression ratios (e.g., syllable-based Burrows-Wheeler Transform achiev-

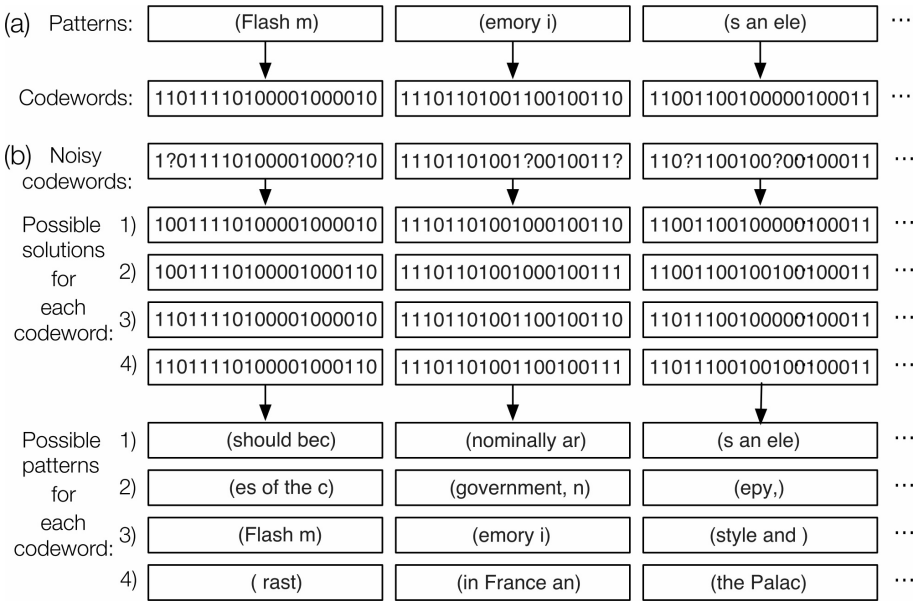

Figure 4.1: (a) Compress a text by LZW. (b) NR-decoding for erasures.

ing 2 bits/character [124]). However, there is still a gap toward Shannon's estimation of 1.34 bits/character for the entropy of English [8], which gives motivation for NR-Decoders. And one may reasonably conjecture that a similar scenario exists for images and videos.

In this work, we study the utilization of NR for erasure correction, including for languages and images. The paper is organized as follows. In Section 4.3, we survey related works. In Section 4.4, we introduce the discovery and utilization of NR in data for erasure correction, including for languages and images. In Section 4.5, we study a scheme that combines NR-decoding with low-density parity-check (LDPC) codes, and derive analytical formulas for the density evolution of LDPC decoding given information from the NR-decoder, which are useful for measuring the overall decoding performance. In Section 4.6, we propose a theoretical model for compressed languages, and study the performance of iterative decoding between the LDPC decoder and the NR-decoder.

4.3 Related Works

Error-correction with NR is related to joint source-channel coding and denoising. The idea of using the inherent redundancy in a source – or the leftover redundancy at the output of a source encoder – to enhance the performance of the ECC has been studied within the field of joint source-channel coding. In [97], source-controlled channel coding using a soft-output Viterbi algorithm is considered. In [94], a trellis based decoder is used as a source decoder in an iterative decoding scheme. Joint decoding of Huffman and Turbo codes is proposed in [96]. In [98], joint decoding of variable length codes (VLCs) and convolutional/Turbo codes is analyzed. Joint decoding using LDPC codes for VLCs and images are illustrated in [101] and [102], respectively. However, not many works have considered JSCC specifically for language-based sources, and exploiting the redundancy in the language structure via an efficient decoding algorithm remains as a significant challenge. Related to joint source-channel coding, denoising is also an interesting and well studied technique [107, 108, 109, 110, 111]. A denoiser can use the statistics and features of input data to reduce its noise level for further processing. For discrete memoryless channels with stationary input sequences, a universal algorithm that performs asymptotically as well as optimal denoisers are given in [125]. The algorithm is also universal for a semi-stochastic setting, where the channel input is an individual sequence and the randomness in the channel output is solely due to the channel's noise.

Spell-checking softwares are a typical example of using NR to correct errors in languages. They are widely used in text editors. A spell-checking software usually works at the character level (namely, it does not consider how characters or text strings are encoded by bits), is for uncompressed texts, and uses the validity of words and the correctness of grammar to correct errors that appear in the typing of texts.

Using NR to correct errors at the bit level in compressed texts has been studied in a number of works. In [112], texts compressed by Huffman coding is considered, and a dynamic programming algorithm is used to partition the noisy bit sequence into subsequences that represents words, and to select likely solutions based on the frequencies of words and phrases. In [1], texts that are

compressed by Huffman coding and then protected by LDPC codes are studied. An efficient greedy algorithm is used to decompress the noisy bit string, and partition it into stable and unstable regions based on whether each region contains recognizable words and phrases. The stable and unstable regions have polarized RBERs, which are provided as soft information to the LDPC code for better decoding performance. The algorithm is enhanced in [113] by a machine learning method for content recognition, and an iterative decoding algorithm between the NR-Decoder and the ECC-Decoder is used to further improve performance. In [114], texts compressed by Huffman coding and protected by Polar codes are studied. The validity of words is used to prune branches in a list sequential decoding algorithm, and a trie data structure for words is used to make the algorithm more efficient. A concatenated-code model that views the text with NR as the outer code and the Polar code as the inner code is considered, and the rate improvement for the Polar code due to NR is analyzed. That model is further studied in [126], where an optimal algorithm that maximizes the code rate improvement by unfreezing some frozen bits to store information is presented. A model that views NR as the output of a side information channel at the channel decoder is also studied, where NR is shown to improve the random error exponent.

4.4 NR-Decoding for Languages and Images

In this section, we present techniques for using NR in compressed data, including languages and images, for correcting erasures.

4.4.1 NR-Decoding for Language

Consider English texts that are compressed by an LZW algorithm that uses a fixed dictionary of size 2^{ℓ}. We have introduced a technique that corrects bit erasures based on the validity of words in Example 8. For long compressed texts with erasures, to make the NR-decoding efficient, we use a decoding algorithm based on sliding-windows of variable lengths as follows. Let n_{min} and n_{max} be two integers, where $n_{min} < n_{max}$. We first use a sliding-window of $n_{min}\ell$ bits to scan the compressed text (where every such window contains exactly n_{min} LZW-codewords), and obtain candidate solutions for each window based on the validity of words (as in Example 8). We then

increase the size of the window to $(n_{min}+1)\ell$, $(n_{min}+2)\ell$, $\cdots$, $n_{max}\ell$, and do decoding for each size in the following way: consider a window of $k\ell$ bits that contains k LZW-codewords C_1, C_2, $\cdots$, C_k. Let $S_1 \subseteq \{0,1\}^{(k-1)\ell}$ be the set of candidate solutions for the sub-window that contains the LZW-codewords C_1, C_2, $\cdots$, C_{k-1}; and let $S_2 \subseteq \{0,1\}^{(k-1)\ell}$ be the set of candidate solutions for the sub-window that contains the LZW-codewords C_2, C_3, $\cdots$, C_k. (Both S_1 and S_2 have been obtained in the previous round of decoding.) We now obtain the set of candidate solutions for the current window, which contains C_1, C_2, $\cdots$, C_k, this way. A bit sequence $(b_1, b_2, \cdots, b_{k\ell})$ is in S only if it satisfies two conditions: (1) its first $(k-1)\ell$ bits are a solution in S_1, and its last $(k-1)\ell$ bits are a solution in S_2; (2) the decompressed text corresponding to it contains no invalid words (except on the boundaries). This way, potential solutions filtered by smaller windows will not enter solutions for larger windows, making decoding more efficient. As a final step, an erased bit is decoded this way: if *any* of the windows of size $n_{max}\ell$ containing it (note that there are up to $2n_{max}-1$ such windows) can recover its value (as we did in Example 8), decode it to that value; otherwise it remains as an erasure.

To make the above decoding algorithm more efficient, we also use phrases (such as "information theory", "flash memory") and features such as word/phrase lengths. If a solution for a window contains a valid word or phrase that is particularly long, we may remove other candidate solutions that contain only short words. That is because long words and phrases are very rare: their density among bit sequences of the same length decreases exponentially fast as the length increases. So if they appear, the chance that they are the correct solution is high based on Bayes' rule. The thresholds for such word/phrase lengths can be set sufficiently high such that the probability of making a decoding error is sufficiently small.

We also enhance the decoding performance by using the *co-location* relationship. Co-location means that certain pairs of words/phrases appear unusually frequently in the same context (because they are closely associated), such as "dog" and "bark", or "information theory" and "channel capacity". If two words/phrases with the co-location relationship are detected among candidate solutions for two windows close to each other, we may keep them as candidate solutions and remove

other less likely solutions. The reason for this approach is similar to that for long words/phrases. The co-location relationship can appear in multiple places in a text, and therefore help decoding in non-trivial ways. For example, for the text in Fig. 4.2 (a), the words/phrases that have the co-location relationship with the phrase "flash memory" are shown in Fig. 4.2 (b). (All of them appear in this text.) How to find words/phrases with the co-location relationship from a corpus of training texts is a well-known technique in Natural Language Processing (NLP) [127]. So we skip its details here.

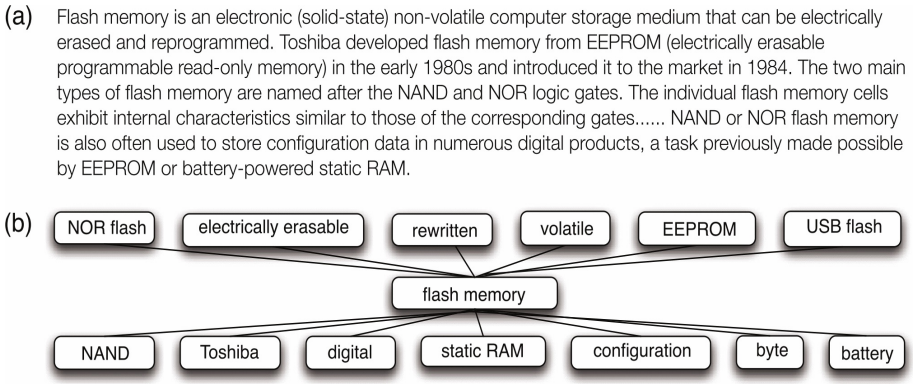

Figure 4.2: (a) A sample paragraph from Wikipedia (part of which was omitted to save space). (b) Phrases in it that have the co-location relationship with "flash memory".

4.4.2 NR-Decoding for Images

Consider the discovery of NR for images. General images can have global features, and using such redundancy for error correction can be difficult. To gain more insight into the nature of NR in images, we focus in particular on images of handwritten digits, as in Fig. 4.3 (a). They are from the National Institute of Standards and Technology (NIST) database, which have $70,000$ images as training or test data. We compress the bi-level images (of size 28×28 pixels) using run-length coding, where the run-lengths of 0s and 1s are compressed by two optimized Huffman codes, respectively. The rate is 0.27 bit/pixel.

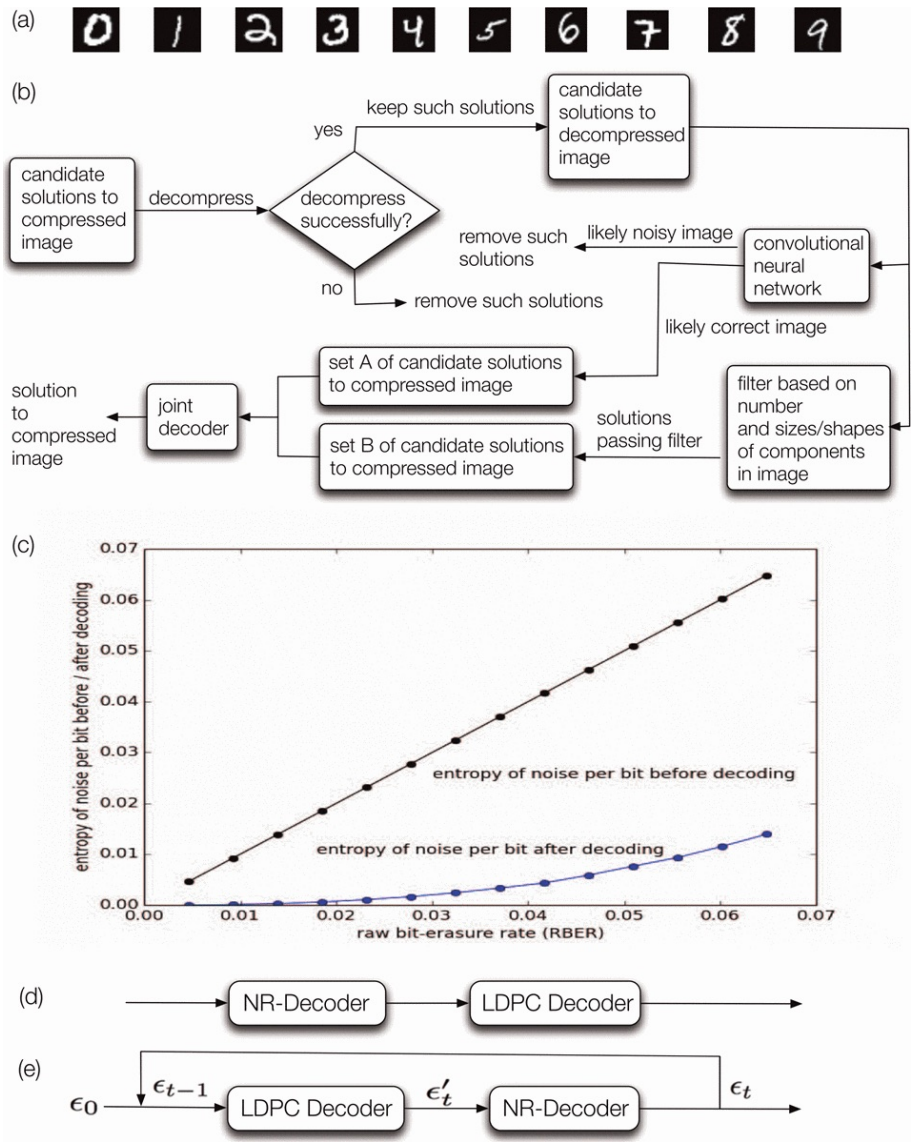

Figure 4.3: (a) Examples of handwritten digits. (b) NR-decoder for images. (c) Performance of NR-decoder. (d) A concatenated decoding scheme. (e) An iterative decoding scheme.

We now present an NR-decoder for images. It is illustrated in Fig. 4.3 (b). Assume that a compressed image has λ erasures. Out of the 2^λ possible candidate solutions, usually only a few decompress successfully. (For example, to decompress successfully, the bit sequence needs to end with a valid Huffman codeword. And errors may make it impossible.) To decode noisy images among the successfully decompressed images, we have trained a convolutional neural network for recognizing noisy images, and designed a specialized filter based on features of connected components in decompressed images, as follows:

4.4.2.1 Convolutional Neural Network

The training and test data consist of noisy as well as clean images of handwritten digits. It consists of one input layer, two hidden layers and a output layer. The input layer consists of a 28×28 bilevel image, and the 2×1 output layer classifies the input images as “clean" or “noisy". The size of the convolution window is 5×5. The number of feature maps used in the first and second hidden layers are 5 and 15 respectively.

4.4.2.2 Filter Based on Connected Components

We count the number of components in an image, but without counting those components that have at most two pixels or components that are vertical lines (which may be caused by human or scanning errors). The images that have the fewest components are accepted as candidate images by this filter.

4.4.2.3 Joint Decoder

The final step of decoding is: if all candidate solutions agree on the value an erased bit, set the bit to that value; otherwise, keep it as an erasure.

Example9. *Suppose that the compressed image with erasures is $1??0?1\cdots$, where “?” is an erasure. Suppose that the NR-decoder finds 3 candidate solutions: $110001\cdots$, $110011\cdots$, $100011\cdots$. Then it returns the solution $1?00?1\cdots$ because the candidate solutions agree on the second erasure, but not the first or the third erasure.* □

4.4.3 Decoding Performance of NR Decoders

The decoding performance for NR decoders can be measured as follows. Let $\epsilon \in [0, 1]$ be the erasure probability before decoding. After the decoding by natural redundancy, let $\delta \in [0, 1]$ be the probability that an originally erased bit remains as an erasure, and let $\rho \in [0, 1 - \delta]$ be the probability that an originally erased bit is decoded to 0 or 1 incorrectly. The amount of noise after NR-decoding can be measured by the entropy of the noise (erasures and errors) per bit: $E_{NR} \triangleq \epsilon(\delta + (1 - \delta)H(\frac{\rho}{1-\delta}))$, where $H(p) = -p \log p - (1 - p) \log(1 - p)$ is the entropy function.

We show E_{NR} for the NR-decoder for images in Fig. 4.3 (c). The NR-decoder reduces noise substantially: it removes noise effectively by over 75% for the compressed images (without any help from ECC), for raw bit-erasure rate (RBER) from 0.5% to 6.5%.

The performance of the NR-decoder introduced above for LZW-compressed English texts, experimented on a large corpus of Wikipedia articles, is shown in Table 4.1. It also reduces noise effectively (between 88.0% and 91.6%) for raw bit-erasure rate from 5% to 30%.

ϵ	0.05	0.10	0.15
δ	8.22×10^{-2}	8.67×10^{-2}	9.19×10^{-2}
ρ	9.18×10^{-5}	1.83×10^{-4}	1.82×10^{-4}
E_{NR}	4.18×10^{-3}	8.92×10^{-3}	1.42×10^{-2}
Noise reduction	91.6%	91.1%	90.6%
ϵ	0.20	0.25	0.30
δ	9.76×10^{-2}	1.05×10^{-1}	1.12×10^{-1}
ρ	3.61×10^{-4}	4.48×10^{-4}	7.11×10^{-4}
E_{NR}	2.04×10^{-2}	2.76×10^{-2}	3.60×10^{-2}
Noise reduction	89.8%	89.0%	88.0%

Table 4.1: Performance of the NR-decoder introduced above for LZW-compressed English text.

4.5 Combine NR-decoding with LDPC Codes

This section discusses the combination of NR-decoders described in the previous section with LDPC codes. We protect compressed data (languages or images) as information bits by a systematic LDPC code of rate R. The decoding process is a concatenation of two decoders: first, the NR-decoder decodes the codeword (possibly only its information bits), and outputs a partially corrected codeword with updated soft information; then, the LDPC decoder takes that as input, and uses belief propagation (BP) for decoding. (See Fig. 4.3 (d) for an illustration.) We present a theoretical analysis for the decoding performance, and show that the NR-decoder can substantially improve the performance of LDPC codes.

Consider a binary-erasure channel (BEC) with erasure probability ϵ_0. Let us call the non-erased bits *fixed bits*. Assume that after NR-decoding, a non-fixed bit (i.e., erasure) remains as an erasure with probability $p_0(\epsilon_0) \in [0,1]$, becomes an error (0 or 1) with probability $(1-p_0(\epsilon_0))\gamma_0(\epsilon_0) \in [0, 1-p_0(\epsilon_0)]$, and is decoded correctly (as 0 or 1) with probability $(1-p_0(\epsilon_0))(1-\gamma_0(\epsilon_0))$. (In general, $p_0(\epsilon_0)$ and $\gamma_0(\epsilon_0)$ may be functions of ϵ_0. Note that if the NR-decoder decodes only information bits, and an erasure in the information bits remains as an erasure with probability $p_0(\epsilon_0)'$, then $p_0(\epsilon_0) = Rp_0(\epsilon_0)' + (1-R)$. Also note that the LDPC decoder needs to decode all bits with both errors and erasures.)

4.5.1 Decoding Algorithm

We design the following iterative LDPC decoding algorithm, which generalizes both the peeling decoder for BEC and the Gallager B decoder for BSC:

Algorithm 10. *Generalized LDPC decoding algorithm.*

(1) Let $\pi \in [1, d_v - 1]$ and $\tau \in [1, d_v - 1]$ be two integer parameters;

(2) In each iteration, for a variable node v that is an erasure, if π or more non-erased message bits come from $d_v - 1$ check nodes and they all have the same value, set v to that bit value;

(3) If v is not a fixed bit and not an erasure (but possibly an error) in this iteration, change v to the opposite bit value if τ or more non-erased message bits come from $d_v - 1$ check nodes and

they all have that opposite value. (The updated value of v will be sent to the remaining check node in the next iteration.)

4.5.2 Density Evolution Analysis

We now analyze the density evolution for the decoding algorithm, for an infinitely long and randomly constructed LDPC code of regular degrees.

For $t = 0, 1, 2 \cdots$, let α_t and β_t be the fraction of codeword bits that are errors or erasures, respectively, after t iterations of LDPC decoding. We have $\alpha_0 = \epsilon_0(1 - p_0(\epsilon_0))\gamma_0(\epsilon_0)$ and $\beta_0 = \epsilon_0 p_0(\epsilon_0)$. Let $\kappa_0 = \epsilon_0(1 - p_0(\epsilon_0))(1 - \gamma_0(\epsilon_0))$.

Theorem 11. *For a regular (d_v, d_c) LDPC code with variable-node degree d_v and check-node degree d_c, we have $\alpha_{t+1} = \alpha_0 C_t + \kappa_0 D_t + \beta_0 \mu_t$, where $C_t = 1 - (1 - A_t)^{d_v - 1} + \sum_{i=0}^{\tau-1} \binom{d_v-1}{i} B_t^i (1 - A_t - B_t)^{d_v-i-1}$, $D_t = \sum_{j=\tau}^{d_v-1} \binom{d_v-1}{j} A_t^j (1 - A_t - B_t)^{d_v-1-j}$, $\mu_t = \sum_{m=\pi}^{d_v-1} \binom{d_v-1}{m} A_t^m (1 - A_t - B_t)^{d_v-1-m}$ with $A_t = \frac{(1-\beta_t)^{d_c-1} - (1-\beta_t-2\alpha_t)^{d_c-1}}{2}$ and $B_t = \frac{(1-\beta_t)^{d_c-1} + (1-\beta_t-2\alpha_t)^{d_c-1}}{2}$. And $\beta_{t+1} = \beta_0(1 - \mu_t - \nu_t)$, where $\nu_t = \sum_{m=\pi}^{d_v-1} \binom{d_v-1}{m} B_t^m (1 - A_t - B_t)^{d_v-1-m}$.*

Proof. Consider the root variable node of a computation tree. After t iterations, let A_t denote the probability that an incoming message to the root node from a neighboring check node is an error, and let B_t denote the probability that the message is correct. Then $1 - A_t - B_t$ is the probability that the message is an erasure. Let μ_t (respectively, ν_t) be the probability that among the $d_v - 1$ incoming messages from neighboring check nodes to the root node, π or more messages are errors (respectively, correct) and the remaining messages are all erasures.

In the $(t + 1)$-th iteration, we can have an error in the root node in one of the following cases:

1. The root node was initially (namely, before decoding begins) an error (which has probability α_0), and either of the two disjoint events happens: 1) fewer than τ check-node messages are correct and the remaining messages are all erasures, which happens with probability $\sum_{i=0}^{\tau-1} \binom{d_v-1}{i} B_t^i (1 - A_t - B_t)^{d_v-i-1}$; 2) at least one check-node message is an error, which happens with probability $1 - (1 - A_t)^{d_v-1}$. The probability that either of the two events occurs is $C_t = 1 - (1 - A_t)^{d_v-1} + \sum_{i=0}^{\tau-1} \binom{d_v-1}{i} B_t^i (1 - A_t - B_t)^{d_v-i-1}$.

2. The root node was initially correct (which has probability κ_0), but τ or more check-node messages are errors and the rest are all erasures (which happens with probability $D_t = \sum_{j=\tau}^{d_v-1} \binom{d_v-1}{j} A_t^j (1 - A_t - B_t)^{d_v-1-j}$).

3. The root node was initially an erasure (which has probability β_0), and π or more check-node messages are errors and the rest are all erasures (which happens with probability μ_t).

Therefore the error rate after $t+1$ iterations will be $\alpha_{t+1} = \alpha_0 C_t + \kappa_0 D_t + \beta_0 \mu_t$.

In the $(t+1)$-th iteration, we can correct an erasure at a root node correctly if the root node was initially an erasure, and π or more check-node messages are correct and the rest are all erasures. This happens with probability $\beta_0 \nu_t$. The root node will remain as an erasure if it is neither corrected mistakenly nor corrected correctly. So the erasure rate after $t+1$ iterations will be $\beta_{t+1} = \beta_0(1 - \mu_t - \nu_t)$.

Now we need to find the values of A_t, B_t, μ_t and ν_t. The incoming message from a check node to the root node is correct if out of the $d_c - 1$ non-root variable nodes connected to the check node, an even number of nodes are errors and the rest are all correct (i.e., neither errors nor erasures). That probability is $B_t = \sum_{k=0}^{\lfloor \frac{d_c-1}{2} \rfloor} \binom{d_c-1}{2k} \alpha_t^{2k} (1 - \alpha_t - \beta_t)^{d_c-1-2k} = \frac{(1-\beta_t)^{d_c-1} + (1-\beta_t-2\alpha_t)^{d_c-1}}{2}$. The incoming message from a check node to the root node is an error if out of the $d_c - 1$ non-root variable nodes connected to the check node, an odd number of nodes are errors and the rest are all correct. That probability is $A_t = \sum_{k=1}^{\lfloor \frac{d_c}{2} \rfloor} \binom{d_c-1}{2k-1} \alpha_t^{2k-1} (1 - \alpha_t - \beta_t)^{d_c-2k} = \frac{(1-\beta_t)^{d_c-1} - (1-\beta_t-2\alpha_t)^{d_c-1}}{2}$. The probability that π or more neighboring check-node messages are errors and the rest are all erasures can be simplified as $\mu_t = \sum_{m=\pi}^{d_v-1} \binom{d_v-1}{m} A_t^m (1 - A_t - B_t)^{d_v-1-m}$. The probability that π or more neighboring check-node messages are correct and the rest are all erasures can be simplified as $\nu_t = \sum_{m=\pi}^{d_v-1} \binom{d_v-1}{m} B_t^m (1 - A_t - B_t)^{d_v-1-m}$. This completes the proof. □

4.5.3 Erasure Threshold

Define *erasure threshold* ϵ^* as the maximum erasure probability (for ϵ_0) for which the LDPC code can decode successfully (which means the error/erasure probabilities α_t and β_t both approach 0 as $t \to \infty$). Let us show how the NR decoder can substantially improve ϵ^*. Consider a regular

LDPC code with $d_v = 5$ and $d_c = 100$, which has rate 0.95 (a typical code rate for storage systems). Without NR-decoding, the erasure threshold is $\tilde{\epsilon}^* = 0.036$. Now let $\pi = 1$ and $\tau = 4$. For compressed images, when $\epsilon_0 = 0.065$, the NR-decoder gives $p_0 = 0.247$ and $\gamma_0 = 0.0008$, for which the LDPC decoder has $\lim_{t\to\infty} \alpha_t = 0$ and $\lim_{t\to\infty} \beta_t = 0$. (The same happens for $\epsilon_0 < 0.065$.) So with NR-decoding, $\epsilon^* \geq 0.065$, which means the improvement in erasure threshold is more than 80.5%.

For LZW-compressed texts, when $\epsilon_0 = 0.3$, the NR-decoder gives $p_0 = 0.156$ and $\gamma_0 = 0.0008$, for which the LDPC decoder has $\lim_{t\to\infty} \alpha_t = 0$ and $\lim_{t\to\infty} \beta_t = 0$. (The same happens for $\epsilon_0 < 0.3$.) So with NR-decoding, $\epsilon^* \geq 0.3$, which means the improvement in erasure threshold is more than 733.3%.

4.6 Iterative LDPC Decoding with NR

In this section, we study the decoding performance when we use *iterative decoding* between the LDPC decoder and NR-decoder, as shown in Fig. 4.3 (e). (In last section's study, the NR-decoder is followed by the LDPC decoder, without iterations between them.) We focus on languages, and present a theoretical model for compressed languages as follows.

4.6.1 NR Decoder For Compressed Languages

Let $T = (b_0, b_1, b_2, \cdots)$ be a compressed text. Partition T into segments $S_0, S_1, S_2 \cdots$, where each segment $S_i = (b_{il}, b_{il+1}, \cdots, b_{il+l-1})$ has l bits. Consider erasures. Let $\theta \in [0, 1]$, $l_\theta \triangleq \lfloor l\theta \rfloor$ and $p \in [0, 1]$ be parameters. We assume that when a segment S_i has at most l_θ erasures, the NR-decoder can decode it by checking the validity of the up to 2^{l_θ} candidate solutions (based on the validity of their corresponding words/phrases, grammar, etc.), and either determines (independently) the correct solution with probability p or makes no decision with probability $1 - p$. And this NR-decoding operation can be performed only *once* for each segment.

Here l_θ models the limit on time complexity (because the decoder needs to check 2^{l_θ} solutions), and p models the probability of making an error-free decision. This is a simplification of the practical NR-decoders shown in the last section that make very high-confidence, although not

totally error-free, decisions. The model is suitable for compression algorithms such as LZW coding with a fixed dictionary, Huffman coding, etc., where each segment can be decompressed to a piece of text. The greater l is, the better the model is.

4.6.2 Iteration with LDPC Decoder

The compressed text T is protected as information bits by a systematic LDPC code. The LDPC code uses the peeling decoder for BEC (where $d_c - 1$ incoming messages of known values at a check node determine the value of the outgoing message on the remaining edge) to correct erasures. See the decoding model in Fig. 4.3 (e). In each iteration, the LDPC decoder runs *one iteration* of BP decoding, then the NR-decoder tries to correct those l-information-bit segments that contain at most l_θ erasures (if those segments were never decoded by the NR-decoder in any of the previous iterations). Let $\epsilon_0 < 1$ be the BEC's erasure rate. Let ϵ'_t and ϵ_t be the LDPC codeword's erasure rate after the t-th iteration of the LDPC decoder and the NR-decoder, respectively. Next, we analyze the density evolution for regular (d_v, d_c) LDPC codes of rate $R = 1 - \frac{d_v}{d_c}$.

Note that since the NR-decoder decodes only information bits, for the LDPC decoder, the information bits and parity-check bits will have different erasure rates during decoding. Furthermore, information bits consist of l-bit segments, while parity-check bits do not. For such an l-bit segment, if the NR-decoder can decode it successfully when it has no more than l_θ erasures, let us call the segment *lucky*; otherwise, call it *unlucky*. Lucky and unlucky segments will have different erasure rates during decoding, too.

Every l-information-bit segment is *lucky* with probability p, and *unlucky* with probability $1-p$. A lucky segment is guaranteed to be decoded successfully by the NR-decoder once the number of erasures in it becomes less than or equal to l_θ; and an unlucky segment can be considered as *never* to be decoded by the NR-decoder (because such decoding will not succeed). Since whether a segment is lucky or not is independent of the party-check constraints and the LDPC-decoder, for analysis we can consider it as an inherent property of the segment (which exists even before the decoding begins).

4.6.3 Density Evolution Analysis

Define $q_0 = 1$, $q_t \triangleq \frac{\epsilon_t}{\epsilon'_t}$ and $d_t \triangleq \frac{\epsilon'_t}{\epsilon_{t-1}}$ for $t \geq 1$. Note that decoding will end after t iterations if one of these conditions occurs: (1) $\epsilon'_t = 0$, because all erasures are corrected by the t-th iteration; (2) $d_t = 1$, because the LDPC decoder corrects no erasure in the t-th iteration, and nor will the NR-decoder since the input codeword is identical to its previous output. We now study density evolution before those boundary cases occur.

For $t = 1, 2, 3 \cdots$ and $k = 0, 1, \cdots, l$, let $f_k(t)$ denote the probability that a lucky segment contains k erasures after t iterations of decoding by the NR-decoder.

Lemma 12.

$$f_k(1) = \begin{cases} \sum_{i=0}^{l_\theta} \binom{l}{i} (\epsilon'_1)^i (1-\epsilon'_1)^{l-i} & \text{if } k = 0 \\ 0 & \text{if } 1 \leq k \leq l_\theta \\ \binom{l}{k} (\epsilon'_1)^k (1-\epsilon'_1)^{l-k} & \text{if } l_\theta + 1 \leq k \leq l \end{cases}$$

Proof. Consider the LDPC-decoding and the NR-decoding in the first iteration. Since the initial erasure rate is ϵ_0, the erasure rate after LDPC decoding will now be $\epsilon'_1 = q_0 \epsilon_0 (1 - (1-\epsilon_0)^{d_c - 1})^{d_v - 1}$ where $q_0 = 1$ by definition. The probability that an l-information-bit segment contains exactly i erasures is given by $\binom{l}{i} (\epsilon'_1)^i (1-\epsilon'_1)^{l-i}$, which is independent of whether the segment is lucky or unlucky. Thus the probability that a *lucky* segment contains up to l_θ erasures is given by $\sum_{i=0}^{l_\theta} \binom{l}{i} (\epsilon'_1)^i (1-\epsilon'_1)^{l-i}$. All such segments are decoded by the NR-decoder successfully, while the remaining segments are not. That leads to the conclusion. □

Lemma 13. *The erasure rate after the first iteration of NR-decoding is*

$$\epsilon_1 = \epsilon_0 d_1 ((1-R) + R(1-p)) + \Big(\sum_{k=l_\theta+1}^{l} \frac{k}{l} f_k(1) \Big) Rp$$

Proof. After NR-decoding, the erasure rate of a lucky segment with k erasures is $\frac{k}{l}$, and the erasure rate for unlucky segments and parity-check bits is still ϵ'_1. We have $d_1 = \epsilon'_1 / \epsilon_0$. Hence the overall erasure rate after the 1st iteration of NR-decoding is $\epsilon_1 = \epsilon_0 d_1 ((1-R) + R(1-p)) +$

$(\sum_{k=l_\theta+1}^{l} \frac{k}{l} f_k(1))Rp$. (See Fig. 4.4 (b) for an illustration of the computation tree for density evolution. For comparison, we show the tree for classic BP decoding for BEC in Fig. 4.4 (a).) □

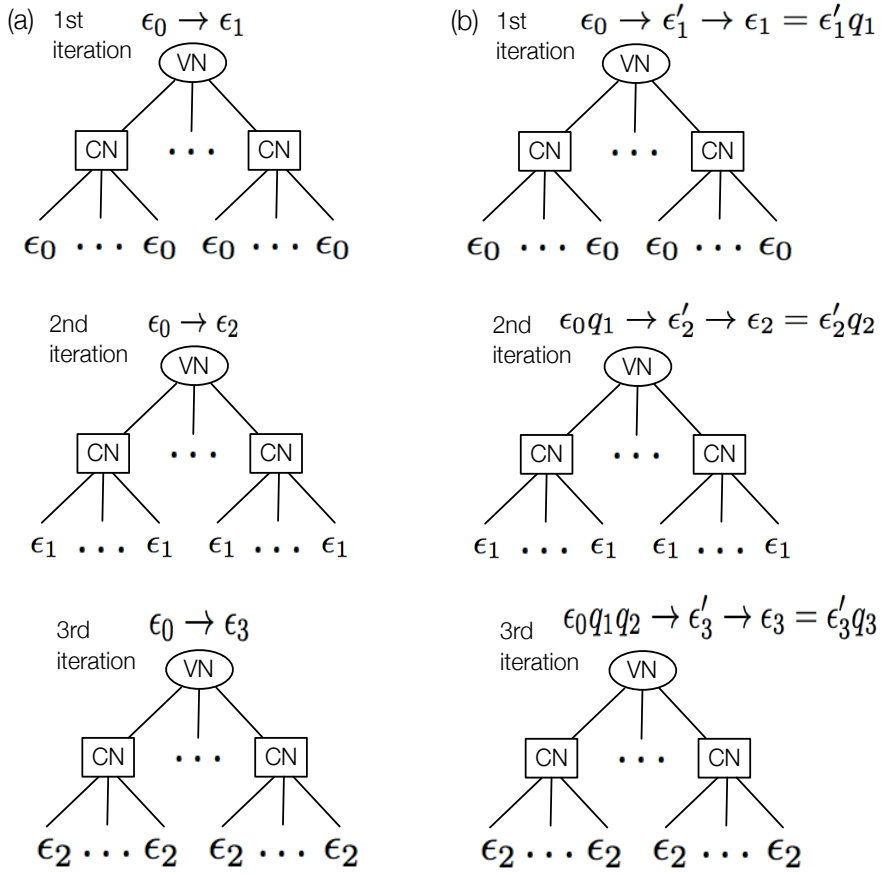

Figure 4.4: (a) First three iterations of classic BP decoding (alone) for BEC. (b) First three iterations of BP-decoding and NR decoding.

Lemma 14. *The erasure rate after the second iteration of LDPC-decoding is*

$$\epsilon_2' = q_0 q_1 \epsilon_0 (1 - (1 - \epsilon_1)^{d_c - 1})^{d_v - 1}$$

.

Proof. We have $q_1 = \frac{\epsilon_1}{\epsilon_1'}$. Since the NR-decoding of the 1st iteration reduces the *overall* erasure probability by a factor of q_1 (from ϵ_1' to ϵ_1), and the root variable node of a computation tree is chosen uniformly at random from the infinitely long and randomly constructed LDPC code, the root node in the tree for the 2nd iteration of LDPC decoding now has the erasure probability $q_1\epsilon_0$. (See Fig. 4.4 (b).) Hence the equation for the LDPC-decoder for the 2nd iteration will be given by $\epsilon_2' = q_0 q_1 \epsilon_0 (1-(1-\epsilon_1)^{d_c-1})^{d_v-1}$. Note that LDPC decoding is independent of NR-decoding because the parity-check constraints are independent of the bits being lucky-segment bits, unlucky-segment bits or parity-check bits. And note that $d_2 = \frac{\epsilon_2'}{\epsilon_1}$ is the probability that an erasure *remains as an erasure* after the LDPC decoding. If $d_2 = 1$, no change was made by the LDPC-decoder; if $d_2 = 0$, all erasures have been corrected. In both cases, the decoding will end. □

Lemma 15. *For $t \geq 2$,*

$$f_k(t) = \begin{cases} f_k(t-1) + \sum_{i=l_\theta+1}^{l} \sum_{j=0}^{l_\theta} f_i(t-1)\binom{i}{j}(d_t)^j(1-d_t)^{i-j} \\ \qquad\qquad \textit{if } k = 0 \\ 0 \qquad\qquad \textit{if } 1 \leq k \leq l_\theta \\ \sum_{i=k}^{l} f_i(t-1)\binom{i}{k}(d_t)^k(1-d_t)^{i-k} \quad \textit{if } l_\theta + 1 \leq k \leq l \end{cases}$$

Proof. Now consider the second iteration of NR-decoding. We only consider the case when $0 < d_2 < 1$. A lucky segment has zero errors after the second iteration if an only if either one of the two cases happen : a) that the segment already has zero errors after the first iteration, or b) the segment had $l_\theta + 1$ or more errors after the first iteration and it has at most l_θ erasures after second iteration of the LDPC-decoding. Thus if $k = 0$,

$$f_k(2) = f_k(1) + \sum_{i=l_\theta+1}^{l} \sum_{j=0}^{l_\theta} f_i(1)\binom{i}{j}(d_2)^j(1-d_2)^{i-j}$$

A lucky segment cannot have $k \leq l_\theta$ erasures (with $k \geq 1$) after the second iteration of NR-

decoding (because if so, it would have corrected those erasures). So we have $f_k(2) = 0$ for that case. Finally, a lucky segment has $l_\theta + 1 \leq k \leq l$ erasures if and only if it had k or more erasures after the first iteration of NR-decoding and it has k erasures after the second iteration of LDPC-decoding. Thus

$$f_k(2) = \sum_{i=k}^{l} f_i(1) \binom{i}{k} (d_2)^k (1-d_2)^{i-k} \text{ if } l_\theta + 1 \leq k \leq l$$

The remaining cases can be analyzed similarly. That leads to the conclusion. □

We now present the analytical formulas for the density evolution of the iterative LDPC-NR decoding scheme. Its proof follows the previous lemmas.

Theorem 16. *For $t \geq 1$,*

$$\epsilon_t = ((1-R) + R(1-p))\epsilon_0 (\prod_{i=1}^{t} d_t) + Rp \sum_{k=l_\theta+1}^{l} \frac{k}{l} f_k(t),$$

$$\epsilon'_t = (\prod_{m=0}^{t-1} q_m)\epsilon_0 (1 - (1-\epsilon_{t-1})^{d_c-1})^{d_v-1}.$$

Proof. The decoding performance for the 2nd iteration of the LDPC-decoding has been analyzed in Lemma 14. The erasure rate in unlucky-segment bits and parity-check bits was decreased from ϵ'_1 to $\epsilon'_1 d_2 = \epsilon_0 d_1 d_2$ by the LDPC-decoding. Now the NR-decoder corrects those lucky segments that had more than l_θ erasures before the LDPC-decoding but now has at most l_θ erasures after the LDPC-decoding. So $\epsilon_2 = \epsilon_0 d_1 d_2((1-R) + R(1-p)) + (\sum_{k=l_\theta+1}^{l} \frac{k}{l} f_k(2))Rp$.

The analysis for the following iterations is similar to the 2nd iteration. In general, since in the i-th iteration the NR-decoder reduces the overall erasure rate by a factor of q_i, the root variable node in the computation tree for the t-th iteration of LDPC decoding has the erasure probability $(\prod_{i=0}^{t-1} q_i)\epsilon_0$. That leads to the conclusion. □

5. STOPPING SET ELIMINATION OF LDPC CODES BY LANGUAGE-BASED NATURAL REDUNDANCY [1]

5.1 Introduction

The amount of data stored in the Internet is growing exponentially fast. With this growth, how to ensure long-term data reliability for all data also becomes more challenging. To assist error-correcting codes (ECC), the redundancy in the content of data itself can be utilized. This type of redundancy – such as features in languages, images and videos, structures in HTML files and databases, etc. – is referred to as *natural redundancy (NR)*, which supplements the more structured redundancy added by error-correcting codes [1, 28]. NR exists in both uncompressed and imperfectly compressed data, which are abundant in storage systems. That makes NR a promising tool to enhance data reliability.

In this work, we propose a relatively generic decoding model for collaborative ECC-Decoding and NR-Decoding that is motivated by language-based NR. The model is shown in Fig. 5.1. The (compressed or uncompressed) data, seen as information bits, are encoded into a systematic ECC codeword. The NR-decoder uses a sliding window of L bits to check a segment of the data each time, and uses its NR to correct errors/erasures in it. We bound the size of the window to L bits because due to the lack of structures in NR, NR-decoding is often not as efficient as ECC-decoding and its complexity grows with L, so a finite L bounds the acceptable complexity of NR-decoding. The NR-Decoder works jointly with the ECC-Decoder to correct errors/erasures.

The above model can be applied to languages compressed by LZW codes or Huffman codes, where some practical decoding algorithms have been presented [1, 28, 112, 113, 114, 126]. In this work, we study a basic theoretical problem for LDPC codes: when the number of erasures in a noisy LDPC codeword exceeds the decoding capability of the LDPC code's ECC-Decoder, what is the minimum number of erasures that an NR-Decoder needs to help correct so that the remaining

[1]

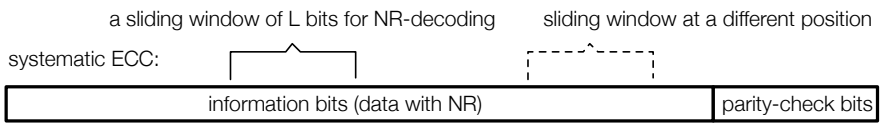

Figure 5.1: A model for collaborative ECC-decoding and NR-decoding.

erasures are decodable by the ECC-Decoder?

Let us define the problem more specifically. Let the LDPC code's ECC-Decoder be the following widely-used iterative belief-propagation (BP) algorithm: in each iteration, use every parity-check equation involving exactly one erasure to decode that erasure; and repeat until every equation involves zero or at least two erasures. If the ECC-Decoding fails, then we are left with a *stopping set*, which is a set of erasures such that every parity-check equation involving any of them involves at least two of them. If we represent the LDPC code by a bipartite Tanner graph, then a stopping set is a subset of variable nodes (representing erasures) such that a check node adjacent to any of them is adjacent to at least two of them.

We illustrate the average sizes of Stopping Sets for different raw bit-erasure rates (RBERs) in Fig. 5.2. It is for an (8192,7561) LDPC code of rate 0.923 and regular degrees $(d_v = 3, d_c = 39)$. (For RBERs near the code's decoding threshold, the uncorrectable bit-erasure rates (UBER) after BP decoding is shown in Fig. 5.2 (a).) For RBERs in the full range from 0 to 1, the average stopping-set sizes (namely, average number of un-decodable erasures after BP-decoding) are shown in Fig. 5.2 (b). It can be seen that the average stopping-set size increases approximately linearly (from 0 to 8192) as RBER increases from 0 to 1.

We now define the capability and limitations of the NR-Decoder. Suppose that for any sliding window of L bits, the NR-Decoder can always correct its erasures if the number of erasures in the window is at most α. Given a stopping set, the objective is to use the NR-Decoder to correct sufficiently many erasures so that the remaining erasures are correctable by the ECC-Decoder. However, notice that since NR-Decoding is typically less efficient than ECC-Decoding, there is an

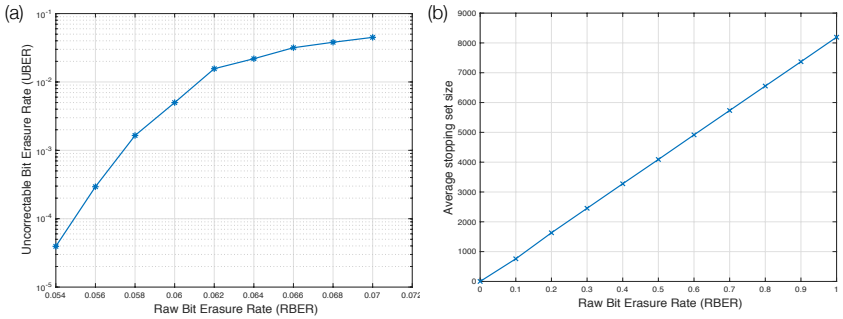

Figure 5.2: Statistics of an (8192,7561) LDPC code. (a) UBER for different RBERs near the code's decoding threshold. (b) Average stopping-set size for different RBERs.

associated cost. Let β denote the number of L-bit windows (at β different locations) used by NR-Decoding. Whether the NR-decoding is implemented in hardware or software (where decoding circuits or software can choose the location of each window), the overall circuit complexity and/or time complexity is proportional to β. (For instance, if circuits are used to decode the windows in parallel, then β circuits are needed for β windows.) Therefore, we need to *minimize* the number of windows used by NR-decoding, and choose the locations of the windows carefully for that purpose.

In this work, we will study a special case of the above problem by setting $L = \alpha = 1$, and we assume that the sliding windows can cover both information bits and parity-check bits. Although it may seem too restrictive at first sight, there are several reasons that still make it quite meaningful. First, storage systems often have low raw bit-error-rates (e.g., less than 0.5%), so for a relatively short window (e.g, tens of bits), the number of erasures in it is often at most 1, which can usually be corrected very effectively by NR-Decoding [28]. In such cases, having $L \geq 1$, $\alpha = 1$ is similar to having $L = \alpha = 1$. Second, ECCs in storage systems often have high rates (e.g., over 0.93), which can make the sliding window's access to all codeword bits similar to accessing only information bits (since information bits are the majority of bits). Third, understanding the basic case of $L = \alpha = 1$ will be the basis for understanding the more general case of $L \geq \alpha \geq 1$. And

last but not least, the case $L = \alpha = 1$ corresponds to a fundamental problem for LDPC codes: assume there is a powerful and unrestricted Oracle decoder that can correct any erasure, but its decoding comes at a high cost; then, how to minimize the number of erasures the Oracle decoder needs to correct in order to make the remaining erasures decodable by the ECC-Decoder? We believe the problem is theoretically important in its own right.

The problem to study can now be defined formally as follows. Let $G = (V \cup C, E)$ be a bipartite graph, where V (representing erasures) is a subset of the variable nodes in an LDPC code's Tanner graph, C is a subset of the check nodes in the same Tanner graph such that every node in C is adjacent to at least one node in V, and E is the set of edges in the Tanner graph with one endpoint in V and another endpoint in C. If every node in C has degree two or more, then G is called a *Stopping Graph* and V is called a *Stopping Set*. If an iterative BP algorithm (as introduced earlier) that runs on G can decode all the variable nodes in V (where every variable node in V is an erasure), then V is called a *Decodable Set* (or simply *decodable*); otherwise, it is a *Non-Decodable Set* (or simply *non-decodable*). Note that a Stopping Set must be a Non-Decodable Set, but not *vice versa*. The problem we study, called *Stopping-Set Elimination (SSE) Problem*, is as follows.

Definition 17. *Given a Stopping Graph $G = (V \cup C, E)$, how to remove the minimum number of variable nodes from V such that the remaining variable nodes are decodable?*

The removed variable nodes represent NR-decoded erasures. Clearly, after the removal, the remaining nodes will no longer contain any Stopping Set.

5.2 Related Applications and Related Works

In this section, we first show two additional applications of the SSE Problem. We then present a brief review of existing works that are related to error correction by NR.

5.2.1 Applications of SSE

In addition to decoding by NR, the SSE Problem also has two additional applications: distributed storage, and satellite-to-ground communication with feedback.

- *Distributed Storage:* Distributed file systems like HDFS have been widely used in big data applications [128]. Typically, they store data in blocks, and ECCs are applied over the blocks (where each block is seen as a codeword symbol of the ECC). Binary LDPC codes are naturally an attractive candidate for distributed storage, as they have excellent code rates, good locality (e.g., a missing block can be recovered by a local disk from a few neighboring blocks), and excellent computational simplicity (only XOR is used for decoding, since when each block has t bits, the decoding can be seen as t binary LDPC codes being decoded in parallel). Meanwhile, almost all big IT companies store multiple copies of their data at different locations. So when one site loses some blocks in an LDPC code and cannot recover them by itself, it needs to retrieve some lost blocks from other remote sites. Since communication with remote sites is much more costly than accessing local disks, it is desirable to minimize the number of blocks retrieved from remote sites as long as the remaining erasures become decodable. And that is the SSE Problem.

- *Satellite-to-Ground Communication with Feedback:* Consider satellite-to-ground communication, where the data (e.g., big sensing images) are partitioned into packets (i.e., blocks), and LDPC codes are applied over the packets (similar to the case for distributed storage) [129]. As the channel is noisy, some packets received by the ground may be un-decodable, and the ground will request the satellite to retransmit some of those lost packets. Since the satellite-to-ground communication can be quite costly, it is desirable to minimize the number of retransmitted packets. That is also the SSE Problem.

5.2.2 Related Works

Error-correction with NR is related to joint source-channel coding and denoising. The idea of using the inherent redundancy in a source – or the leftover redundancy at the output of a source encoder – to enhance the performance of the ECC has been studied within the field of joint source-channel coding. In [97], source-controlled channel coding using a soft-output Viterbi algorithm is considered. In [94], a trellis based decoder is used as a source decoder in an iterative decoding

scheme. Joint decoding of Huffman and Turbo codes is proposed in [96]. In [98], joint decoding of variable length codes (VLCs) and convolutional/Turbo codes is analyzed. Applications of turbo codes to image/video transmission are shown in [95], [100] and [99]. Joint decoding using LDPC codes for VLCs and images are illustrated in [101] and [102], respectively. However, not many works have considered JSCC specifically for language-based sources, and exploiting the redundancy in the language structure via an efficient decoding algorithm remains as a significant challenge. Related to joint source-channel coding, denoising is also an interesting and well studied technique [103, 104, 105, 106, 107, 108, 109, 110, 111]. A denoiser can use the statistics and features of input data to reduce its noise level for further processing. For discrete memoryless channels with stationary input sequences, a universal algorithm that performs asymptotically as well as optimal denoisers are given in [125]. The algorithm is also universal for a semi-stochastic setting, where the channel input is an individual sequence and the randomness in the channel output is solely due to the channel's noise.

Spell-checking softwares are a typical example of using NR to correct errors in languages. They are widely used in text editors. A spell-checking software usually works at the character level (namely, it does not consider how characters or text strings are encoded by bits), is for uncompressed texts, and uses the validity of words and the correctness of grammar to correct errors that appear in the typing of texts.

Using NR to correct errors at the bit level in compressed texts has been studied in a number of works. In [112], texts compressed by Huffman coding is considered, and a dynamic programming algorithm is used to partition the noisy bit sequence into subsequences that represents words, and to select likely solutions based on the frequencies of words and phrases. In [1], texts that are compressed by Huffman coding and then protected by LDPC codes are studied. An efficient greedy algorithm is used to decompress the noisy bit string, and partition it into stable and unstable regions based on whether each region contains recognizable words and phrases. The stable and unstable regions have polarized RBERs, which are provided as soft information to the LDPC code for better decoding performance. The algorithm is enhanced in [113] by a machine learning method for

content recognition, and an iterative decoding algorithm between the NR-Decoder and the ECC-Decoder is used to further improve performance. In [114], texts compressed by Huffman coding and protected by Polar codes are studied. The validity of words is used to prune branches in a list sequential decoding algorithm, and a trie data structure for words is used to make the algorithm more efficient. A concatenated-code model that views the text with NR as the outer code and the Polar code as the inner code is considered, and the rate improvement for the Polar code due to NR is analyzed. That model is further studied in [126], where an optimal algorithm that maximizes the code rate improvement by unfreezing some frozen bits to store information is presented. A model that views NR as the output of a side information channel at the channel decoder is also studied, where NR is shown to improve the random error exponent.

5.3 NP-Hardness of SSE Problem

In this section, we prove that the SSE Problem is NP-hard. The proof has two steps: first, using the well-known Set Cover Problem, we prove that a related covering problem where nearly all elements are covered – which we call the *Pseudo Set Cover Problem* – is NP-complete; then, we reduce the latter problem to the SSE Problem.

5.3.1 NP-completeness of Pseudo Set Cover Problem

Consider the well-known Set Cover Problem. Let $T = \{t_1, t_2, \cdots, t_n\}$ be a universe of n elements. Let $S_1, S_2, \cdots, S_m$ be m subsets of T such that $T = \bigcup_{i=1}^{m} S_i$. Let $k \leq m$ be a positive integer. The Set Cover Problem asks if there exist k subsets $S_{i_1}, S_{i_2}, \cdots, S_{i_k}$ such that $T = \bigcup_{j=1}^{k} S_{i_j}$. (Note that a subset S_i is said to "cover" its elements. So the Set Cover Problem asks if there exist k subsets that together cover all the elements of T.)

Let us now define a related problem called the *Pseudo Set Cover Problem*. It has the same input as the Set Cover Problem, and differs only in its question: it asks if there exist k subsets $S_{i_1}, S_{i_2}, \cdots, S_{i_k}$ such that $|\bigcup_{j=1}^{k} S_{i_j}| \geq |T| - 1$. (Instead of covering all the $|T|$ elements, the Pseudo Set Cover Problem aims at covering at least $|T| - 1$ elements.) We now prove that the problem is NP-complete.

Theorem 18. *The Pseudo Set Cover Problem is NP-complete.*

Proof. It is easy to see that the Pseudo Set Cover Problem is in NP. We now construct a polynomial-time reduction from the Set Cover Problem to the Pseudo Set Cover Problem.

Let an instance of the Set Cover Problem have input parameters $T = \{t_1, t_2, \cdots, t_n\}$, S_1, S_2, $\cdots$, S_m and $k \leq m$ as introduced above. For the corresponding instance of the Pseudo Set Cover Problem, let its universe of elements be

$$T' = \{t_1, t_2, \cdots, t_n, t_{n+1}\},$$

where t_{n+1} is a new element, and let its subsets be

$$S_1, S_2, \cdots, S_m, S_{m+1},$$

where

$$S_{m+1} = \{t_{n+1}\}.$$

It is simple to see that the mapping between the two instances takes polynomial (in fact, linear) time.

Let us now prove the following claim: the Set Cover Problem has k subsets covering all the n elements in T if and only if the Pseudo Set Cover Problem has k subsets covering at least $|T'| - 1 = n$ elements in T'.

Consider one direction of the proof: suppose that the Set Cover Problem has k subsets covering all elements of T. Then the same k subsets cover exactly n elements of T'. (The only uncovered element is t_{n+1}.)

Now consider the other direction of the proof: suppose that the Pseudo Set Cover Problem has k subsets

$$S_{i_1}, S_{i_2}, \cdots, S_{i_k}$$

covering at least n elements in T'. Without loss of generality (WLOG), assume that

$$i_1 < i_2 < \cdots < i_k.$$

There are three possible cases:

- Case 1: The k subsets cover all the $n+1$ elements of T'. Then $i_k = m+1$, and the remaining $k-1$ subsets cover all the elements in T. By adding to the $k-1$ remaining subsets any other subset in $\{S_1, S_2, \cdots, S_m\}$, we get k subsets covering all elements of T for the Set Cover Problem.

- Case 2: The k subsets cover n elements of T', including t_{n+1}. Then $i_k = m+1$, and there must be exactly one uncovered element in T. Say that uncovered element is t_i, and let S_j (where $1 \leq j \leq m$) be any subset that contains t_j. (Such a subset S_j must exist.) By replacing $S_{i_k} = S_{m+1}$ by S_j, we get k subsets that cover all the elements of T.

- Case 3: The k subsets cover n elements of T', but not covering t_{n+1}. Then the same k subsets cover all the elements of T.

Therefore there is a polynomial-time reduction from the Set Cover Problem to the Pseudo Set Cover Problem. Since the Set Cover Problem is known to be NP-complete, the conclusion holds. □

5.3.2 NP-hardness of Stopping Set Elimination Problem

We now prove the NP-hardness of the SSE Problem by using a reduction from the Pseudo Set Cover Problem. Let us begin with some constructions.

Consider the bipartite graph shown in Fig. 5.3 (a). It consists of four variable nodes (s_i, t_j, $u_{i,j}$ and $w_{i,j}$) and three check nodes ($c^1_{i,j}$, $c^2_{i,j}$ and $c^3_{i,j}$). We denote it by $D_{i,j}$ to indicate that it connects node s_i and node t_j. We prove some basic property it has on iterative BP decoding.

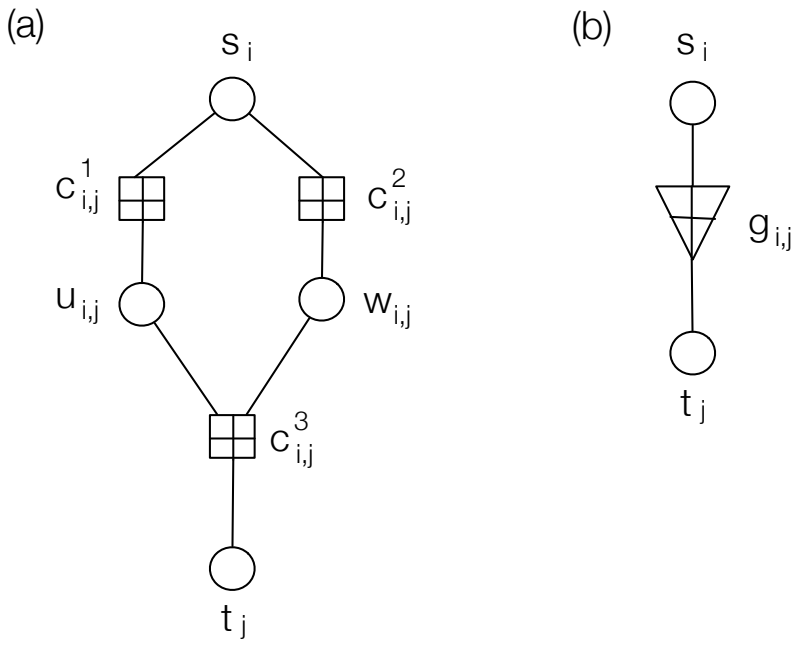

Figure 5.3: (a) A bipartite graph $D_{i,j}$ that connects variable nodes s_i and t_j. (b) A symbol for the graph $D_{i,j}$.

Lemma 19. *In the graph $D_{i,j}$ that contains the variable nodes s_i, t_j, $u_{i,j}$, $w_{i,j}$ as a Stopping Set, if the value of the variable node s_i becomes known, the BP decoding algorithm will recover the values of all the three remaining variable nodes.*

On the other hand, if the value of the variable node t_j becomes known, the BP decoding algorithm will not recover the value of any of the other three variable nodes.

Proof. If the value of s_i becomes known, by using the check nodes $c^1_{i,j}$ and $c^2_{i,j}$, the BP decoding algorithm will recover the values of $u_{i,j}$ and $w_{i,j}$, respectively. Then via the check node $c^3_{i,j}$, it will recover the value of t_j.

If the value of t_j becomes known, since $c^3_{i,j}$ has degree 3, the BP algorithm will not recover any more values. □

The graph $D_{i,j}$ will be viewed as a "gadget" that connects node s_i with node t_j. To simplify

the presentation, in the following, we often represent it by the symbol shown in Fig. 5.3 (b), where the "gate" $g_{i,j}$ represents the five nodes ($c^1_{i,j}$, $c^2_{i,j}$, $c^3_{i,j}$, $u_{i,j}$, $w_{i,j}$) and their incident edges. The "direction" of the gate $g_{i,j}$ indicates the "directed" property shown in the above lemma: decoding s_i leads to decoding t_j, but not *vice versa*.

Consider the Pseudo Set Cover Problem with input parameters $T = \{t_1, t_2, \cdots, t_n\}$, S_1, S_2, $\cdots$, S_m and $k \leq m$ as introduced earlier. To reduce it to the SSE Problem, we will map every instance of the Pseudo Set Cover Problem to some instance of the SSE Problem.

Let us start by building a graph G_I. We start by assigning $m + n$ nodes: for every subset S_i (for $1 \leq i \leq m$) or element t_j (for $1 \leq j \leq n$) in the Pseudo Set Cover Problem, there is a corresponding variable node s_i or t_j in G_I. Then, whenever the Pseudo Set Cover Problem has

$$t_j \in S_i,$$

we connect nodes s_i and t_j by the bipartite graph $D_{i,j}$. The graph obtained this way is G_I. An example is shown below.

Example 20. *Let an instance of the Pseudo Set Cover Problem be $T = \{t_1, t_2, t_3, t_4, t_5\}$ and $S_1 = \{t_1, t_3, t_4\}$, $S_2 = \{t_1, t_3\}$, $S_3 = \{t_2, t_4, t_5\}$. The value of parameter k is irrelevant to the mapping, so we do not specify it here. The instance is illustrated in Fig. 5.4 (a), where there is an edge between S_i and t_j if and only if $t_j \in S_i$.*

The corresponding graph G_I is shown in both Fig. 5.4 (b) and (c), where the symbol for each $D_{i,j}$ is used in Fig. 5.4 (b), and the full details of G_I are shown in Fig. 5.4 (c). It is easy to see the correspondence between G_I and the Pseudo Set Cover Problem. □

It is clear that G_I is a bipartite graph.

Let us now create a graph G_{II} as follows. Given graph G_I, we add $m + 1$ additional check nodes

$$c_0, c_1, c_2, \cdots, c_m.$$

For $0 \leq i \leq m$ and $1 \leq j \leq n$, add an edge between the check node c_i and the variable node

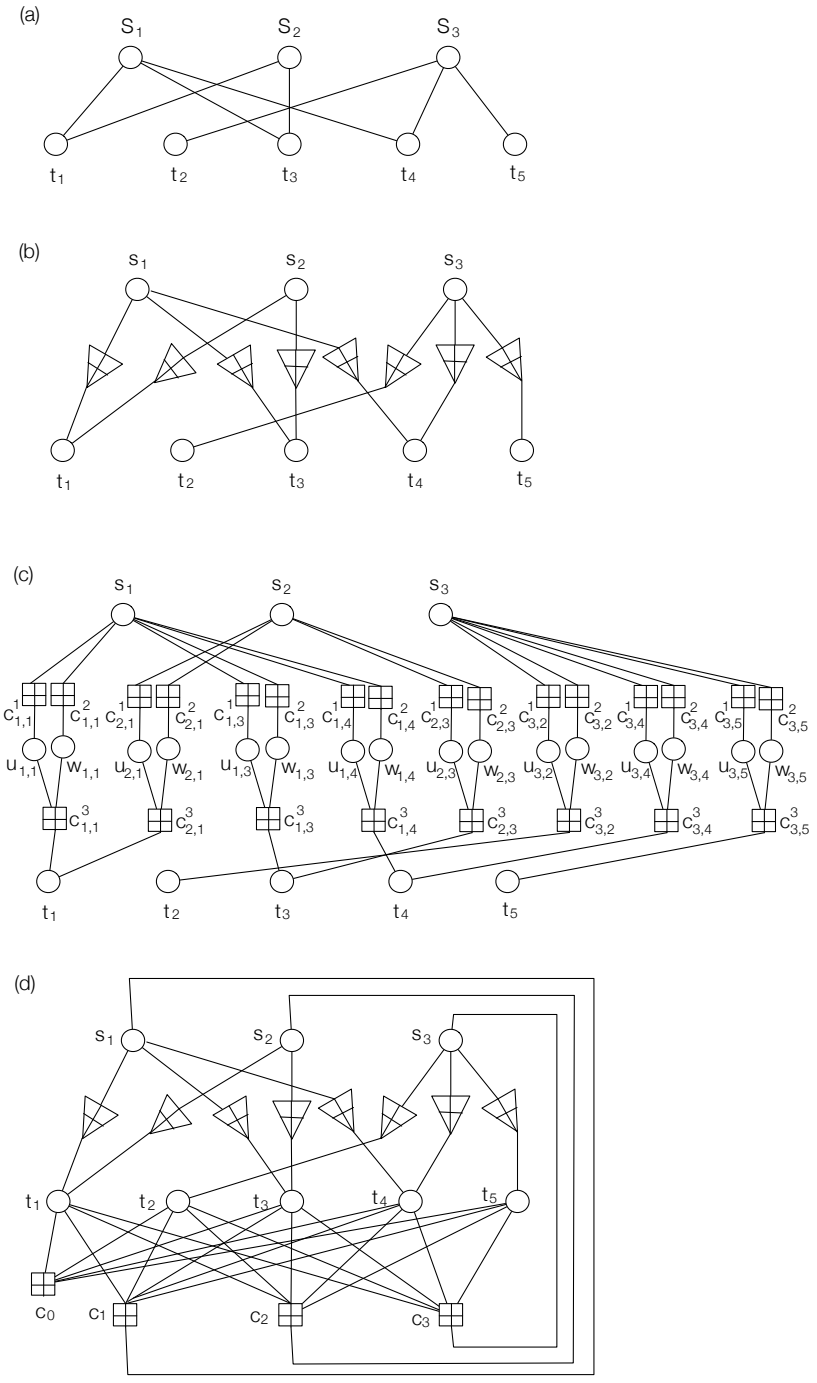

Figure 5.4: (a) An instance of the Pseudo Set Cover Problem, where $T = \{t_1, t_2, t_3, t_4, t_5\}$ and $S_1 = \{t_1, t_3, t_4\}$, $S_2 = \{t_1, t_3\}$, $S_3 = \{t_2, t_4, t_5\}$. (b) The corresponding graph G_I. (c) The corresponding graph G_I with full details. (d) The corresponding graph G_{II}.

t_j. For $1 \leq i \leq m$, add an edge between the check node c_i and the variable node s_i. The graph obtained this way is G_{II}. An example is shown below.

Example 21. *Following Example 20, the graph G_{II} is shown in Fig. 5.4 (d).* □

It is clear that G_{II} is also a bipartite graph.

In the following, we consider only cases where $n > 1$. (The case $n = 1$ is trivial.) It is then simple to see that in G_{II}, the degree of every check node is at least two. So it is a Stopping Graph, namely, an instance of the SSE Problem.

Lemma 22.. *If for the Pseudo Set Cover Problem, there exist k subsets that cover at least $n - 1$ elements of T, then for the corresponding graph G_{II}, k variable nodes can be removed so that the remaining variable nodes form a Decodable Set.*

Proof. Suppose that

$$S_{i_1}, S_{i_2}, \cdots, S_{i_k}$$

are k chosen subsets that cover at least $n - 1$ elements of T. Let us remove the corresponding k variable nodes

$$s_{i_1}, s_{i_2}, \cdots, s_{i_k}$$

from the graph G_{II}. Since removing a variable node is equivalent to turning the node from an erasure to a known value, by the "directed" property of $D_{i,j}$ proved earlier, we know that the BP decoding algorithm will recover the values of at least $n - 1$ variable nodes among

$$t_1, t_2, \cdots, t_n.$$

That is because if an element t_j is covered by some chosen subset S_{i_r} (where $1 \leq r \leq k$), since the value of the variable node s_{i_r} is now known, via the "gadget" $D_{i_r,j}$, the BP decoding algorithm can recover the value of t_j.

We now show that the BP decoding algorithm can recover the values of all n variable nodes $t_1, t_2, \cdots, t_n$. From the above discussion, we know that at most one of them – say t_x – is not

decoded yet. So the BP algorithm can use the check node c_0 (which has degree n) to recover the value of t_x as

$$t_x = \oplus_{1 \le i \le n, i \neq x} t_i.$$

Since the values of $t_1, t_2, \cdots, t_n$ are all known now, for $i = 1, 2, \cdots, m$, the BP decoding algorithm can use the check node c_i to recover the value of s_i (if its value is not already known). So all the variable nodes can recover their values. Therefore, the remaining variable nodes form a Decodable Set. □

When a set of variable nodes $S \subseteq V$ is removed from a Stopping Graph $G = (V \cup C, E)$, if the remaining nodes of V become a Decodable Set, let us call S an *Elimination Set* of size $|S|$.

Lemma 23. *If G_{II} has an Elimination Set of size $k \le m$, then G_{II} has an Elimination Set of size k that is also a subset of*

$$\{s_1, s_2, \cdots, s_m\}.$$

.

Proof. Let

$$X = \{x_1, x_2, \cdots, x_k\}$$

be an Elimination Set of G_{II}, where each x_i is a variable node. Let us create a set

$$Y = \{y_1, y_2, \cdots, y_k\} \subseteq \{s_1, \cdots, s_m\}$$

as follows. For $i = 1, 2, \cdots, k$, do:

- If $x_i \in \{s_1, s_2, \cdots, s_m\}$, let $y_i = x_i$.
- If x_i is either $u_{i',j'}$ or $w_{i',j'}$ – namely, it is a variable node in the "gadget" $D_{i',j'}$ (more specifically, $g_{i',j'}$) that connects $s_{i'}$ and $t_{j'}$ – let $y_i = s_{i'}$ if $s_{i'}$ is not in Y yet, and let y_i be any node in $\{s_1, s_2, \cdots, s_m\}$ that is not yet in Y otherwise.

- If $x_i = t_j$ for some $1 \leq j \leq n$, let $s_{i'}$ be a node such that there is a "gadget" $D_{i',j}$ connecting $s_{i'}$ and t_j. (Such a node $s_{i'}$ must exist because in the Pseudo Set Cover Problem, t_j is covered by at least one subset.) If $s_{i'}$ is not in Y yet, let $y_i = s_{i'}$; otherwise, let y_i be any node in $\{s_1, s_2, \cdots, s_m\}$ that is not yet in Y.

With the above construction, for any node x_i in X, there exists a node $s_{i'}$ in Y such that either $s_{i'} = x_i$, or $s_{i'}$ and x_i exist in the same "gadget" $D_{i',j}$ for some j. By the "directed" property of gadgets $D_{i',j}$, we see that when the values of variable nodes in Y are known, the BP algorithm can decode all the variable nodes in X; and since X is an Elimination Set, the BP algorithm can consequently decode all the variable nodes in G_{II}. So Y is an Elimination Set of size k that is a subset of $\{s_1, s_2, \cdots, s_m\}$. □

Lemma 24. *If G_{II} has an Elimination Set of size k $\{s_{i_1}, s_{i_2}, \cdots, s_{i_k}\} \subseteq \{s_1, s_2, \cdots, s_m\}$, then for the corresponding Pseudo Set Cover Problem, the k subsets*

$$S_{i_1}, S_{i_2}, \cdots, S_{i_k}$$

cover at least $n - 1$ elements of T.

Proof. The proof is by contradiction. Suppose that $S_{i_1}, S_{i_2}, \cdots, S_{i_k}$ cover at most $n - 2$ elements of T. Then in G_{II}, when the values of $\{s_{i_1}, s_{i_2}, \cdots, s_{i_k}\}$ are known, the BP algorithm can use the "gadgets" $D_{i,j}$ to decode at most $n - 2$ variable nodes among

$$t_1, t_2, \cdots, t_n.$$

Then the BP algorithm gets stuck because it cannot use any check node to decode any more variable node:

- For any check node c_i (where $0 \leq i \leq m$), at least two adjacent nodes in $\{t_1, t_2, \cdots, t_n\}$ are not decoded yet. So the BP algorithm cannot use c_i to decode more variable nodes.

- For any "gadget" $D_{i,j}$ that connects s_i and t_j, if $s_i \notin \{s_{i_1}, s_{i_2}, \cdots, s_{i_k}\}$, by the "directed" property of the gadget, the BP algorithm cannot use it to decode s_i whether the node t_j has been decoded or not.

That means $\{s_{i_1}, s_{i_2}, \cdots, s_{i_k}\}$ is not an Elimination Set, which is a contradiction. That leads to the conclusion. □

By combining the above two lemmas, we get:

Lemma 25. *If G_{II} has an Elimination Set of size $k \leq m$, then for the corresponding Pseudo Set Cover Problem, there exist k subsets that cover at least $n-1$ elements of T.*

We now prove our main result here.

Theorem 26. *The SSE Problem is NP-hard.*

Proof. The SSE Problem is an optimization problem. Let us consider its decision problem: given a Stopping Graph $G = (V \cup C, E)$ and a positive integer k, does it have an Elimination Set of size k? Let us call this decision problem P_{sse}. It is clear that $P_{sse} \in NP$.

We have shown a mapping that maps every instance of the Pseudo Set Cover Problem to an instance of P_{sse}. The mapping takes polynomial time. By combining Lemma 22 and Lemma 25, we see that the answer to the Pseudo Set Cover Problem is "yes" (namely, there exist k subsets that cover at least $n-1$ elements of T) if and only if the answer to P_{sse} is "yes" (namely, G_{II} has an Elimination Set of size k). So the mapping is a polynomial-time reduction. By Theorem 18, the Pseudo Set Cover Problem is NP-complete. So P_{sse} is NP-complete, which leads to the conclusion. □

5.4 SSE with Constraint on Belief-Propagation Iterations and Its NP-Hardness

In this section, we extend the SSE Problem by considering the time for BP decoding. After the nodes in an Elimination Set are removed (namely, after NR-decoding corrects those erasures), the remaining erasures are guaranteed to form a Decodable Set, and therefore the BP decoder

can correct them. However, there is no guarantee on *how many iterations* are needed by the BP decoder to correct the remaining erasures. Here we assume a standard parallel-implementation of BP decoding: in each iteration, first, all variable nodes transmit their values to neighboring check nodes in parallel; then, all check nodes use incoming messages to correct erasures and send the decoding results back to variable nodes, also in parallel. So the time for BP decoding can be measured by the number of BP iterations.

It can be seen that for a Stopping Set of n variable nodes (namely, n erasures), after an Elimination Set is removed, the BP decoder may still use as many as $\Theta(n)$ iterations to correct the remaining erasures. The example below is an illustration.

Example27. *A stopping set of n variable nodes and n check nodes are shown in Fig. 5.5 (a), where all nodes have degree two and they together form a cycle. By eliminating one variable node v_1, the remaining variable nodes become decodable, as shown in Fig. 5.5 (b). Then the BP decoder corrects two variable nodes in each iteration: in the 1st iteration, it corrects v_2 and v_n because they both have a neighboring check node of degree one (as shown in Fig. 5.5 (c)); in the 2nd iteration, it corrects v_3 and v_{n-1} for the same reason (as shown in Fig. 5.5 (d)); and so on. So the BP decoder will use $\lceil \frac{n-1}{2} \rceil = \Theta(n)$ iterations to correct the remaining erasures.* $\square$

For BP decoding, its decoding time is an important measure of performance. So it is useful to limit the number of iterations needed by BP decoding, which offers a performance guarantee. That motivates us to study this extended SSE Problem.

Definition28. *Given a Stopping Graph $G = (V \cup C, E)$ and an integer k, how to remove the minimum number of variable nodes from V such that the remaining variable nodes can be corrected by the BP decoder in no more than k iterations?*

We call the above problem the SSE_k Problem. In comparison, the SSE Problem studied earlier has no constraint on k, so it can be seen as the SSE_∞ Problem.

We have already proved that SSE_∞ is NP-hard. The question now is: if k is a constant – namely, we want the BP decoding to finish within a fixed number of iterations – does the SSE_k

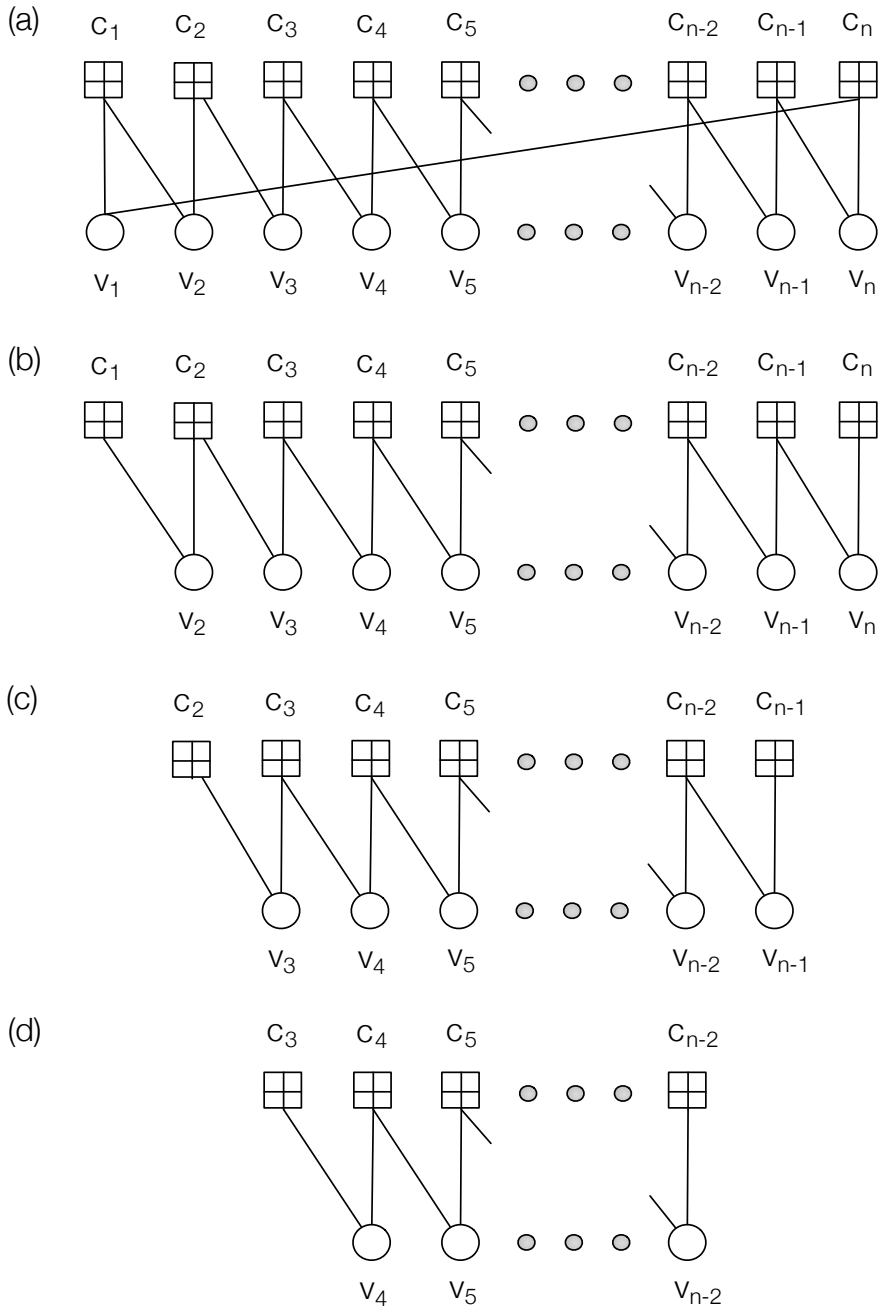

Figure 5.5: (a) A Stopping Set of n variable nodes and n check nodes. (b) After removing a variable node v_1, the remaining nodes become decodable. (c) After the 1st iteration of BP decoding, v_2 and v_n are corrected. (d) After the 2nd iteration of BP decoding, v_3 and v_{n-1} are corrected.

problem become polynomial-time solvable? A positive answer seems possible at first sight, because having a small k puts more constraints on solutions and limits its search space. For example, if $k = 1$, to correct all remaining erasures in just one iteration, in the subgraph induced by the remaining variable nodes and their adjacent check nodes, every variable node needs to be adjacent to at least one check node of degree one. That is a very local property for the bipartite graph and can possibly make the problem simpler. However, our study below will give a negative answer. We will prove that even the

$$SSE_1$$

Problem is NP-hard.

There have been a number of works on the *node-deletion problem* (also called the *maximum subgraph problem*) [130, 131, 132, 133], which can be generally stated as follows: find the minimum number of vertices to delete from a given graph so that the remaining subgraph satisfies a property π. The node-deletion problem includes many well-known problems as special cases. Some examples are:

- Max Clique Problem: the property π is that the remaining subgraph is a complete graph.
- Feedback Vertex Set Problem: the property π is that the remaining subgraph has no cycles.
- Vertex Cover Problem: the property π is that the remaining subgraph contains only isolated nodes, without edges.

Some node-deletion problems are NP-complete on both general graphs and bipartite graphs, such as the feedback vertex set problem. However, some are NP-complete on general graphs but polynomial-time solvable on bipartite graphs, such as the vertex cover problem.

The SSE_k Problem is different from the previously studied problems in several ways. First, its property π is for the remaining subgraph to be decodable within k iterations, which is different from the property π in other problems. Second, the previous works focus on properties π that are *hereditary on induced subgraphs*, namely, whenever a graph G satisfies π, by deleting nodes from

G, the remaining subgraphs also satisfies π [130, 131, 132, 133]. (For example, the property π for the max clique problem is hereditary because when nodes are removed from a complete graph, the remaining subgraph is also a complete graph. The same holds for the feedback vertex set problem and the vertex cover problem.) However, for the SSE_k Problem, the property π is *not hereditary*, because when a check node is removed, it may turn a Decodable Set into a Non-decodable Set. An example is shown below.

Example 29. *A Decodable Set is shown in Fig. 5.6 (a), which satisfies the property π of the SSE_k problem. As shown in Fig. 5.6 (b), after the check nodes c_1 and c_3 are removed, the remaining subgraph becomes non-decodable, which violates the property π. So for the SSE_k Problem, the property π is not hereditary.* □

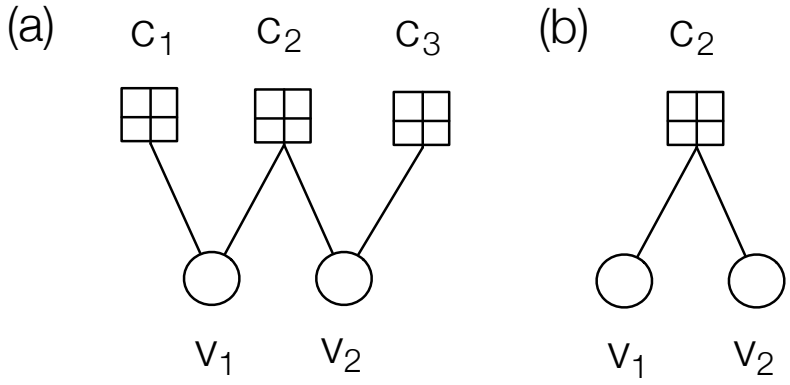

Figure 5.6: (a) A graph with a Decodable Set. (b) After check nodes c_1 and c_3 are removed, the remaining variable nodes form a Non-decodable Set.

We now prove the NP-hardness of the SSE_1 Problem. We use a reduction from the NP-complete Not-all-equal SAT Problem [134], similar to a proof technique used in [133]. However, due to the differences between the SSE_k problem and the previously studied node-deletion problems (as mentioned above), the two proofs also have significant differences: they use different

mappings from the Not-all-equal SAT Problem to the target problem, which also lead to some substantially different properties in the mapped structures.

We first define the Not-all-equal SAT Problem [134]: Let $x_1, x_2, \cdots, x_n$ be n Boolean variables. A *literal* is either x_i or $\bar{x}_i$ (namely, the NOT of x_i) for some $i \in \{1, 2, \cdots, n\}$. Let a *clause* be a set of three literals. Let

$$S = \{C_1, C_2, \cdots, C_k\}$$

be a set of k *clauses*. The question is: *Is there a truth assignment to the n Boolean variables such that for every clause in S, the three literals in the clause are neither all true nor all false (namely, every clause has at least one true literal and also at least one false literal)?* (If the answer is "yes", the problem is called "satisfiable".)

By convention, "true" is also represented by 1, and "false" is also represented by 0. We give an example of the Not-all-equal SAT Problem.

Example 30. *Consider the following instance of the Not-all-equal SAT Problem. Let $n = 4$ and $k = 5$. Let the Boolean variables be x_1, x_2, x_3, x_4, and let the set of clauses be $C_1 = (x_1, \bar{x}_2, x_3)$, $C_2 = (\bar{x}_1, \bar{x}_2, x_4)$, $C_3 = (x_2, x_3, x_4)$, $C_4 = (x_1, \bar{x}_3, \bar{x}_4)$, $C_5 = (\bar{x}_1, x_2, x_3)$.*

The above instance is satisfiable because we can let the truth assignment be

$$x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 1, x_5 = 1.$$

Correspondingly, the clauses become $C_1 = (1, 0, 0)$, $C_2 = (0, 0, 1)$, $C_3 = (1, 0, 1)$, $C_4 = (1, 1, 0)$, $C_5 = (0, 1, 0)$. None of the clauses is $(1, 1, 1)$ (namely, all true) or $(0, 0, 0)$ (namely, all false). □

We now construct a mapping from the Not-all-equal SAT Problem to the SSE_1 Problem. And afterwards, we will analyze its properties.

5.4.1 Reducing Not-all-equal SAT Problem to SSE_1 Problem

In this subsection, we construct a reduction that maps every instance of the Not-all-equal SAT Problem to an instance of the SSE_1 Problem.

For every Boolean variable x_i of the Not-all-equal SAT Problem (for $1 \leq i \leq n$), we create a graph as shown in Fig. 5.7 (a), which will be called the "gadget V_i". It is a bipartite graph of three variable nodes and three check nodes. (Here nodes X_i^1 and X_i^0 represent the true and false values of x_i, respectively.)

For every clause C_j of the Not-all-equal Problem (for $1 \leq j \leq k$), we create two graphs as shown in Fig. 5.7 (b), which will be called gadgets U_j^1 and U_j^2, respectively. (Here for $t = 1, 2, 3$, nodes A_j^t and B_j^t represent the true and false values of the t-th literal in clause C_j, respectively.) We then connect them into one larger gadget W_j as shown in Fig. 5.7 (c), where for $t = 1, 2, 3$, two paths are used to connect the nodes A_j^t and B_j^t. (For example, the two paths between A_j^1 and B_j^1 have nodes d_j^1, d_j^2 and the four check nodes by them.)

In the final graph corresponding to the instance, the gadget V_i will be connected to the rest of the graph only through nodes X_i^1 and X_i^2. So to simplify the presentation, we sometimes represent V_i by the symbol in Fig. 5.7 (d), where the two "interface nodes" X_i^1, X_i^2 are shown and the remaining details are hidden. Also in the final graph, the gadget W_j will be connected to the rest of the graph only through nodes A_j^1, A_j^2, A_j^3, B_j^1, B_j^2, B_j^3; so we sometimes represent it by the symbol in Fig. 5.7 (e).

We now connect the gadgets for clauses to the gadgets for Boolean variables. Consider a clause C_j, and assume its literals are

$$C_j = (l_1, l_2, l_3).$$

For $t = 1, 2, 3$, if $l_t = x_i$ for some $1 \leq i \leq n$, we connect A_j^t to X_i^1 and connect B_j^t to X_i^0 (through some intermediate nodes) as shown in Fig. 5.7 (f). Otherwise $l_t = \bar{x}_i$ for some $1 \leq i \leq n$, and we connect A_j^t to X_i^0 and connect B_j^t to X_i^1 as shown in Fig. 5.7 (g).

Example31. *Assume that a clause is $C_j = (l_1, l_2, l_3) = (x_1, x_3, \bar{x}_4)$. Its gadget W_j is connected to the gadgets V_1, V_3, V_4 as in Fig. 5.7 (h).*

To simplify the presentation of the graph, we represent the connection between a node A_j^t (or B_j^t) and a node x_i^1 (or x_i^0) by a rectangle that is generally denoted by the "H bar". Then the graph

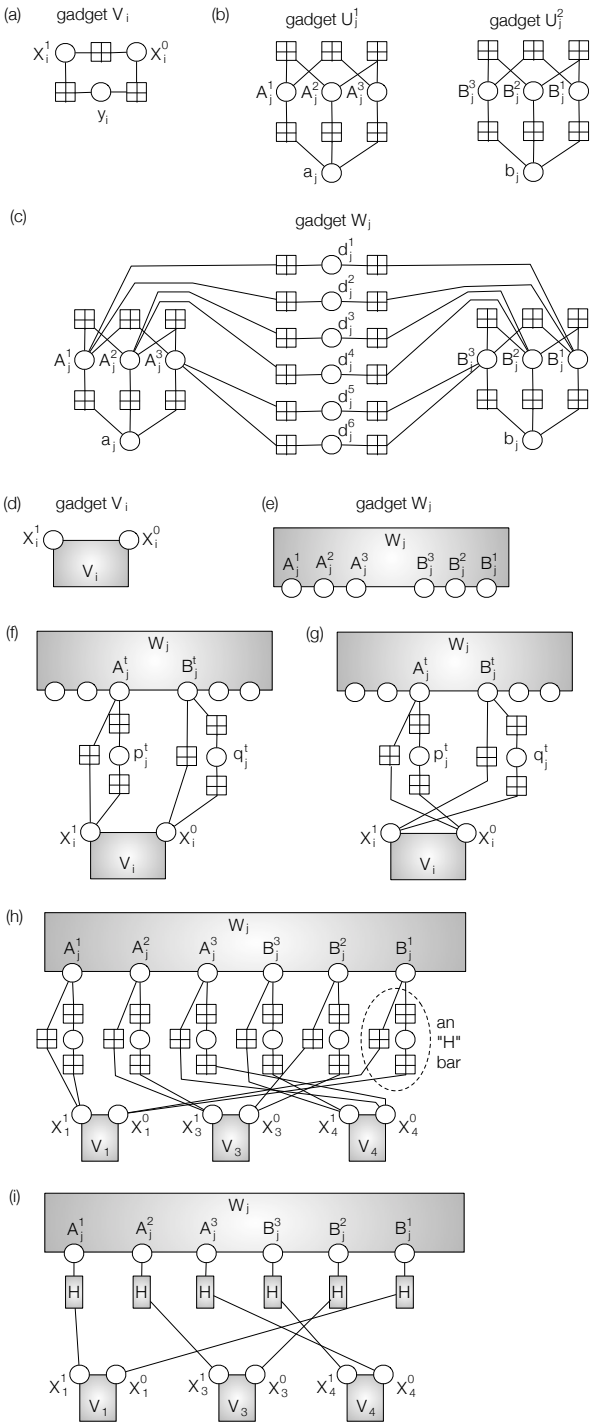

Figure 5.7: (a) The gadget corresponding to a Boolean variable x_i, for $i = 1, 2, \cdots, n$. (b) Two gadgets corresponding to a clause C_j, for $j = 1, 2, \cdots, k$. (c) The connected gadget corresponding to a clause C_j, for $j = 1, 2, \cdots, k$. (d) Symbol for V_i. (e) Symbol for W_j. (f) Connect clause gadget to Boolean variable gadget: case one. (g) Connect clause gadget to Boolean variable gadget: case two. (h) An example of connecting a clause gadget to variable gadgets. (i) Simplified representation of the graph in (h).

in Fig. 5.7 (h) is simplified as the presentation in Fig. 5.7 (i), which shows the connections more clearly. However, it should be noted that each A_j^t, B_j^t, x_i^1 or x_i^0 is connected to an H bar via two edges, not one. □

By now, we have constructed the whole graph that corresponds to an instance of the Not-all-equal Problem. The graph will be denoted by

$$G_{sse}.$$

Let us see an example.

Example 32. *For the Not-all-equal Problem, let $n = 4$ and $k = 2$. Let the two clauses be*

$$C_1 = (x_1, x_3, \bar{x}_4),$$

$$C_2 = (x_1, \bar{x}_2, \bar{x}_3).$$

Then the corresponding graph G_{sse} is shown in Fig. 5.8 (a), where its gadgets are represented by symbols for clarity, and also in Fig. 5.8 (b), where its full details are presented. □

It is easy to see that G_{sse} is a bipartite graph, where every check node has degree more than one. (Specifically, every check node has degree two.) So G_{sse} is a Stopping Graph.

The subsequent analysis will prove that the Not-all-equal SAT Problem is satisfiable if and only if G_{sse} has an Elimination Set of size

$$n + 3k$$

such that after its nodes are removed, the BP algorithm can decode the remaining variable nodes in just one iteration.

5.4.2 Properties of Reduction

In the previous subsection, the mapping from any instance of the Not-all-equal SAT Problem to a graph G_{sse} is shown. We now analyze its properties.

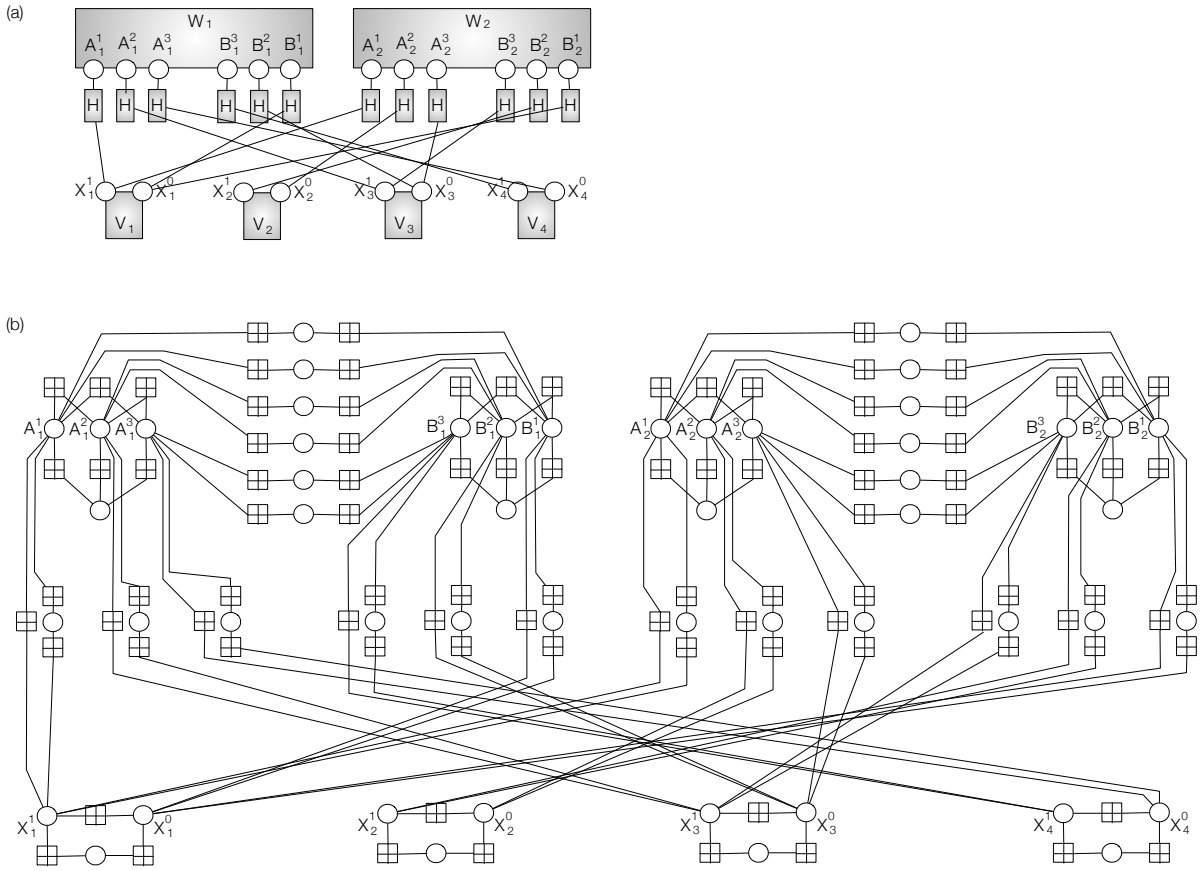

Figure 5.8: (a) The graph G_{sse} corresponding to the Not-all-equal Problem where $n = 4$, $k = 2$, $C_1 = (x_1, x_3, \bar{x}_4)$, $C_2 = (x_1, \bar{x}_2, \bar{x}_3)$, where gadgets are represented by symbols. (b) The same graph G_{sse} in full detail.

We first show a general property for the SSE_1 Problem.

Lemma 33. *Given a bipartite graph G (including the graph G_{sse}), where variable nodes represent erasures, the BP algorithm can decode all erasures in just one iteration if and only if for every variable node in G, it has at least one neighboring check node whose degree is 1.*

Proof. In each iteration of BP decoding, a check node c can recover the value of a neighboring variable node v if and only if v is its only neighboring erasure. □

Note that it is unnecessary for all neighboring check nodes to have degree 1. For example, the graph in Fig. 5.6 can be decoded in one iteration, although both v_1 and v_2 have a neighboring check node c_2 that has degree two. In one iteration of BP decoding, the check node c_1 (which has degree 1) is sufficient for decoding v_1, and the check node c_3 (which also has degree 1) is sufficient for decoding v_2.

Let the bipartite graph G_{sse} be

$$G_{sse} = (V_{sse} \cup C_{sse}, E_{sse}),$$

where V_{sse} is the set of variable nodes, C_{sse} is the set of check nodes, and E_{sse} is the set of edges. We now define the concepts of *Interface Nodes*, *One-Iteration Elimination Set* and *Canonical Elimination Set*.

Definition 34. *Let*

$$\begin{aligned} I_{sse} \triangleq \ & \{X_i^j \mid 1 \le i \le n, 0 \le j \le 1\} \cup \\ & \{A_i^j \mid 1 \le i \le k, 1 \le j \le 3\} \cup \\ & \{B_i^j \mid 1 \le i \le k, 1 \le j \le 3\} \end{aligned}$$

be a subset of variable nodes in G_{sse}. Every node in I_{sse} is called an "Interface Node."

As an example, the interface nodes are shown as circles in Fig. 5.8 (a).

Definition 35. *Let $T \subseteq V_{sse}$ be a set of variable nodes in G_{sse}. If after removing T from G_{sse}, the BP algorithm can decode the remaining variable nodes in one iteration, then T is called a*

"One-Iteration Elimination Set."

If T is a one-iteration elimination set and

$$T \subseteq I_{sse},$$

then T is called a "Canonical Elimination Set."

Lemma36. *If G_{sse} has a One-Iteration Elimination Set of α nodes, then G_{sse} also has a Canonical Elimination Set of at most α nodes.*

Proof. Let $F \subseteq V_{sse}$ be a One-Iteration Elimination Set of α nodes. We will prove the existence of a Canonical Elimination Set $\hat{F} \subseteq I_{sse}$ with $|\hat{F}| \leq \alpha$ nodes. Note that the nodes in G_{sse} are in three kinds of gadgets: gadget V_i, gadget W_j, or H bar. (See Fig. 5.8 (a) for an illustration.) The main idea of the proof is to transform F into $\hat{F}$ by switching nodes of F to interface nodes.

First, consider a gadget V_i (for $1 \leq i \leq n$). (See Fig. 5.7 (a) for an illustration.) Note that X_i^1 and X_i^0 are its only two nodes connecting to nodes outside V_i in G_{sse}. (That is why X_i^1 and X_i^0 are called Interface Nodes.) If $y_i \in F$ (note that y_i is the other variable node in the gadget V_i), then consider three cases:

1. $X_i^1 \in F$ and $X_i^0 \in F$. In this case, we can delete y_i from F and still get a one-iteration elimination set, because y_i's two neighboring check nodes already have degree 1 after X_i^1 and X_i^0 are removed from G_{sse}.

2. $X_i^1 \in F$ or $X_i^0 \in F$. Without loss of generality (WLOG), say $X_i^1 \in F$ and $X_i^0 \notin F$. In this case, we can replace y_i by X_i^0 in F and still get a one-iteration elimination set, because the replacement will only remove more edges from the part of the graph G_{sse} that is outside V_i, and y_i's two neighboring check nodes will have degree 1 after X_i^1 and X_i^0 are removed from G_{sse}.

3. $X_i^1 \notin F$ and $X_i^0 \notin F$. In this case, we can replace y_i by X_i^1 and still get a one-iteration elimination set, because after X_i^1 is removed from G_{sse}, both X_i^0 and y_i will have a neighboring

check of degree 1, and more edges will be removed the part of the graph G_{sse} that is outside V_i.

So we can obtain a new one-iteration elimination set F_1 of at most α nodes, where

$$F_1 \cap \{y_i \mid 1 \leq i \leq n\} = \emptyset.$$

Next, consider an H bar. WLOG, suppose the H bar is between the two nodes A_j^t and X_i^1. (Such an H bar is illustrated in Fig. 5.7 (f).) Note that when the H bar is combined with the nodes A_j^t and X_i^1 (and the edges between them), we get a cycle that has the same structure as the gadget V_i (which is discussed above). So by the same argument that has been used for V_i, from F_1, we can obtain a new one-iteration elimination set F_2 of at most α nodes, where

$$F_2 \cap \{y_i \mid 1 \leq i \leq n\} = \emptyset,$$

$$F_2 \cap \{p_j^t \mid 1 \leq j \leq k,\ 1 \leq t \leq 3\} = \emptyset,$$

and

$$F_2 \cap \{q_j^t \mid 1 \leq j \leq k,\ 1 \leq t \leq 3\} = \emptyset.$$

Now consider a gadget W_j (for $1 \leq j \leq k$), which is shown in Fig. 5.7 (c). First, consider the two nodes d_j^1 and d_j^2, which are on two paths connecting A_j^1 and B_j^1. Those two paths together form a cycle of 4 variable nodes and 4 check nodes, with A_j^1 and B_j^1 as "interface nodes" connecting to the rest of the graph. (This cycle is quite similar to the cycle in gadget V_i.) With a similar analysis as the one for V_i, we can always turn F_2 into a new one-iteration elimination set by replacing d_j^1 and/or d_j^2 by A_j^1 and/or B_j^1. (The only slightly different case to note is when $A_j^1 \notin F_2$ and $B_j^1 \notin F_2$. In this case, we have $d_j^1 \in F_2$ and $d_j^2 \in F_2$, and we can replace them by A_j^1 and B_j^1 in F_2.) The same analysis applies to d_j^3, d_j^4, d_j^5 and d_j^6. So from F_2, we can obtain a new one-iteration elimination set F_3 of at most α nodes, where

$$F_3 \cap \{y_i \mid 1 \leq i \leq n\} = \emptyset,$$

$$F_3 \cap \{p_j^t \mid 1 \le j \le k,\ 1 \le t \le 3\} = \emptyset,$$

$$F_3 \cap \{q_j^t \mid 1 \le j \le k,\ 1 \le t \le 3\} = \emptyset$$

and

$$F_3 \cap \{d_j^t \mid 1 \le j \le k,\ 1 \le t \le 6\} = \emptyset.$$

Now consider the two component gadgets in W_j: U_j^1 and U_j^2, which are shown in Fig. 5.7 (b). WLOG, we just need to consider U_j^1. Suppose $a_j \in F_3$, and consider two cases:

1. $A_j^1 \in F_3$, $A_j^2 \in F_3$, or $A_j^3 \in F_3$. WLOG, suppose $A_j^1 \in F_3$. In this case, we can delete a_j from F_3 and still get a one-iteration elimination set, because after A_j^1 is removed from G_{sse}, A_j^2, A_j^3 and a_j already have neighboring check nodes of degree 1.

2. $A_j^1 \notin F_3$, $A_j^2 \notin F_3$, and $A_j^3 \notin F_3$. In this case, we can replace a_j by A_j^1 in F_3, and still get a one-iteration elimination set, for the same reason as the above case (and the fact that more edges outside U_j^1 will be removed by this replacement because A_j^1 is an "interface node" and a_j is not).

So from F_3, we can obtain a new one-iteration elimination set F_4 of at most α nodes, where

$$F_4 \cap \{y_i \mid 1 \le i \le n\} = \emptyset,$$

$$F_4 \cap \{p_j^t \mid 1 \le j \le k,\ 1 \le t \le 3\} = \emptyset,$$

$$F_4 \cap \{q_j^t \mid 1 \le j \le k,\ 1 \le t \le 3\} = \emptyset$$

$$F_4 \cap \{d_j^t \mid 1 \le j \le k,\ 1 \le t \le 6\} = \emptyset,$$

$$F_4 \cap \{a_j \mid 1 \le j \le k\} = \emptyset,$$

and

$$F_4 \cap \{b_j \mid 1 \le j \le k\} = \emptyset.$$

Let $\hat{F} = F_4$. Clearly, $\hat{F} \subseteq I_{sse}$. That concludes the proof. $\square$

Some properties of Canonical Elimination Sets are shown in the next lemma. We first define "endpoints of an H bar."

Definition 37. *Let u be any node in*

$$\{A_j^t \mid 1 \leq j \leq k, 1 \leq t \leq 3\} \cup \{B_j^t \mid 1 \leq j \leq k, 1 \leq t \leq 3\},$$

and let v be any node in

$$\{X_i^1 \mid 1 \leq i \leq n\} \cup \{X_i^0 \mid 1 \leq i \leq n\}.$$

If u and v are connected by an H bar, then they are called the two endpoints of that H bar.

Example 38. *In Fig. 5.7 (f), the endpoints of H bars are (A_j^t, X_i^1) and (B_j^t, X_i^0). In In Fig. 5.7 (g), such endpoint pairs are (A_j^t, X_i^0) and (B_j^t, X_i^1).* $\square$

Lemma 39. *For the graph G_{sse}, a Canonical Elimination Set F has the following properties:*

- *Property 1: For $i = 1, 2, \cdots, n$, either*

$$X_i^1 \in F$$

or

$$X_i^0 \in F.$$

- *Property 2: For $j = 1, 2, \cdots, k$ and $t = 1, 2, 3$, either*

$$A_j^t \in F$$

or

$$B_j^t \in F.$$

- *Property 3: For $j = 1, 2, \cdots, k$,*

$$|F \cap \{A_j^1, A_j^2, A_j^3\}| \geq 1$$

and

$$|F \cap \{B_j^1, B_j^2, B_j^3\}| \geq 1.$$

- *Property 4: If u and v are the two endpoints of an H bar, then either*

$$u \in F$$

or

$$v \in F.$$

Proof. For the gadget V_i (see Fig. 5.7 (a)), if neither X_i^1 nor X_i^0 is in F, then the BP algorithm cannot decode y_i in one iteration since both of y_i's neighboring check nodes will have degree 2. So Property 1 is true.

For the gadget W_j (see Fig. 5.7 (c)), for the pair (A_j^1, B_j^1), if neither A_j^1 nor B_j^1 is in F, then the BP algorithm cannot decode d_j^1 in one iteration since both of d_j^1's neighboring check nodes will have degree 2. The similar analysis holds for the pairs (A_j^2, B_j^2) and (A_j^3, B_j^3). So Property 2 is true.

For the gadget U_j^1 (see Fig. 5.7 (b)), if none of A_j^1, A_j^2, A_j^3 is in F, then the BP algorithm cannot decode a_j in one iteration since all of d_j^1's three neighboring check nodes will have degree 2. The same analysis holds for U_j^2. So Property 3 is true.

For the H bar (see Fig. 5.7 (f) and (g)) between u and v, if neither u nor v is in F, then the variable node in the H bar – which is labelled as p_j^t or q_j^t in Fig. 5.7 (f) and (g) – cannot be decoded by the BP algorithm in one iteration because both of its neighboring check node will have degree 2. So Property 4 is true. □

Corollary 40. *If F is a One-Iteration Elimination Set of G_{sse}, then*

$$|F| \geq n + 3k.$$

Proof. If F is a Canonical Elimination Set, by Property 1 and Property 2 in Lemma 39, we get $|F| \geq n + 3k$. Then by Lemma 36, the same conclusion holds for any One-Iteration Elimination Set. □

Definition 41. *Let F be a Canonical Elimination Set of G_{sse}. If*

$$|F| = n + 3k,$$

then F is called an "Ideal Elimination Set" of G_{sse}.

Here "Ideal" means "of minimum possible size." Note that an Ideal Eliminate Set may or may not exist for G_{sse}.

Lemma 42. *An Ideal Elimination Set F of G_{sse} has these properties:*

- *Property 1: For $i = 1, 2, \cdots, n$, either X_i^1 or X_i^0 is in F, but not both.*
- *Property 2: For $j = 1, 2, \cdots, k$ and $t = 1, 2, 3$, either A_j^t or B_j^t is in F, but not both.*
- *Property 3: For $j = 1, 2, \cdots, k$, in the set $\{A_j^1, A_j^2, A_j^3\}$, at least one node is in F, and at least one node is not in F. The same is true for the set $\{B_j^1, B_j^2, B_j^3\}$.*
- *Property 4: If u and v are the two endpoints of an H bar, then either u or v is in F, but not both.*

Proof. Given that $|F| = n + 3k$, Properties 1, 2 and 3 here follow directly from Properties 1, 2 and 3 in Lemma 39.

Now consider Property 4 here. WLOG, suppose that $u = A_j^t$ and $v = X_i^1$ for some i, j, t. (The other cases can be analyzed similarly because of symmetry.) Consider two cases:

1. $u \in F$. In this case, by Property 2, $B_j^t \notin F$. By the construction of G_{sse}, B_j^t and X_i^0 are the two endpoints of another H bar. So by Property 4 in Lemma 39, since $B_j^t \notin F$, we get $X_i^0 \in F$. Then by Property 1, we have $v = X_i^1 \notin F$.

2. $u \notin F$. In this case, by Property 4 in Lemma 39, we have $v = X_i^1 \in F$.

Therefore $u \in F$ if and only if $v \notin F$. So Property 4 is true. □

Given an Ideal Elimination Set of G_{sse}, we can construct a solution to the Not-all-equal SAT Problem as follows.

Definition43. *Let F be an Ideal Elimination Set of G_{sse}. A corresponding solution $Sol(F)$ for the Not-all-equal SAT Problem is constructed as follows: $\forall 1 \leq i \leq n$, the Boolean variable $x_i = 1$ (namely, x_i is true) if and only if $X_i^1 \in F$.*

Clearly, in the above solution $Sol(F)$, a Boolean variable $x_i = 0$ (namely, x_i is false) if and only if $X_i^0 \in F$.

Lemma44. *Let F be an Ideal Elimination Set of G_{sse}, and let $Sol(F)$ be its corresponding solution to the Not-all-equal SAT Problem. Then for $1 \leq j \leq k$ and $1 \leq t \leq 3$, the t-th literal in the clause C_j is*

$$true$$

if and only if

$$A_j^t \notin F.$$

Proof. Let l_j^t denote the t-th literal in the clause C_j. Consider two cases:

- Case 1: l_j^t is x_i for some $1 \leq i \leq n$. In this case, by the construction of G_{sse}, A_j^t is connected to X_i^1 by an H bar. l_j^t is true if and only if $X_i^1 \in F$, which – by Property 4 of Lemma 42 – happens if and only if $A_j^t \notin F$.

- Case 2: l_j^t is $\bar{x}_i$ for some $1 \leq i \leq n$. In this case, by the construction of G_{sse}, A_j^t is connected to X_i^0 by an H bar. l_j^t is true if and only if x_i is false, which happens if and only if $X_i^0 \in F$, which – by Property 4 of Lemma 42 – happens if and only if $A_j^t \notin F$.

So in both cases, the conclusion holds. □

Lemma 45. *If F is an Ideal Elimination Set of G_{sse}, then $Sol(F)$ is a satisfying solution to the Not-all-equal SAT Problem.*

Proof. For $1 \leq j \leq k$, let $A_j^{t_1} \in F$ and $A_j^{t_2} \notin F$. By Property 3 of Lemma 42, such two integers $t_1, t_2 \in \{1, 2, 3\}$ exist. Consider the clause C_j. By Lemma 44, the t_1-th literal of C_j is false, and t_2-th literal of C_j is true. So for the Not-all-equal SAT Problem, every clause has at least one true literal and at least one false literal. So $Sol(F)$ is a satisfying solution to the Not-all-equal SAT Problem. □

The above lemma is useful for the scenario where G_{sse} has a One-Iteration Elimination Set of $n + 3k$ nodes. We now consider another possible scenario: the Not-all-equal SAT Problem is satisfiable.

Given a satisfying solution to the Not-all-equal SAT Problem, we can construct an Ideal Elimination Set of G_{sse}. We first define the corresponding set.

Definition 46. *Let π be a satisfying solution to the Not-all-equal SAT Problem; that is, with the solution π, every clause has at least one true literal and at least one false literal. A corresponding set of nodes, $\mathcal{F}(\pi)$, in G_{sse} is constructed as follows:*

- *For $i = 1, 2, \cdots, n$, if $x_i = 1$ in the solution π, then $X_i^1 \in \mathcal{F}(\pi)$ and $X_i^0 \notin \mathcal{F}(\pi)$; otherwise, $X_i^1 \notin \mathcal{F}(\pi)$ and $X_i^0 \in \mathcal{F}(\pi)$.*

- *For $j = 1, 2, \cdots, k$ and $t = 1, 2, 3$, if the t-th literal of clause C_j is true given the solution π, then $A_j^t \notin \mathcal{F}(\pi)$ and $B_j^t \in \mathcal{F}(\pi)$; otherwise, $A_j^t \in \mathcal{F}(\pi)$ and $B_j^t \notin \mathcal{F}(\pi)$.*

Lemma 47. *Let π be a satisfying solution to the Not-all-equal SAT Problem. Then $\mathcal{F}(\pi)$ is an Ideal Elimination Set of G_{sse}.*

Proof. Let us first show that $\mathcal{F}(\pi)$ is an One-Iteration Elimination Set of G_{sse}. Consider nodes in the following gadgets of G_{sse}:

- Consider the gadget V_i, for $1 \leq i \leq n$. (See Fig. 5.7 (a).) By the construction of $\mathcal{F}(\pi)$, either X_i^1 or X_i^0 is in $\mathcal{F}(\pi)$. Either way, after the node in $\mathcal{F}(\pi)$ is removed, the other two variable nodes in V_i will have neighboring check nodes of degree 1.

- Consider the gadget W_j, for $1 \leq j \leq k$. (See Fig. 5.7 (c).) Since the clause C_j has at least one true literal and at least one false literal, WLOG, let us suppose that its 1st and 2nd literals are true, and its 3rd literal is false. (There are totally 6 cases. And the other 5 cases can be proved similarly by symmetry.) By the construction of $\mathcal{F}(\pi)$, we have $A_j^1 \notin \mathcal{F}(\pi)$, $B_j^1 \in \mathcal{F}(\pi)$, $A_j^2 \notin \mathcal{F}(\pi)$, $B_j^2 \in \mathcal{F}(\pi)$, $A_j^3 \in \mathcal{F}(\pi)$, $B_j^3 \notin \mathcal{F}(\pi)$. It is not hard to see that after B_j^1, B_j^2 and B_j^3 are removed, the remaining variable nodes in W_j will all have neighboring check nodes of degree 1: removing A_j^3 will cause that effect for nodes in U_j^1, removing B_j^1 (and B_j^2) will cause that effect for nodes in U_j^2, and removing all those three nodes will cause that effect for $d_j^1, d_j^2, \cdots, d_j^6$.

- Consider an H bar. (See Fig. 5.7 (f) and (g).) Let u and v be the two endpoints of the H bar, where u is A_j^t or B_j^t, and v is X_i^1 or X_i^0 (for some parameters i, j, t). There are four possible cases:

  - Case 1: u is A_j^t and v is X_i^1. In this case, by the construction of G_{sse}, the t-the literal of clause C_j is x_i. By the construction of $\mathcal{F}(\pi)$, if $x_i = 1$, then $X_i^1 \in \mathcal{F}(\pi)$ and $A_j^t \notin \mathcal{F}(\pi)$; otherwise, $X_i^1 \notin \mathcal{F}(\pi)$ and $A_j^t \in \mathcal{F}(\pi)$.
  - Case 2: u is A_j^t and v is X_i^0. In this case, by the construction of G_{sse}, the t-the literal of clause C_j is $\bar{x}_i$. By the construction of $\mathcal{F}(\pi)$, if $x_i = 1$, then $X_i^0 \notin \mathcal{F}(\pi)$ and $A_j^t \in \mathcal{F}(\pi)$; otherwise, $X_i^0 \in \mathcal{F}(\pi)$ and $A_j^t \notin \mathcal{F}(\pi)$.
  - Case 3: u is B_j^t and v is X_i^1. In this case, by the construction of G_{sse}, the t-the literal of clause C_j is $\bar{x}_i$. By the construction of $\mathcal{F}(\pi)$, if $x_i = 1$, then $X_i^1 \in \mathcal{F}(\pi)$ and $B_j^t \notin \mathcal{F}(\pi)$; otherwise, $X_i^1 \notin \mathcal{F}(\pi)$ and $B_j^t \in \mathcal{F}(\pi)$.

– Case 4: u is B_j^t and v is X_i^0. In this case, by the construction of G_{sse}, the t-the literal of clause C_j is x_i. By the construction of $\mathcal{F}(\pi)$, if $x_i = 1$, then $X_i^0 \notin \mathcal{F}(\pi)$ and $B_j^t \in \mathcal{F}(\pi)$; otherwise, $X_i^0 \in \mathcal{F}(\pi)$ and $B_j^t \notin \mathcal{F}(\pi)$.

So in all four cases, either u or v is in $\mathcal{F}(\pi)$. If $u \in \mathcal{F}(\pi)$, then after u is removed from G_{sse}, both v and the variable node in the H bar (p_j^t or q_j^t) will have a neighboring check node of degree 1; and the same holds if $v \in \mathcal{F}(\pi)$.

So after nodes in $\mathcal{F}(\pi)$ are removed, the remaining variable nodes in G_{sse} will all have neighboring check nodes of degree 1. So $\mathcal{F}(\pi)$ is an One-Iteration Elimination Set of G_{sse}. Then it can be seen that all the nodes in $\mathcal{F}(\pi)$ are Interface Nodes (namely, $\mathcal{F}(\pi) \subset I_{sse}$) and $|\mathcal{F}(\pi)| = n+3k$. So $\mathcal{F}(\pi)$ is an Ideal Elimination Set of G_{sse}. □

We now prove the NP-hardness of the SSE_1 Problem.

Theorem 48. *The SSE_1 Problem is NP-hard.*

Proof. The SSE_1 Problem is an optimization Problem. Consider its decision problem: "Given a Stopping Graph G and an integer t, is it possible to remove t variable nodes from G so that the BP algorithm can decode the remaining variable nodes in one iteration?" Let it be called the "$SSE_1^{decision}$ Problem." Clearly, the problem is in NP. We will show now that there is a polynomial-time reduction from the NP-complete Not-all-equal SAT Problem to the $SSE_1^{decision}$ Problem.

Corresponding to the Not-all-equal SAT Problem, it has been introduced how to construct a bipartite graph G_{sse}. Let G be G_{sse}, and let t be $n + 3k$. Then we have a mapping from the Not-all-equal SAT Problem to the $SSE_1^{decision}$ Problem. It is not difficult to see that the mapping tokes polynomial time. We now need to prove that the Not-all-equal SAT Problem is satisfied if and only if the corresponding $SSE_1^{decision}$ Problem has a positive answer:

1. If the Not-all-equal SAT Problem is satisfiable, let π be such a satisfying solution. By Lemma 47, $\mathcal{F}(\pi)$ is an Ideal Elimination Set of G_{sse}, which has size $n + 3k$. So the $SSE_1^{decision}$ Problem has a positive answer.

2. If the $SSE_1^{decision}$ Problem has a positive answer, then G_{sse} has an One-Iteration Elimination Set of size $n+3k$. By Lemma 36 and Corollary 40, G_{sse} has a Canonical Elimination Set F of size $n+3k$, which, by Definition 41, is also an Ideal Elimination Set. Then by Lemma 45, the Not-all-equal SAT Problem is satisfiable.

So there is a polynomial-time reduction from the Not-all-equal SAT Problem to the $SSE_1^{decision}$ Problem. So the $SSE_1^{decision}$ Problem is NP-complete, and the SSE_1 Problem is NP-hard. □

5.5 Approximation Algorithm for SSE_1 Problem

In this section, we present an approximation algorithm for the SSE_1 problem, for Stopping Graphs whose degrees of variable nodes and check nodes are upper bounded by d_v and d_c, respectively. Its approximation ratio is

$$d_v(d_c - 1).$$

(Clearly, the same result also applies to regular (d_v, d_c) LDPC codes and irregular codes with the same constraint on maximum degrees.) Note that the optimization objective is to minimize the size of the elimination set (namely, the number of removed variable nodes). So the approximation ratio means the maximum ratio of the size of an elimination set produced by the approximation algorithm to the size of an optimal (i.e., minimum) elimination set.

Definition 49. *In the Stopping Graph $G = (V \cup C, E)$, $\forall\ v \in V$, define its "variable-node neighborhood" as $\Lambda(v) \triangleq$*

$$\{u \in V - \{v\} \mid \exists\ c \in C \text{ such that } (u, c) \in E \text{ and } (v, c) \in E\}.$$

That is, every variable node in $\Lambda(v)$ shares a common neighboring check node with v.

Example 50. *For the Stopping Graph in Fig. 5.9, we have $\Lambda(v_1) = \{v_4, v_6\}$, $\Lambda(v_2) = \{v_3, v_8\}$, $\Lambda(v_3) = \{v_2, v_5, v_8\}$, and so on.* □

We now introduce an approximation algorithm. The algorithm will assign three colors to variable nodes:

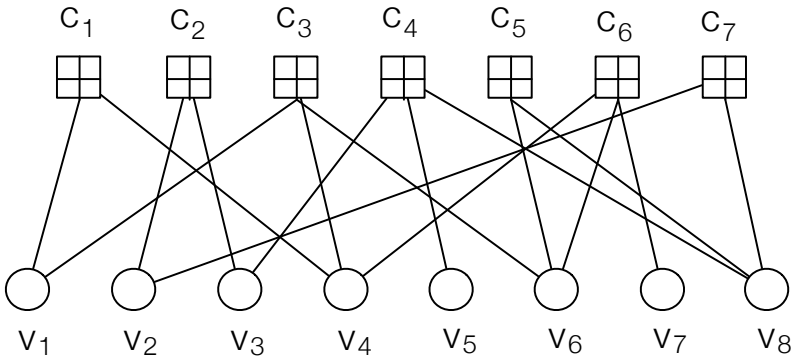

Figure 5.9: A Stopping Graph $G = (V \cup C,\ E)$.

- Initially, every variable node is of the color *white*. It means that this variable node cannot be decoded by one iteration of BP-decoding yet.
- As the algorithm proceeds, if a variable node' color turns *black*, it means the algorithm has included it in the Elimination Set (namely, the algorithm has removed it).
- As the algorithm proceeds, if a variable node's color turns *gray*, it means the variable node is not yet removed, but it will be decodable after one iteration of BP decoding.

When the algorithm ends, every variable node's color will be either *black* or *gray*.

The algorithm works as follows. It takes a greedy approach, and updates the colors of variable nodes iteratively. In each iteration, it identifies an arbitrary *white* variable node (say it is node v), and does the following:

1. Step One: Let U_v denote the set of variable nodes in $\Lambda(v)$ that are currently *white* or *gray*. Turn the colors of the nodes in U_v to *black*, and turn the color of v to *gray*.
2. Step Two: For every check node c that is connected to at least one variable node in U_v, check if exactly one of c's neighboring variable node is *white* and all c's other neighboring variable nodes are *black*. If so, turn that neighboring variable node's color from *white* to *gray*.

The algorithm keeps iterating as above until all variable nodes are either *black* or *gray*. Then it returns the set of *black* variable nodes as the Elimination Set. It is not difficult to see that the remaining variable nodes are decodable by BP in one iteration: by the algorithm, every time a variable node is turned from *white* to *gray*, it has at least one neighboring check node c such that c' other neighboring variable nodes are all *black* (namely, removed), and an BP iteration using the check node c will help decode that *gray* variable node.

The algorithm is formally presented below. Note that it uses two sets, S_{white} and S_{black}, to keep track of the white and black nodes, respectively.

Algorithm 51. *Approximation Algorithm for* SSE_1

Input: Stopping Graph $G = (V \cup C, E)$.

Output: A one-iteration elimination set.

Algorithm:

1. $S_{white} \leftarrow V$, $S_{black} \leftarrow \emptyset$.
2. **for** $v \in V$
3. $color(v) \leftarrow white$.
4. **while** $S_{white} \neq \emptyset$
5. *{*
6. *Pick an arbitrary node* v *from* S_{white}.
7. $U_v \leftarrow \emptyset$.
8. **for** $u \in \Lambda(v)$
9. *{*
10. **if** $color(u) = white$

11. *{*

12. $U_v \leftarrow U_v \cup \{u\}$.

13. $color(u) \leftarrow black$.

14. $S_{black} \leftarrow S_{black} \cup \{u\}$.

15. $S_{white} \leftarrow S_{white} - \{u\}$.

16. *}*

17. **else if** $color(u) = gray$

18. *{*

19. $U_v \leftarrow U_v \cup \{u\}$.

20. $color(u) \leftarrow black$.

21. $S_{black} \leftarrow S_{black} \cup \{u\}$.

22. *}*

23. *}*

24. $S_{white} \leftarrow S_{white} - \{v\}$, $color(v) \leftarrow gray$.

25. **for** $u \in U_v$

26. **for** *every check node* c *adjacent to* u

27. **if** *exactly one of* c*'s neighboring nodes is white*

and all c*'s other neighboring nodes are black*

28. *{*

29. *Let* w *be that white neighboring node.*

30. $S_{white} \leftarrow S_{white} - \{w\}$, $color(w) \leftarrow gray$.

31. }

32. }

33. **return** S_{black}.

We show an example of the above algorithm.

Example 52. *Let the graph G be as shown in Fig. 5.10 (a). The algorithm first identifies a white node v_1, turns nodes in $U_{v_1} = \{v_2, v_3\}$ black, then turns v_1 and v_4 gray (see Fig. 5.10 (b)). Next, it identifies a white node v_5, turns nodes in $U_{v_5} = \{v_6, v_7\}$ black, then turns v_5 gray (see Fig. 5.10 (c)). Next, it identifies a white node v_8, turns the nodes in $U_{v_8} = \{v_9\}$ black, then turns v_8 gray (see Fig. 5.10 (d)). So the Elimination Set is $S = \{v_2, v_3, v_6, v_7, v_9\}$. If we delete nodes in S from G, we get the subgraph in Fig. 5.10 (e), where we can see that all the remaining nodes are gray and can be decoded by one BP iteration.*

It is not hard to verify that every one-iteration elimination set for G has size no less than 5. Since $|S| = 5$, the output of the algorithm is actually optimal in this example. □

We analyze the time complexity of the above algorithm. Let d_v and d_c denote the maximum degrees of variable nodes and check nodes in the Stopping Graph G, respectively. The algorithm has time complexity $O(d_v^2 d_c^2 |V|)$ because it identifies up to $O(|V|)$ white variable nodes and for each such node v, it checks its neighboring check nodes and nodes in $\Lambda(v)$, check nodes adjacent to nodes in $\Lambda(v)$, and variable nodes that share common neighboring check nodes with any node in $\Lambda(v)$; and there are $O(d_v^2 d_c^2)$ such nodes for each v.

We now analyze the approximation ratio of the algorithm. We first introduce a few lemmas.

Lemma 53. *Let $G_1 = (V_1 \cup C_1,\ E_1)$ and $G_2 = (V_2 \cup C_2,\ E_2)$ be two Stopping Graphs. Let s_1 and s_2 denote the size of the minimum one-iteration elimination set in G_1 and G_2, respectively. If G_2 can be obtained from G_1 by removing some variable nodes and their incident edges, then*

$$s_2 \leq s_1.$$

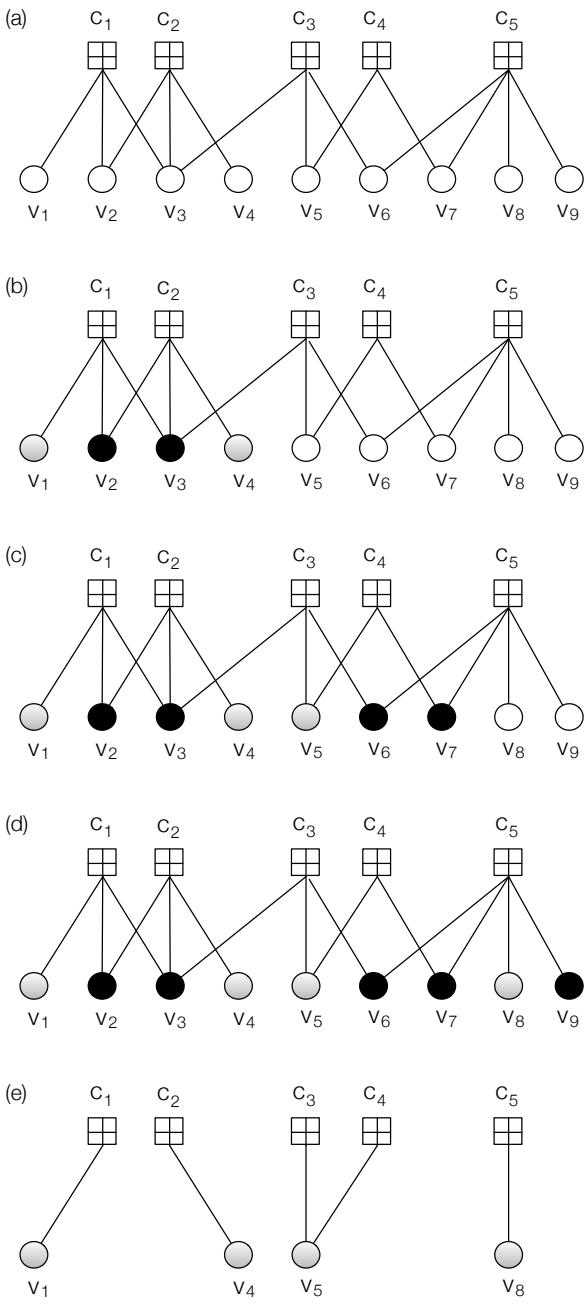

Figure 5.10: An example of the approximation algorithm for SSE_1.

Proof. Removing variable nodes is the same as knowing the values of those erased codeword bits, which only helps BP decoding. □

Say that the algorithm identifies a sequence of white variable nodes

$$\hat{v}_1, \hat{v}_2, \cdots, \hat{v}_t$$

in the Stopping Graph $G = (V \cup C, E)$, and turns the variable nodes in

$$U_{\hat{v}_1}, U_{\hat{v}_2}, \cdots, U_{\hat{v}_t}$$

black. Let us define a sequence of subgraphs

$$G_0, G_1, \cdots, G_t$$

accordingly.

Definition 54.. *Let $G_0 = G$. For $i = 1, 2, \cdots, t$, let G_i be obtained from G_{i-1} by removing the nodes in*

$$U_{\hat{v}_i} \cup \{\hat{v}_i\} \cup \{\textit{check nodes adjacent to } \hat{v}_i\}$$

and their incident edges.

Note that for $i = 1, 2, \cdots, t$, in the i-th iteration, the algorithm removes only the variable nodes in $U_{\hat{v}_i}$ (namely, turning them black) from the subgraph G_{i-1}, not $\hat{v}_i$ or its adjacent check nodes. (It turns $\hat{v}_i$ to gray.) However, once $U_{\hat{v}_i}$ is removed, all the nodes in $\Lambda(\hat{v}_i)$ are removed, so $\hat{v}_i$ and its adjacent check nodes become disconnected from the rest of the graph (which is G_i). Therefore it becomes sufficient to consider the SSE_1 Problem for G_i in the next iteration, and it can be seen that

$$\hat{v}_{i+1}, U_{\hat{v}_{i+1}}, \{\text{check nodes adjacent to } \hat{v}_{i+1}\},$$

$$\hat{v}_{i+2}, U_{\hat{v}_{i+2}}, \{\text{check nodes adjacent to } \hat{v}_{i+2}\},$$

$$\cdots,$$

$$\hat{v}_t, U_{\hat{v}_t}, \{\text{check nodes adjacent to } \hat{v}_t\}$$

are all nodes (or sets of nodes) in G_i.

Lemma 55. *For $i = 0, 1, \cdots, t-1$, every one-iteration elimination set for G_i contains at least one variable node in*

$$U_{\hat{v}_{i+1}} \cup \{\hat{v}_{i+1}\}.$$

Proof. Consider the graph G_i and its variable node $\hat{v}_{i+1}$. To make the remaining variable nodes decodable by one iteration of BP decoding after some variable nodes are removed, it is *necessary* (although not sufficient) that $\hat{v}_{i+1}$ is either removed, or decodable after one such iteration of BP decoding; and that requires one of these two conditions to be true:

1. $\hat{v}_{i+1}$ is removed.

2. $\hat{v}_{i+1}$ is not removed, but it has a neighboring check node c such that all c's other neighboring variable nodes in G_i are removed. (The check node c will help decode $\hat{v}_{i+1}$ in one iteration of BP decoding. And note that those "other neighboring variable nodes" of c are all nodes in $U_{\hat{v}_{i+1}}$. Also note that since v_{i+1} is turned from white to gray in the $(i+1)$-th iteration of the algorithm, at the beginning of the $(i+1)$-th iteration, every check node adjacent to v_{i+1} must have degree at least two in G_i. So the set of those "other neighboring variable nodes" of c cannot be empty.)

So it is necessary that at least one variable node in $U_{\hat{v}_{i+1}} \cup \{\hat{v}_{i+1}\}$ is removed. □

Lemma 56. *For $i = 0, 1, \cdots, t$, let α_i denote the minimum size of a one-iteration elimination set for G_i. Then*

$$\alpha_i \geq t - i.$$

Proof. The proof is by induction, but in the reverse order for i (i.e, from $i = t, t-1 \cdots$ down to

0). When $i = t$, clearly $\alpha_i \geq t - i = 0$, so the conclusion holds for the base case. Now assume that the conclusion holds for $\alpha_t, \alpha_{t-1}, \cdots, \alpha_{i+1}$, and consider the case for α_i.

Consider an optimal (i.e., minimum-sized) one-iteration elimination set S for G_i. Define $Y \triangleq S \cap (U_{\hat{v}_{i+1}} \cup \{\hat{v}_{i+1}\})$. By Lemma 55, S removes at least one variable node in $U_{\hat{v}_{i+1}} \cup \{\hat{v}_{i+1}\}$, so $|Y| \geq 1$. Let $\tilde{G}$ be the bipartite graph obtained by removing the variable nodes in Y from G_i (and their incident edges), and let $\tilde{\alpha}$ denote the minimum size of a one-iteration elimination set for $\tilde{G}$. Then $\alpha_i = |S| = |Y| + \tilde{\alpha} \geq \tilde{\alpha} + 1$.

G_{i+1} is obtained from G_i by removing the variable nodes in $U_{\hat{v}_{i+1}} \cup \{\hat{v}_{i+1}\}$, which is a superset of Y. So G_{i+1} can also be obtained from $\tilde{G}$ by removing the variable nodes in $(U_{\hat{v}_{i+1}} \cup \{\hat{v}_{i+1}\}) - Y$. So by Lemma 53, $\alpha_{i+1} \leq \tilde{\alpha}$. By the induction assumption, we get $\alpha_{i+1} \geq t-(i+1)$. By combining the above results, we get $\alpha_i \geq \tilde{\alpha} + 1 \geq \alpha_{i+1} + 1 \geq t - (i+1) + 1 = t - i$. □

Theorem 57. *Let d_v and d_c denote the maximum degrees of variable nodes and check nodes, respectively, in the Stopping Graph $G = (V \cup C, E)$. Then the above algorithm has an approximation ratio of*

$$d_v(d_c - 1).$$

Proof. By setting $i = 0$ in Lemma 56, we get $\alpha_0 \geq t$, namely, any one-iteration elimination set for G removes at least t variable nodes. The algorithm removes the nodes in

$$U_{\hat{v}_1} \cup U_{\hat{v}_2} \cup \cdots \cup U_{\hat{v}_t},$$

whose size is

$$|\bigcup_{i=1}^{t} U_{\hat{v}_i}| = \sum_{i=1}^{t} |U_{\hat{v}_i}| \leq \sum_{i=1}^{t} |\Lambda(\hat{v}_i)| \leq t \cdot d_v(d_c - 1).$$

So the approximation ratio is at most $d_v(d_c - 1)$. □

5.6 Analysis and Algorithms for SSE_k Problems

In this section, we present more analysis and algorithms for SSE_k Problems, including $k = \infty$. We first analyze how an important factor, RBER (raw bit-erasure rate), affects the performance of

approximation algorithms, and show that for high-rate codes with high actual erasure rates, all algorithms have good approximation ratios. We then present exact algorithms for SSE_∞ and SSE_k Problems when the Stopping Graph is a tree (or a forest). The algorithms output optimal solutions and have linear time complexity.

5.6.1 Effect of RBER for Approximation Algorithms

We first analyze the effect of RBER for approximation algorithms. Consider an (N, K) LDPC code with N codeword bits and K information bits (where $K < N$), whose code rate is $R \triangleq K/N$. Let $G = (V \cup C, E)$ be its Stopping Graph, where V is the Stopping Set. As shown in Fig. 5.2 (b), the higher RBER is, the greater $|V|$ becomes on average. Let $\epsilon \triangleq |V|/N$ be called the *actual erasure rate relative to stopping set* V.

Lemma 58. *Let $S \subseteq V$ be any solution (i.e., an Elimination Set) to the SSE_k Problem. If $|V| \geq N - K$, then*

$$|S| \geq |V| - N + K.$$

Proof. The proof is by contradiction. If $|S| < |V| - N + K$, then after the erased bits in the Elimination Set are decoded by the NR-Decoder, the total number of codeword bits with known values is $(N - |V|) + |S| < (N - |V|) + (|V| - N + K) = K$. Then the ECC-Decoder will not be able to recover the K bits of information in the codeword. □

Theorem 59. *For the SSE_k Problem, if $\epsilon > 1 - R$, then the approximation ratio of any algorithm is at most*

$$\frac{\epsilon}{\epsilon - (1 - R)}.$$

Proof. Let S^* and S be an optimal solution and the solution of an arbitrary algorithm, respectively, to the SSE_k Problem. If $\epsilon > 1 - R$, then $|V| = \epsilon N > (1 - K/N)N = N - K$. By Lemma 58, $|S^*| \geq |V| - N + K$. Since $S \subseteq V$, we get $\frac{|S|}{|S^*|} \leq \frac{|V|}{|V|-N+K} = \frac{\epsilon}{\epsilon-1+R}$. □

So for high rate codes (where R approaches 1), if the RBER is high (which approaches 1), then with high probability, ϵ also approaches 1. In this case, $\frac{\epsilon}{\epsilon-(1-R)}$ approaches 1, so all algorithms

have good approximation ratios.

5.6.2 Exact Algorithm for SSE_∞ Problem with Stopping Tree

The Stopping Graph $G = (V \cup C, E)$ can be a tree, especially when the RBER is low. In this case, we call G a *Stopping Tree*. Note that if G is a forest, the SSE_k Problem can be solved for each of its tree components independently.

In this subsection, we present an efficient and exact algorithm for the SSE_∞ Problem. The algorithm will be extended to the SSE_k Problem for general k subsequently.

Given a Stopping Tree $G = (V \cup C, E)$, we can pick an arbitrary variable node $v \in V$ as the root, run Breadth-First Search (BFS) on G starting with v, and label the nodes of G by v_1, v_2, $\cdots$, $v_{|V|+|C|}$ based on their order of discovery in the BFS. (Note that the root node v is labelled by v_1, and siblings nodes in the BFS tree always have consecutive labels.) We denote the resulting BFS tree by G_{BFS}.

The algorithm for SSE_∞ is as follows.

Algorithm 60. *Exact Algorithm for SSE_∞*

Input: Stopping Tree $G = (V \cup C, E)$.

Output: An Elimination Set of minimum size in G.

Algorithm:

1. *Generate G_{BFS} by running BFS on G.*
2. *Let S be an empty set.*
3. $i \leftarrow |V| + |C|$.
4. **while** $i \geq 1$
5. *{*
6. **if** $i > 1$ *and* $v_i \in V$
7. *{*

8. *Let j be the minimum index such that v_j is a sibling*

9. *of v_i in the BFS tree G_{BFS} rooted at v_1.*

10. **if** $j < i$

11. *Add $v_j, v_{j+1}, \cdots, v_{i-1}$ to set S.*

12. $i \leftarrow j - 1$.

13. *}*

14. **else if** $i = 1$

15. *{*

16. *Add v_1 to set S.*

17. $i = 0$.

18. *}*

19. **else**

20. $i \leftarrow i - 1$.

21. *}*

22. **Return** S.

The algorithm first runs BFS to generate G_{BFS}. It then processes the nodes in the reverse order of their labels (from $v_{|V|+|C|}$ to v_1). Every time it comes to a node v_i, if v_i is a variable node and has siblings (of smaller labels), its siblings are included in the Elimination Set. The root v_1 is also included in the Elimination Set. The following is an example of the algorithm.

Example 61. *A Stopping Tree and its BFS tree are shown in Fig. 5.11 (a) and (b), respectively. (Note that the node labels v_1, v_2, $\cdots$, v_{17} in Fig. 5.11 (a) are not known* a priori*; instead, they are obtained after we run BFS on the graph with v_1 as its root.) Then algorithm then processes the nodes in the reverse order of their labels. (Note that when it comes to a check node, no action is taken.) It first comes to v_{17}, and includes its sibling v_{16} in the Elimination Set. (See Fig. 5.11 (c).) It then comes to v_{15}, and includes its siblings v_{14} and v_{13} in the Elimination Set. (See Fig. 5.11 (d).) It then comes to v_{12}, and takes no action since v_{12} has no sibling. (See Fig. 5.11 (e).) It then comes to v_{11}, v_{10} and v_9 sequentially and takes no action since they are check nodes. It then comes to v_8, and includes its sibling v_7 in the Elimination Set. (See Fig. 5.11 (f).) It then comes to v_6, v_5, $\cdots$, v_2 sequentially and takes no action since they either have no sibling or are check nodes. Finally, it comes to the root v_1 and includes it in the Elimination Set. (See Fig. 5.11 (i).) The Elimination Set returned by the algorithm is $\{v_1, v_7, v_{13}, v_{14}, v_{16}\}$.* $\square$

We now show that the algorithm returns an optimal (i.e., minimum-sized) Elimination Set. We suppose $|V| \geq 2$. (The case of $|V| = 1$ is trivial.)

Lemma 62. *In G_{BFS}, let B denote the set of sibling nodes of $v_{|V|+|C|}$. (B could be empty.) Then any Elimination Set for G_{BFS} contains at least $|B|$ nodes in $B \cup \{v_{|V|+|C|}\}$.*

Proof. If B is an empty set, the conclusion is clearly true. So now assume B is not empty. Let c be the parent of $v_{|V|+|C|}$ (and also all the nodes in B) in G_{BFS}, which is a check node. Let w be the parent of c, which is a variable node. We can see that c is adjacent to $|B| + 2$ variable nodes in total.

Since $v_{|V|+|C|}$ has the greatest label among all nodes, by the property of BFS, it has the greatest distance to the root among all nodes; and so do its siblings. So all nodes in $B \cup \{v_{|V|+|C|}\}$ are leaves in G_{BFS}.

We now prove the lemma by contradiction. If fewer than $|B|$ nodes in $B \cup \{v_{|V|+|C|}\}$ are included in an Elimination Set, then at least two leaves in $B \cup \{v_{|V|+|C|}\}$ – say v_i and v_j – are not in the Elimination Set. Since c is the only check node adjacent to them, even if other nodes of the tree

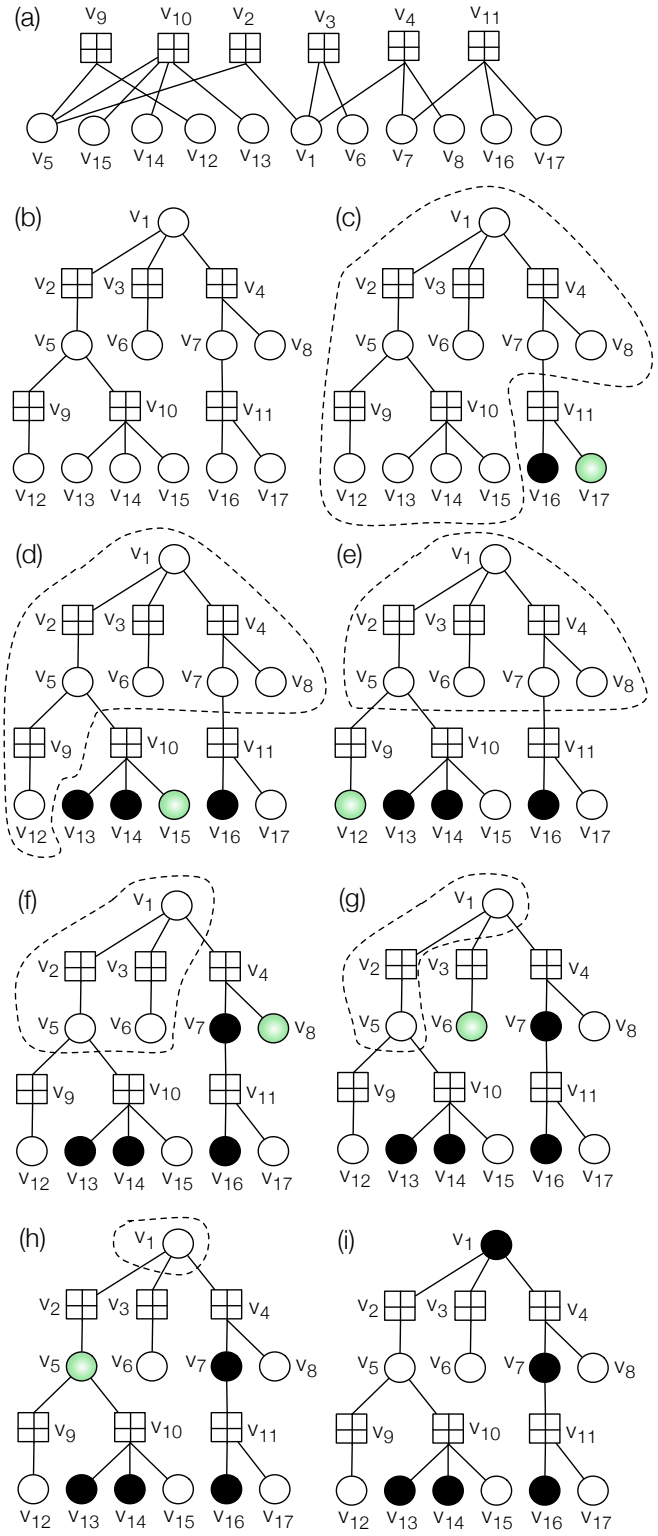

Figure 5.11: Algorithm for SSE_∞ on a Stopping Tree. (a) A Stopping Tree $G = (V \cup C,\ E)$. (b) Its BFS (Breadth-First Search) tree G_{BFS}. (c) Process v_{17}. (d) Process v_{15}. (e) Process v_{12}. (f) Process v_8. (g) Process v_6. (h) Process v_5. (i) Process v_1.

are all in the Elimination Set, v_i and v_j still cannot be decoded. And that contradicts the definition of Elimination Set. So the conclusion is true. □

For any non-root node v in G_{BFS}, let $\pi(v)$ denote its parent. Let

$$G_{sub}$$

denote the subtree of G_{BFS} obtained this way: if we remove the subtree rooted at $\pi(v_{|V|+|C|})$ from G_{BFS}, the remaining subgraph is G_{sub}. (For instance, in Fig. 5.11, we have $v_{|V|+|C|} = v_{17}$, $\pi(v_{17}) = v_{11}$, and G_{sub} is the subtree in the dashed circle in Fig. 5.11 (c).)

Lemma 63. *Let S^* be a minimum-sized Elimination Set of G_{sub}, and let B denote the set of sibling nodes of $v_{|V|+|C|}$. Then $S^* \cup B$ is a minimum-sized Elimination Set of G_{BFS}.*

Proof. Let us call an Elimination Set *normalized* if it includes all the nodes in B but does not include $v_{|V|+|C|}$. By Lemma 62, an Elimination Set of G_{BFS} contains either $|B|$ or $|B|+1$ nodes of $B \cup \{v_{|V|+|C|}\}$. If it is the latter case, by replacing $\{v_{|V|+|C|}\}$ by $\pi(\pi(v_{|V|+|C|}))$ in the Elimination Set (which makes all the variable nodes adjacent to $\pi(v_{|V|+|C|})$ except $v_{|V|+|C|}$ included in the Elimination Set), we can get another Elimination Set of no greater size. Therefore, there exists a minimum-sized Elimination Set that is normalized.

Consider a normalized Elimination Set $\tilde{S} \subseteq V$. Since $B \subseteq \tilde{S}$ and $v_{|V|+|C|} \notin \tilde{S}$, $\tilde{S} - B$ must be an Elimination Set of G_{sub}. On the other hand, given an Elimination Set $\hat{S}$ of G_{sub}, $\hat{S} \cup B$ must be a normalized Elimination Set of G_{BFS}. (The BP decoding algorithm will first decode all the variable nodes in G_{sub}, then use the check node $\pi(v_{|V|+|C|})$ to decode $v_{|V|+|C|}$.) Since S^* is a minimum-sized Elimination Set of G_{sub}, $S^* \cup B$, as a normalized Elimination Set for G_{BFS}, is minimum-sized. Since there exists a minimum-sized Elimination Set that is normalized, $S^* \cup B$, as an Elimination Set for G_{BFS} (without the restriction of being normalized), is also minimum-sized. □

Theorem 64. *Algorithm 60 returns an optimal (i.e., minimum-sized) Elimination Set of $G = (V \cup C, E)$.*

Proof. By Lemma 63, the problem of finding an optimal Elimination Set for G_{BFS} (which is the same as G) can be reduced to the problem of finding an optimal Elimination Set for its subtree G_{sub}. Algorithm 60 uses that technique repeatedly to reduce the problem to smaller and smaller subtrees, until it comes to the final case where the subtree contains only the root node v_1 (whose optimal Elimination Set is simply $\{v_1\}$). (In Fig. 5.11, such a sequence of shrinking subtrees are shown in dashed circles from (c) to (h).) That leads to the conclusion. □

Therefore Algorithm 60 is an exact algorithm for the SSE_∞ Problem. Its time complexity is $O(|V| + |C|)$.

5.6.3 Exact Algorithm for SSE_k Problem with Stopping Tree

We now extend the previous analysis, and design an exact algorithm for the SSE_k Problem of linear time complexity.

The algorithm first runs BFS on G to get the tree G_{BFS} that labels nodes by $v_1, v_2, \cdots, v_{|V|+|C|}$, where v_1 is the root. Then (similar to the algorithm for SSE_∞), it processes the nodes in the reverse order of their labels, and keeps reducing the SSE_k Problem – actually, a more general form of the SSE_k Problem, which shall be called the $gSSE_k$ Problem – to smaller and smaller subtrees. Let us now define this $gSSE_k$ Problem.

Definition 65 ($gSSE_k$ Problem)**.** *Let $G = (V \cup C, E)$ be a Stopping Graph. and let k be a non-negative integer. Every variable node $v \in V$ is associated with two parameters*

$$\delta(v) \in \{1, 2, \cdots, k, \infty\}$$

and

$$\omega(v) \in \{0, 1, \cdots, k, \infty\}$$

satisfying the condition that either $\delta(v) = \infty$ or $\omega(v) = \infty$, but not both; and when the BP decoder runs on G, v's value can be recovered (namely, v can become a non-erasure) by the end of the $\delta(v)$-th iteration automatically (namely, without any help from neighboring check nodes).

Then, how to remove the minimum number of variable nodes from V such that for every remaining variable node v with $\omega(v) \leq k$, it can be corrected by the BP decoder in no more than $\omega(v)$ iterations? (By default, if $\omega(v) = 0$, v has to be removed from V because the BP decoder starts with the 1st iteration.)

A solution to the $gSSE_k$ Problem (namely, the set of removed nodes) is called a *g-Elimination Set*. We see that if $\delta(v) = \infty$ and $\omega(v) = k$ for every $v \in V$, then the $gSSE_k$ Problem is identical to the SSE_k Problem.

In G_{BFS}, let $\tau \in \{1, 2, \cdots, |V| + |C|\}$ denote the minimum integer such that v_τ either is a sibling of $v_{|V|+|C|}$ or is $v_{|V|+|C|}$ itself. (So $v_\tau, v_{\tau+1}, \cdots, v_{|V|+|C|}$ are siblings.) Define

$$\mathcal{P} \triangleq \{i \mid \tau \leq i \leq |V| + |C|, \omega(v_i) \leq k\}$$

and

$$\mathcal{Q} \triangleq \{i \mid \tau \leq i \leq |V| + |C|, \delta(v_i) \leq k\}.$$

Since $\forall\ v \in V$, either $\delta(v)$ or $\omega(v)$ is ∞ but not both, $\mathcal{P}$ and $\mathcal{Q}$ form a partition of the set $\{\tau, \tau + 1, \cdots, |V| + |C|\}$.

By convention, for the empty set $\emptyset$, we say $\max_{i\in\emptyset} \delta(v_i) = \max_{i\in\emptyset} \omega(v_i) = 0$. We first make some observations.

Lemma 66. *Suppose $\max_{i\in\mathcal{P}} \omega(v_i) > \max_{i\in\mathcal{Q}} \delta(v_i)$. Let i^* be an integer in $\mathcal{P}$ such that $\omega(v_{i^*}) = \max_{i\in\mathcal{P}} \omega(v_i)$. Then there exists a minimum-sized g-Elimination Set for G_{BFS} that includes the nodes in $\{v_i | i \in \mathcal{P}, i \neq i^*\}$ but not v_{i^*}.*

Proof. Any g-Elimination Set for G_{BFS} has to include at least $|\mathcal{P}| - 1$ nodes in $\{v_i | i \in \mathcal{P}\}$ because otherwise the un-included nodes in $\{v_i | i \in \mathcal{P}\}$ will never be corrected. It is an optimal strategy to include the $|\mathcal{P}| - 1$ nodes in $\{v_i | i \in \mathcal{P}, i \neq i^*\}$ in the g-Elimination Set because their $\omega(v_i)$ values impose more restrictive requirements than $\omega(v_{i^*})$ does. Now let T be a g-Elimination Set for G_{BFS} that includes all the nodes in $\{v_i | i \in \mathcal{P}, i \neq i^*\}$. If $v_{i^*} \in T$, we can replace it

by $\pi(\pi(v_{|V|+|C|}))$ in T and get another g-Elimination Set T' for G_{BFS}, with $|T'| \leq |T|$ (since $\pi(\pi(v_{|V|+|C|}))$ may already be in T). (Note that with T', since $\max_{i\in\mathcal{P}} \omega(v_i) > \max_{i\in\mathcal{Q}} \delta(v_i)$, the check node $\pi(v_{|V|+|C|})$ can help correct v_{i^*} by iteration $\max_{i\in\mathcal{Q}} \delta(v_i)+1 \leq \max_{i\in\mathcal{P}} \omega(v_i) = \omega(v_{i^*})$; and since $\pi(\pi(v_{|V|+|C|})) \in T'$, the BP decoding in G_{sub} becomes independent of the subtree rooted at $\pi(v_{|V|+|C|})$.) So there exists a minimum-sized g-Elimination Set for G_{BFS} that includes the nodes in $\{v_i | i \in \mathcal{P}, i \neq i^*\}$ but not v_{i^*}. □

Lemma67. *Suppose* $\max_{i\in\mathcal{P}} \omega(v_i) \leq \max_{i\in\mathcal{Q}} \delta(v_i)$. *Then there exists a minimum-sized* g*-Elimination Set for* G_{BFS} *that contains all the nodes in* $\{v_i | i \in \mathcal{P}\}$.

Proof. If $\mathcal{P} = \emptyset$, the conclusion automatically holds. If $\mathcal{P} \neq \emptyset$ and $\max_{i\in\mathcal{P}} \omega(v_i) = 0$, any g-Elimination Set for G_{BFS} has to include $\{v_i | i \in \mathcal{P}\}$, so the conclusion also holds.

Now consider the case where $\max_{i\in\mathcal{P}} \omega(v_i) > 0$. Let T be a minimum-sized g-Elimination Set for G_{BFS}. T has to include at least $|\mathcal{P}| - 1$ nodes in $\{v_i | i \in \mathcal{P}\}$ because otherwise the un-included nodes in $\{v_i | i \in \mathcal{P}\}$ cannot be corrected (using the check node $\pi(v_{|V|+|C|})$). If $|T| = |\mathcal{P}| - 1$, the let $j \in \mathcal{P}$ be an integer such that $v_j \notin T$. If $T \cap \{v_i | i \in \mathcal{Q}\} = \emptyset$, then v_j cannot be corrected by iteration $\omega(v_j) \leq \max_{i\in\mathcal{P}} \omega(v_i) \leq \max_{i\in\mathcal{Q}} \delta(v_i)$ because not all nodes in $\{v_i | i \in \mathcal{Q}\}$ will be corrected by iteration $\omega(v_j) - 1$, so this is an impossible case. So $T \cap \{v_i | i \in \mathcal{Q}\} \neq \emptyset$. Let $m \in \mathcal{Q}$ be an integer such that $v_m \in T$; then we can replace v_m by v_j in T and get another g-Elimination Set T' for G_{BFS} because v_m helps decoding more than v_j: v_m can be corrected automatically. Since $|T'| = |T|$ and $\{v_i | i \in \mathcal{P}\} \subseteq T'$, the conclusion holds. □

The next two lemmas show how to reduce the $gSSE$ Problem from G_{BFS} to its subtree G_{sub}. In some cases, in the derived $gSSE$ Problem for G_{sub}, the values of $\delta(\pi(\pi(v_{|V|+|C|})))$ and $\omega(\pi(\pi(v_{|V|+|C|})))$ in G_{sub} may be different from their original values in G_{BFS}; and in such cases, to avoid confusion, we will denote the tree G_{sub} by $\hat{G}_{sub}$.

Lemma68. *Suppose* $\max_{i\in\mathcal{P}} \omega(v_i) \leq \max_{i\in\mathcal{Q}} \delta(v_i)$. *Consider five cases:*

1. *Case 1: If* $|\mathcal{Q}| > 0$ *and* $\max_{i\in\mathcal{Q}} \delta(v_i) = k$, *let* S *be a minimum-sized* g*-Elimination Set for* G_{sub}.

2. *Case 2: If* $|\mathcal{Q}| > 0$, $\max_{i\in\mathcal{Q}} \delta(v_i) < k$ *and* $\delta(\pi(\pi(v_{|V|+|C|}))) \leq k$, *let* S *be a minimum-sized* g*-Elimination Set for* $\hat{G}_{sub}$ *where* $\delta(\pi(\pi(v_{|V|+|C|})))$ *is changed to*

$$\min\{\delta(\pi(\pi(v_{|V|+|C|}))), \max_{i\in\mathcal{Q}} \delta(v_i) + 1\}.$$

3. *Case 3: If* $|\mathcal{Q}| > 0$ *and* $\omega(\pi(\pi(v_{|V|+|C|}))) \leq \max_{i\in\mathcal{Q}} \delta(v_i) < k$, *let* S *be a minimum-sized* g*-Elimination Set for* G_{sub}.

4. *Case 4: If* $|\mathcal{Q}| > 0$ *and* $\max_{i\in\mathcal{Q}} \delta(v_i) < \omega(\pi(\pi(v_{|V|+|C|}))) \leq k$, *let* S *be a minimum-sized* g*-Elimination Set for* $\hat{G}_{sub}$ *where* $\delta(\pi(\pi(v_{|V|+|C|})))$ *is changed to*

$$\max_{i\in\mathcal{Q}} \delta(v_i) + 1$$

and $\omega(\pi(\pi(v_{|V|+|C|})))$ *is changed to*

$$\infty.$$

5. *Case 5: If* $|\mathcal{Q}| = 0$, *there are two sub-cases: (1) if* $\omega(\pi(\pi(v_{|V|+|C|}))) = 0$, *let* S *be a minimum-sized* g*-Elimination Set for* G_{sub}*; (2) otherwise, let* S *be a minimum-sized* g*-Elimination Set for* $\hat{G}_{sub}$ *where* $\delta(\pi(\pi(v_{|V|+|C|})))$ *is changed to 1 and* $\omega(\pi(\pi(v_{|V|+|C|})))$ *is changed to* ∞.

Then $S \cup \{v_i | i \in \mathcal{P}\}$ *is a minimum-sized* g*-Elimination Set for* G_{BFS}.

Proof. By Lemma 67, there exists a minimum-sized g-Elimination Set for G_{BFS} that contains all the nodes in $\{v_i | i \in \mathcal{P}\}$. Now consider only minimum-sized g-Elimination Sets for G_{BFS} that contain all the nodes in $\{v_i | i \in \mathcal{P}\}$. See the nodes in $\{v_i | i \in \mathcal{P}\}$ as removed (because nodes in an Elimination Set are removed before decoding begins); then to prove the conclusion, we just need to prove this assertion: *when* $P = \emptyset$, S *is a minimum-sized* g*-Elimination Set for* G_{BFS} .

For Case 1, since $\max_{i\in\mathcal{Q}} \delta(v_i) = k$, the subtree rooted at $\pi(v_{|V|+|C|})$ cannot help correct the node $\pi(\pi(v_{|V|+|C|}))$ in the first k iterations. Every node v with $\omega(v) \neq \infty$ is in G_{sub} and has

$\omega(v) \leq k$. So finding a minimum-sized g-Elimination Set for G_{BFS} is equivalent to finding such as set for G_{sub}. So the assertion holds.

For Case 2, if we compare G_{sub} and $\hat{G}_{sub}$, we see that they differ only in their values of $\delta(\pi(\pi(v_{|V|+|C|})))$. (For $\hat{G}_{sub}$, that value is $\min\{\delta(\pi(\pi(v_{|V|+|C|}))), \max_{i\in\mathcal{Q}} \delta(v_i) + 1\}$.) Now observe the check node $\pi(v_{|V|+|C|})$ and its neighboring variable nodes: when BP decoder runs on G_{BFS}, all the nodes in $\{v_i | i \in \mathcal{Q}\}$ can be corrected automatically by iteration $\max_{i\in\mathcal{Q}} \delta(v_i) < k$; so by using the check node $\pi(v_{|V|+|C|})$, the node $\pi(\pi(v_{|V|+|C|}))$ can be corrected by iteration $\max_{i\in\mathcal{Q}} \delta(v_i) + 1 \leq k$. That is equivalent to turning $\delta(\pi(\pi(v_{|V|+|C|})))$ into

$$\min\{\delta(\pi(\pi(v_{|V|+|C|}))), \max_{i\in\mathcal{Q}} \delta(v_i) + 1\}$$

and turning G_{sub} into $\hat{G}_{sub}$ when it comes to BP decoding. That leads to the assertion.

For Case 3, the node $\pi(\pi(v_{|V|+|C|}))$ needs to be corrected by iteration $\omega(\pi(\pi(v_{|V|+|C|})))$. But since the nodes in $\{v_i | i \in \mathcal{Q}\}$ will not all be corrected automatically until iteration $\max_{i\in\mathcal{Q}} \delta(v_i)$, and it takes one more iteration for the check node $\pi(v_{|V|+|C|})$ to propagate information to node $\pi(\pi(v_{|V|+|C|}))$, they cannot help decode $\pi(\pi(v_{|V|+|C|}))$. So for G_{sub}, it makes no difference whether the subtree rooted at $\pi(v_{|V|+|C|})$ is there or not when it comes to BP decoding. That leads to the assertion.

For Case 4, if we compare G_{sub} and $\hat{G}_{sub}$, we see that they differ only in their values of $\delta(\pi(\pi(v_{|V|+|C|})))$ and $\omega(\pi(\pi(v_{|V|+|C|})))$. Now observe the check node $\pi(v_{|V|+|C|})$ and its neighboring variable nodes: when BP decoder runs on G_{BFS}, all the nodes in $\{v_i | i \in \mathcal{Q}\}$ can be corrected automatically by iteration $\max_{i\in\mathcal{Q}} \delta(v_i)$; so the check node $\pi(v_{|V|+|C|})$ can help correct the node $\pi(\pi(v_{|V|+|C|}))$ by iteration $\max_{i\in\mathcal{Q}} \delta(v_i) + 1 \leq \omega(\pi(\pi(v_{|V|+|C|})))$. That is equivalent to turning $\delta(\pi(\pi(v_{|V|+|C|})))$ into $\max_{i\in\mathcal{Q}} \delta(v_i)+1$, turning $\omega(\pi(\pi(v_{|V|+|C|})))$ to ∞ and turning G_{sub} into $\hat{G}_{sub}$ when it comes to BP decoding. That leads to the conclusion.

For Case 5, since $\mathcal{Q} = \emptyset$, the check node $\pi(v_{|V|+|C|})$ can help correct the node $\pi(\pi(v_{|V|+|C|}))$ in the 1st iteration. With an analysis similar to the above ones, we see that the assertion holds for

both sub-cases. □

Lemma 69. *Suppose* $\max_{i\in\mathcal{P}} \omega(v_i) > \max_{i\in\mathcal{Q}} \delta(v_i)$. *Let* i^* *be an integer in* $\mathcal{P}$ *such that* $\omega(v_{i^*}) = \max_{i\in\mathcal{P}} \omega(v_i)$. *Consider two cases:*

1. *Case 1: If* $\max_{i\in\mathcal{P}} \omega(v_i) > \delta(\pi(\pi(v_{|V|+|C|})))$, *let* S *be any minimum-sized* g*-Elimination Set for* G_{sub}.

2. *Case 2: If* $\max_{i\in\mathcal{P}} \omega(v_i) \leq \delta(\pi(\pi(v_{|V|+|C|})))$, *let* S *be any minimum-sized* g*-Elimination Set for* $\hat{G}_{sub}$ *where* $\delta(\pi(\pi(v_{|V|+|C|})))$ *is changed to*

$$\infty$$

and $\omega(\pi(\pi(v_{|V|+|C|})))$ *is changed to*

$$\min\{\omega(\pi(\pi(v_{|V|+|C|}))), \max_{i\in\mathcal{P}} \omega(v_i) - 1\}.$$

Then $S \cup \{v_i | i \in \mathcal{P}, i \neq i^*\}$ *is a minimum-sized* g*-Elimination Set for* G_{BFS}.

Proof. By Lemma 66, there exists a minimum-sized g-Elimination Set for G_{BFS} that includes the nodes in $\{v_i | i \in \mathcal{P}, i \neq i^*\}$ but not v_{i^*}. Let T^* be such a minimum-sized g-Elimination Set for G_{BFS}.

For Case 1, when the g-Elimination Set for G_{BFS} is T^*, the subtree rooted at the check node $\pi(v_{|V|+|C|})$ cannot help correct the node $\pi(\pi(v_{|V|+|C|}))$. Instead, those nodes of T^* that are in G_{sub} will be a g-Elimination Set for G_{sub}, and the BP decoder will correct the un-removed nodes in G_{sub} (within each of their required number of iterations $\omega(v)$). If $\pi(\pi(v_{|V|+|C|})) \in T^*$, the check node $\pi(v_{|V|+|C|})$ will correct v_{i^*} in the 1st iteration; otherwise, the BP decoder will correct $\pi(\pi(v_{|V|+|C|}))$ in at most $\delta(\pi(\pi(v_{|V|+|C|})))$ iterations, so $\pi(v_{|V|+|C|})$ will correct v_{i^*} in at most $\delta(\pi(\pi(v_{|V|+|C|}))) + 1 \leq \omega(v_{i^*})$ iterations. Since T^*'s size is minimized, the number of nodes of T^* that are in G_{sub} is also minimized. That leads to the conclusion.

For Case 2, when the g-Elimination Set for G_{BFS} is T^*, the BP decoder needs to correct the node $\pi(\pi(v_{|V|+|C|}))$ by iteration $\max_{i\in\mathcal{P}}\omega(v_i) - 1 < \delta(\pi(\pi(v_{|V|+|C|})))$ because only then will the check node $\pi(v_{|V|+|C|})$ help correct the node v_{i^*} by iteration $\max_{i\in\mathcal{P}}\omega(v_i) = \omega(v_{i^*})$. That is equivalent to turning $\omega(\pi(\pi(v_{|V|+|C|})))$ into $\min\{\omega(\pi(\pi(v_{|V|+|C|}))), \max_{i\in\mathcal{P}}\omega(v_i) - 1\}$, turning $\delta(\pi(\pi(v_{|V|+|C|})))$ into ∞ and turning G_{sub} into $\hat{G}_{sub}$ when it comes to BP decoding. That leads to the conclusion. □

By using the above two lemmas repeatedly, we can reduce the $gSSE$ Problem from G_{BFS} to smaller and smaller subtrees, until the subtree contains only the root node v_1 (and v_1 will be included in the g-Elimination Set if and only if $\omega(v_1) \leq k$ at that moment). An algorithm based on the above idea is presented below.

Algorithm 70. *Exact Algorithm for SSE_k*

Input: Stopping Tree $G = (V \cup C, E)$, integer $k > 0$.

Output: A k-iteration Elimination Set of minimum size in G.

Algorithm:

1. *Generate G_{BFS} by running BFS on G.*
2. **for** $i = 1$ *to* $|V| + |C|$
3. *{*
4. **if** $v_i \in V$
5. $\delta(v_i) \leftarrow \infty$, $\omega(v_i) \leftarrow k$.
6. *}*
7. *Let S be an empty set.*
8. $i \leftarrow |V| + |C|$.
9. **while** $i \geq 1$

10. {

11. **if** $i > 1$ *and* $v_i \in V$

12. *{*

13. *Let* τ *be the minimum integer such that* v_τ *either is a sibling of* v_i *or is* v_i *itself.*

14. $\mathcal{P} \leftarrow \{y | \tau \leq y \leq i, \omega(v_y) \leq k\}$.

15. $\mathcal{Q} \leftarrow \{y | \tau \leq y \leq i, \delta(v_y) \leq k\}$.

16. **if** $\max_{j \in \mathcal{P}} \omega(v_j) \leq \max_{j \in \mathcal{Q}} \delta(v_j)$

17. *{*

18. **if** $|\mathcal{Q}| > 0$, $\max_{j \in \mathcal{Q}} \delta(v_j) < k$ *and* $\delta(\pi(\pi(v_{|V|+|C|}))) \leq k$

19. *{*

20. $\delta(\pi(\pi(v_{|V|+|C|}))) \leftarrow \min\{\delta(\pi(\pi(v_{|V|+|C|}))), \max_{j \in \mathcal{Q}} \delta(v_j) + 1\}$.

21. *}*

22. **else if** $|\mathcal{Q}| > 0$ *and* $\max_{j \in \mathcal{Q}} \delta(v_j) < \omega(\pi(\pi(v_{|V|+|C|}))) \leq k$

23. *{*

24. $\delta(\pi(\pi(v_{|V|+|C|}))) \leftarrow \max_{j \in \mathcal{Q}} \delta(v_j) + 1$.

25. $\omega(\pi(\pi(v_{|V|+|C|}))) \leftarrow \infty$.

26. *}*

27. **else if** $|\mathcal{Q}| = 0$ *and* $\omega(\pi(\pi(v_{|V|+|C|}))) > 0$

28. *{*

29. $\delta(\pi(\pi(v_{|V|+|C|}))) \leftarrow 1.$

30. $\omega(\pi(\pi(v_{|V|+|C|}))) \leftarrow \infty.$

31. *}*

32. $S \leftarrow S \cup \{v_j | j \in \mathcal{P}\}$

33. *}*

34. **else**

35. *{*

36. *Let* i^* *be an integer in* $\mathcal{P}$ *such that* $\omega(v_{i^*}) =$ $\max_{j \in \mathcal{P}} \omega(v_j)$.

37. **if** $\max_{j \in \mathcal{P}} \omega(v_j) \leq \delta(\pi(\pi(v_{|V|+|C|})))$

38. *{*

39. $\delta(\pi(\pi(v_{|V|+|C|}))) \leftarrow \infty.$

40. $\omega(\pi(\pi(v_{|V|+|C|}))) \leftarrow$ $\min\{\omega(\pi(\pi(v_{|V|+|C|}))), \max_{j \in \mathcal{P}} \omega(v_j) - 1\}.$

41. *}*

42. $S \leftarrow S \cup \{v_j | j \in \mathcal{P}, j \neq i^*\}.$

43. *}*

44. $i \leftarrow \tau - 1$.

45. *}*

46. **else if** $i = 1$

47. *{*

48. **if** $\omega(v_1) \leq k$

49. $S \leftarrow S \cup \{v_1\}$.

50. $i = 0$.

51. *}*

52. **else**

53. $i \leftarrow i - 1$.

54. *}*

55. **Return** S.

Based on the previous analysis, we get the correctness of the algorithm.

Theorem71. *Algorithm 70 returns an optimal (i.e., minimum-sized)* k*-iteration Elimination Set of* $G = (V \cup C, E)$.

The algorithm has time complexity $O(|V| + |C|)$.

6. DEEP LEARNING FOR REPRESENTATION-OBLIVIOUS ERROR CORRECTION BY NATURAL REDUNDANCY [1]

6.1 Introduction

The amount of data in storage systems is increasing at an accelerating speed in the big data era. Storage systems have a strong need for substantially improving their error correction capabilities, especially for long-term storage where the accumulating errors can exceed the decoding threshold of error-correcting codes (ECCs) [135]. Memory scrubbing alone is not a sufficient solution: even for nonvolatile memory systems such as SSDs (solid-state drives) that have fast read/write speeds, scrubbing all data periodically is still too costly due to the volume of the data. Therefore it is highly necessary to find new techniques to assist ECCs and substantially enhance their error correction performance.

One promising technique is to use the internal redundancy in data for error correction, and combine it with ECC's decoding algorithm. This type of redundancy, called *natural redundnacy*, has been explored in recent works [30, 112, 113, 114]. In practical storage systems, many files are either uncompressed or compressed imperfectly, especially for languages and images because their highly complex data models make perfect compression infeasible due to prohibitively high computational complexities. The residual redundancy (i.e., natural redundancy) in data can then be combined with the redundancy artificially added by ECCs (i.e., parity-check bits) for joint error correction. There is often plenty of natural redundancy in data. For instance, for the English language, state-of-the-art compression algorithms (e.g., syllable-based Burrows-Wheeler Transform) can compress it to 2 bits/character [124], which is still far from Shannon's estimation of 1.34 bits/character as the entropy of printed English [8]. For images, their true entropy remains unknown. But recent progress in deep learning, such as the inpainting techniques for completing images [136], suggests that the natural redundancy in even compressed images is still substantial.

[1] ©IEEE 2019. Reprinted, with permission, from P. Upadhyaya and A. A. Jiang, "Representation-Oblivious Error Correction by Natural Redundancy," 2019 IEEE International Conference on Communications (ICC), Shanghai, China, 2019

In [30, 114], natural redundancy has been used to help ECCs – including LDPC codes and polar codes – correct errors and achieved significant performance improvement. Those works have addressed both languages and images, but mainly for texts in English compressed by Huffman codes or fixed-codebook LZW codes [1].

In this work, we study how to use natural redundancy for error correction in a more practical setting. We consider noisy file segments from files of different types (e.g., HTML, LaTeX, PDF and JPEG), and correct errors in them even when their bit error rates (BERs) have significantly exceeded the decoding threshold of the ECC. The scheme is *representation-oblivious*: it requires no prior knowledge on how data are represented in those different file types, e.g., how symbols/characters are mapped to bits, how/whether data are compressed, and how meta data are used in those files. This approach makes the solution more convenient to use for storage systems. It is different from the previous works (e.g., [1, 30]), where the data (e.g., texts) are compressed by known compression algorithms (e.g., Huffman or LZW code) and without any additional data formatting (e.g., meta data or file formats) that brings more complexity. (In those works, the codebook of Huffman or LZW code is used for decoding. In this work we do not use any codebook.) We take this representation-oblivious approach because in storage systems (such as SSDs), many file types have proprietary compression algorithms or file formats that are often unrevealed to the public, including to storage device manufacturers. Also, since error correction is a low-layer function in the storage architecture, the controllers of storage devices do not necessarily have access to file systems to get information on file types, data formats or compression algorithms. By taking the representation-oblivious approach, we can explore error correction schemes based on natural redundancy that are more widely usable in storage systems.

The coding scheme of this paper is illustrated in Fig. 6.1. When files are stored, each file is partitioned into segments of k bits, and each file segment is encoded by a systematic (n, k) ECC into a codeword of n bits. Then each ECC codeword passes through a noisy channel, which models the errors in a storage device. During decoding, first, a deep neural network (DNN) uses the k noisy information bits to recognize the file type (e.g. HTML, LaTeX, PDF or JPEG) of the file segment.

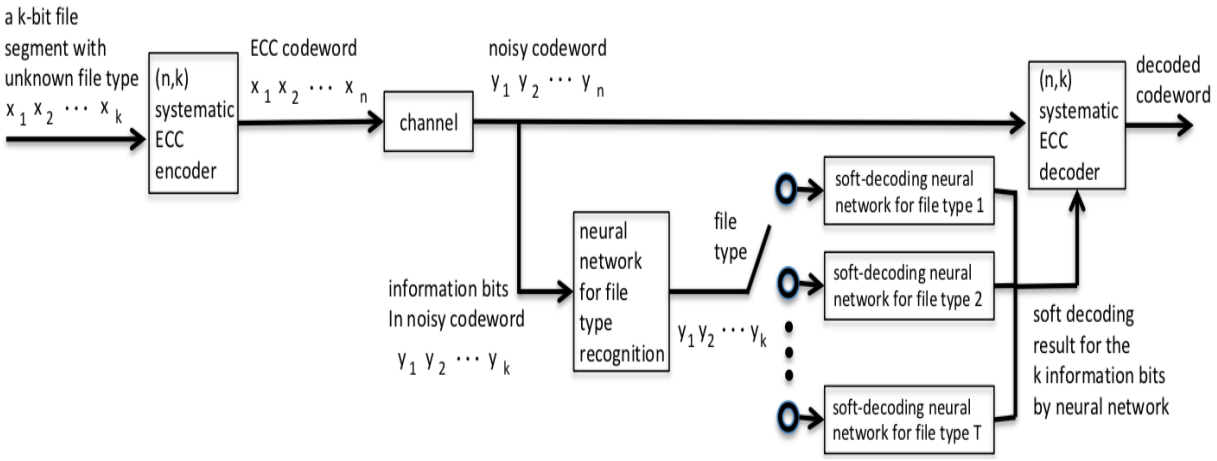

Figure 6.1: Encoding and decoding scheme for a noisy file segment of an initially unknown file type. The k-bit file segment is encoded by a systematic (n, k) ECC into an n-bit codeword. The codeword is transmitted through a channel to get a noisy codeword. Two neural networks use natural redundancy to decode the k noisy information bits: the first network determines the file type of the file segment, and then a corresponding neural network for that file type performs soft decoding for the k noisy information bits. The soft decoding result and the noisy codeword are both given to the ECC decoder for further error correction.

Then, a second DNN for that file type performs soft decoding on the k noisy information bits based on natural redundancy, and outputs k probabilities, where for $i = 1, 2, \cdots, k$, the i-th output is the probability for the i-th information bit to be 1. The k probabilities are given as additional information to the ECC's decoder. The ECC decoder then performs its decoding and outputs the final result. (In our experiments, the ECC is a systematic LDPC code, and the k probabilities are combined with the initial LLRs (log-likelihood ratios) for information bits to obtain their updated LLRs. The LDPC code then runs its belief-propagation (BP) decoding algorithm.)

The above scheme can be extended to the case where the two decoders – the decoder based on natural redundancy (NR decoder) and the ECC decoder – perform iterative decoding between them. That is, each decoder's output is given to the other decoder as input, and the decoding process iterates between the two decoders. Iterative decoding of this type for the English language compressed by known compression algorithms has been studied in [30, 113]. The scheme can also be extended to the case where an ECC codeword may contain multiple file segments of multiple file types. For simplicity, such extensions are not explored here.

This work has several contributions. First, it designs a deep neural network that recognizes file types with high accuracy from noisy bits. For error correction, this DNN helps recognize the type of natural redundancy in the noisy data.

Second, it designs deep neural networks for decoding data with natural redundancy, where the data have errors from the binary-symmetric channel (BSC). The DNNs perform soft decoding instead of hard-decision decoding, which can be more useful for ECCs such as LDPC codes. Since the data used to train the DNNs do not contain soft decoding results, we design a new portfolio theory-based approach to train the DNNs. The results show that the DNNs can learn soft decoding with high accuracy, even though the training data are extremely sparse compared to possible data patterns.

Third, the paper presents a scheme that combines the natural redundancy based decoding, which applies deep learning to noisy file segments of different file types, with ECC decoding. The experimental results confirm that the scheme substantially improves the reliability of different types of files.

There have been numerous recent works on using deep learning for information theory [137, 138], especially for wireless and optical communications. They mainly focus on using deep learning to model complex channels, design codes, and approximate or improve decoding algorithms [139, 140]. In contrast to those works, this work focuses on using deep learning for *data* with complex structures, and explore error correction for such complex data. These different directions can complement each other in a communication or storage system with both complex data and complex channels.

6.2 File Type Recognition Using Deep Learning

In this section, we present a *Deep Neural Network* (DNN) for file type recognition. The DNN takes a noisy file segment of k bits, $(y_1, y_2, \cdots, y_k)$, as input, and outputs one of T file types (e.g., HTML, LaTeX, PDF or JPEG). The errors in the file segment come from a binary-symmetric channel (BSC) of bit-error rate (BER) p. We first introduce the architecture of the DNN and its training method. We then present the experimental results, which show that it achieves high

accuracy for file type recognition.

6.2.1 DNN Architecture and Training

Our DNN architecture is shown in Fig. 6.2. It is a Convolutional Neural Network (CNN) that takes the k bits of a noisy file segment as input. In our experiments, we let $k = 4095$. (The LDPC code we use is a $(4376, 4095)$ code designed by MacKay [115], which can tolerate BER of 0.2%. Both the code length and the BER are in the typical range of parameters for storage systems.) The CNN has T outputs that correspond to the T possible file types, namely, the T classification results. The output with the highest value leads to the selection of the corresponding file type. In our experiments, we consider four file types: HTML, LaTeX, PDF and JPEG. So $T = 4$. Note that HTML and LaTeX files are both text sequences but have different file structures; PDF files contain both texts and images; and JPEG files are images. In the following, we will present DNNs and experiments using those parameters for the convenience of presentation. Note that the designs can be extended to other file-segment lengths and more file types.

In Fig. 6.2, there are $L = 9$ convolution layers $\{C_1, C_2, ...C_L\}$ where each layer C_d (for $d = 1, 2, \cdots, L$) is followed by a max pooling layer M_d. The last max pooling layer M_L is followed by a dense layer D. For $d = 1, 2, \cdots, L$, let n_d denote the number of feature maps of the convolution layer C_d. (In Fig. 6.2, $n_1 = 32$ and $n_2 = n_3 = \cdots = n_9 = 64$.) Those feature maps are obtained by taking convolution on the output of the previous layer using n_d filters of size $l_d = 3$. In its subsequent max pooling layer M_d, pooling windows of size 2 are applied to each feature map of C_d with a stride of two. Let K_d denote the length of each feature map (in the dimension of the CNN's input) of the layer C_d. Then $K_1 = k - l_d + 1$, and $K_d = \lfloor \frac{K_{d-1}}{2} \rfloor - l_d + 1$ for $2 \leq d \leq L$.

The CNN uses *ReLU* and *sigmoid* as the activation function of its convolutional layers and output layer, respectively. It uses *cross entropy* as its loss function. (Let $(z_1, z_2, z_3, z_4) \in \{0, 1\}^4$ denote the desired output; where $z_i = 1$, if the correct file type is type i, and $z_j = 0$ for any $j \neq i$. Let (z'_1, z'_2, z'_3, z'_4) be the actual output of the network. Then cross entropy is defined as $-\sum_{i=1}^{T} z_i log z'_i$.) Its optimizer is chosen to be an *Ada Delta Optimizer*, whose parameters are: learning rate = 1.0, ρ = 0.95, ϵ = *none* and decay = 0. During training, each mini-batch has 100

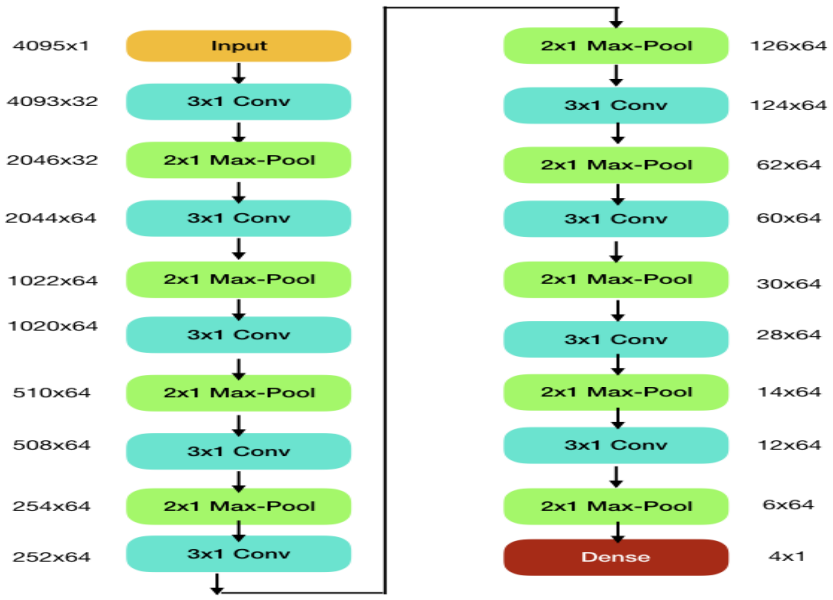

Figure 6.2: Architecture of the CNN (convolutional neural network) for File Type Recognition. Its input is a noisy file segment of 4095 bits, and its output corresponds to 4 candidate file types (HTML, LaTex, PDF and JPEG). The numbers beside each layer (namely, 4095×1, 4093×32, $\cdots$, 4×1) are the dimension sizes of the layer's output data. The numbers inside each layer (namely, 3×1 or 2×1) are the dimension sizes of the corresponding feature-map filter or pooling window.

training samples, where each training sample consists of a noisy file segment and its file-type label (i.e., one of the T file types).

A large dataset has been used to train and test the CNN. For each of the $T = 4$ file types, 24,000 noiseless file segments are used for training data, 4,000 noiseless file segments are used for validation data, and 4,800 noiseless file segments are used for test data. During training and testing, random errors of BER p are added to each file segment, where each file segment uses an independently generated error pattern.

6.2.2 Experimental Performance

The $(4376, 4095)$ LDPC code used in our experiments can correct errors of BER up to 0.2% *by itself*. (That is, when it is used in the conventional way without the extra help of natural redundancy, it has a decoding threshold of 0.2%.) Our goal is to use the natural redundancy in file segments to correct errors of substantially higher BERs. So we have selected the target BER p with substantially

Bit Error Rate (BER)	Overall Test Accuracy	HTML Test Accuracy	JPEG Test Accuracy	PDF Test Accuracy	LaTeX Test Accuracy
0.2%	99.61%	99.98%	99.52%	99.17%	99.77%
0.4%	99.69%	99.96%	99.60%	99.25%	99.96%
0.6%	99.60%	99.94%	99.48%	99.06%	99.90%
0.8%	99.69%	99.98%	99.50%	99.35%	99.92%
1.2%	99.66%	99.96%	99.23%	99.48%	99.96%
1.6%	99.58%	99.96%	99.60%	98.83%	99.92%

Table 6.1: Bit error rate (BER) vs Test Accuracy for File Type Recognition (FTR). Here the "overall test accuracy" is for all 4 types of files together. The last four columns show the test accuracy for each individual type of files. (Their average value is the overall test accuracy.)

higher values, ranging from 0.2% to 1.6%. We then train the CNN with the given target BER p.

We measure the performance of the CNN by the *accuracy* of file type recognition (FTR), which is defined as the fraction of file segments whose file types are recognized correctly. The CNN is trained using the training and validation data. Its final performance is measured using the test data, where file segments of the $T = 4$ file types are randomly mixed. The test performance is shown in Table 6.1. It can be seen that file types can be recognized by the CNN with high accuracy: for all BERs, the accuracy is close to 1.

We can also examine the accuracy for recognizing each file type, and see if there is variance in performance from file type to file type. The results are shown in the last four columns of Table 6.1. It can be seen that overall, the accuracy is constantly high for all file types.

The CNN's performance compares favorably with existing results on FTR, which has been studied previously for applications such as disk recovery. The work [141] considered a classification method for a pair of file types using Fisher's linear discriminant and longest common subsequence methods. The accuracy ranges between 87% and 99% depending on which pair of file types are considered. The work [142] introduced an NLP (natural language processing) based method, where unigram and bigram counts of bytes and other statistics are used to generate feature representation, which is then followed by support vector machine (SVM) for classification of

various file types. The classification accuracy varies from 17.4% for JPEG files, 62.5% for PDF files to 94.8% for HTML files. The work [143] used PCA (principal component analysis) and a feed-forward auto-associative unsupervised neural network for feature extraction, and a three layer multi-layer perceptron network for classification. The classification accuracy is 98.33% for six file types while considering entire files instead of file segments. Our deep-learning based method can be seen to achieve high performance, without the need to train separate modules for feature extraction and classification.

The CNN has robust performance because it works well not only for the BER it is trained for, but also for other BERs in the considered range. (For example, a CNN trained for $BER = 1.2\%$ also works well for other BERs in the range $[0.2\%, 2.0\%]$.) For succinctness we skip the details. The robustness of the overall error correction performance for different BERs will be presented in Section IV.

6.3 Soft Decoding by Deep Neural Networks

In this section, we study how to design DNNs that can perform soft decoding on noisy file segments. For each of the T file types, we will design and train a different DNN, because different types of files have different types of natural redundancy. Given a file type, we will design a DNN whose input is a noisy file segment of k bits $Y = (y_1, y_2, \cdots, y_k)$. As before, the errors in the noisy file segment come from a binary-symmetric channel (BSC) of bit-error rate (BER) p. The output of the DNN is a vector $Q = (q_1, q_2, \cdots, q_k)$, where for $i = 1, 2, \cdots, k$, the real-valued output $q_i \in [0, 1]$ represents the DNN's belief that for the i-th bit in the file segment, the probability that its correct value should be 1 is q_i. In other words, if we use $X = (x_1, x_2, \cdots, x_k)$ to denote an error-free file segment, and let it pass through a BSC of BER p to obtain a noisy file segment $Y = (y_1, y_2, \cdots, y_k)$, then q_i is the DNN's estimation for $Pr\{x_i = 1 \mid Y, p\}$. Note that the k bits are not independent of each other because of the natural redundancy in them. So $Pr\{x_i = 1 \mid Y, p\}$ depends on not only y_i and p, but also the overall value of Y. The goal of the DNN is to learn the natural redundancy in file segments, and use it to make the probability estimation q_i be as close to the true probability $Pr\{x_i = 1 \mid Y, p\}$ as possible, for each i and for each possible value Y of the

noisy file segment.

We treat this problem as a regression problem. Note that since errors are random without an upper bound to it's Hamming weight, given a noisy codeword the corresponding correct codeword is not unique in principle. So, the soft decoding result for each codeword bit should really be a probability (instead of a 0-or-1 label). However, the training data only have binary codewords, so only 0-or-1 labels are available for training.

In the following, we present a new approach based on portfolio theory that teaches a DNN *soft decoding*. We first present the idea, and verify its performance for data with small samples spaces. We then extend it to file segments, which have complex natural redundancy and a very large sample space.

6.3.1 Portfolio Theory-based Soft Decoding

Consider a channel, whose input is a variable $X \in \{a_i \mid 1 \leq i \leq K\}$, and whose output is a single bit $b \in \{0, 1\}$. For $i = 1, 2, \cdots, K$, let $p_i \triangleq Pr\{b = 1 \mid X = a_i\}$ be the probability that the channel's output is 1 given that its input is a_i. Consider a sequence of N such input-output pairs of the channel

$$(X_1, b_1),\ (X_2, b_2),\ \cdots,\ (X_N, b_N)$$

where for $j = 1, 2, \cdots, N$, X_j and b_j are the input and output of the channel, respectively, for the j-th use of the channel. Now assume that we do not know the channel's transition probabilities p_1, p_2, $\cdots$, p_K. Instead, we have the sequence of N input-output pairs, and want to use them to estimate those transition probabilities. (We would like to achieve this goal without counting how many times the channel output is 1 for every given channel input value, because when this method is applied to file segments later, the channel will have $K = 2^k = 2^{4095}$ possible input values, which is too large for counting to work due to the sparsity of training data and the memory constraint.)

Now suppose that we have derived a policy, which estimates the probability $Pr\{b = 1 \mid X = a_i\}$ as q_i (when its true value should be p_i). Consider the following game on horse race, which is between two horses – a white horse and a black horse – and takes X as its environment parameter

(e.g., the wind speed, temperature, etc. at the race). Let p_i (respectively, $1 - p_i$) denote the probability that the white (respectively, black) horse wins when $X = a_i$. We bet on a sequence of N races. For $j = 1, 2, \cdots, N$, for the j-th race, if its environment parameter $X = a_i$ (for some $i \in \{1, 2, \cdots, K\}$), we bet a fraction of q_i of our money on the white horse, and bet the remaining $1 - q_i$ of our money on the black horse. If the white (respectively, black) horse wins in that race – which corresponds to the channel output $b = 1$ (respectively, $b = 0$) – the money we get is the amount of money we bet on the white (respectively, black) horse times some constant c. Now let us define a variable S_j for the j-th race: if the white horse wins, we let $S_j = q_i$; otherwise, we let $S_j = 1 - q_i$.

Suppose that we started with 1 dollar. After the N races, the money we have is $c^N \prod_{j=1}^{N} S_j$. Define the *doubling rate* as $R = \frac{log_2(\prod_{j=1}^{N} S_j)}{N} = \frac{1}{N} \sum_{j=1}^{N} log(S_j)$.

Example 72. *Let $K = 4$, and $N = 5$. Assume that we get the following sequence: $(X_1, b_1) = (a_2, 0), (X_2, b_2) = (a_1, 1), (X_3, b_3) = (a_3, 1), (X_4, b_4) = (a_3, 0), (X_5, b_5) = (a_4, 0)$. Then the doubling rate is $R = \frac{1}{5}[log_2(1 - q_2) + log_2(q_1) + log_2(q_3) + log_2(1 - q_3) + log_2(1 - q_4)]$.*

By portfolio theory [144], when $N \to \infty$, the doubling rate R is maximized only if $q_i = p_i$ for $i = 1, 2, \cdots, K$. Therefore, to learn the transition probabilities p_1, p_2, $\cdots$, p_K, we can design a neural network (NN) that takes the sequence of N channel input-output pairs

$$(X_1, b_1),\ (X_2, b_2),\ \cdots,\ (X_N, b_N)$$

as training data, and define the loss function of the NN as $-R$ namely, the *negative doubling rate*. (It is interesting to see that the loss function is in fact the cross entropy between the codeword bits of the training data and the output probabilities here, which means the problem can also be treated as a classification problem.) For $j = 1, 2, \cdots, N$, each X_j is an input to the NN, and – assuming $X_j = a_i$ for some i – the NN's output is considered to be q_i; and based on whether b_j is 1 or 0, an additive term of $\log_2 q_i$ or $\log_2(1 - q_i)$ is included in the loss function. As the NN is trained, it

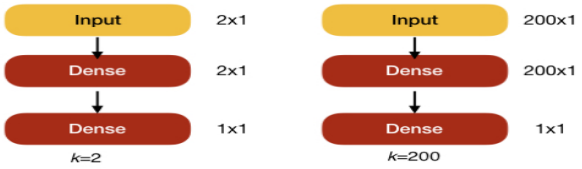

Figure 6.3: Neural networks for $K = 2$ (left) and $K = 200$ (right).

minimizes the loss function, which is equivalent to learning the correct transition probabilities and maximizing the doubling rate R.

In practice, the NN needs to gradually train its weights as it gets more and more training data, and N cannot be infinite. So we need to partition the channel input-output pairs into batches, and let the NN use every batch to compute its loss functions and adjust its weights. To verify if the NN can learn the true transition probabilities effectively using batches of small sizes, we use the following experiments.

For $K \geq 2$, we design a NN as follows. We take $N = K \times 50000$ channel input-output pairs in total. (Here we select each transition probability p_i uniformly randomly from the range $(0, 1)$. Then, given $p_1, p_2, \cdots, p_K$, we generate the N channel input-output samples following those transition probabilities.) Let the batch size be $K \times 50$. Let the NN have three layers: an input layer, a hidden layer, and an output layer. The input layer uses one-hot encoding for the K possible values for X; so the size of the input layer is $K \times 1$. The size of the hidden layer is set to $K \times 1$ and is fully connected to the input layer. The size of the output layer is 1×1 (namely, just one number). (For illustration, the architectures of the NNs for $K = 2$ and $K = 200$ are shown in Fig. 6.3.) After the NN is trained, for each input $X = a_i$, it can output the corresponding probability estimation q_i (for $i = 1, 2, \cdots, K$).

We measure the distance between the true channel transition probabilities and the NN's estimation by their average Kullback-Liebler (KL) divergence

$$\Delta_K = \frac{1}{K} \sum_{i=1}^{K} D(p_i \,||\, q_i),$$

K	2	4	10	100	200
Δ_K	0.000069	0.000022	0.00015	0.00023	0.00021

Table 6.2: Average KL Divergence between true and learned transition probabilities.

where $D(p_i \mid\mid q_i) = p_i log_2 \frac{p_i}{q_i} + (1-p_i) log_2 \frac{1-p_i}{1-q_i}$. The average KL divergence for different values of K are shown in Table 6.2. It can be seen that the KL divergence is very small, which means that the NN has learned the true transition probabilities well.

6.3.2 Soft Decoding for Noisy File Segments

In the previous subsection, we have shown that the portfolio theory-based approach, which sets the NN's loss function as the negative doubling rate, works well for relatively simple channel models. However, when we apply this approach to file segments, several challenges appear. First, the output is no longer the probability for only one bit b; instead, it consists of k probabilities $q_1, q_2, \cdots, q_k$ for the k bits in the file segment. Our DNN needs to estimate them jointly using one network architecture. Second, in the experiments of the last subsection, the K transition probabilities $p_1, p_2, \cdots, p_K$ are chosen independently and therefore have a simple structure; however, for file segments, the natural redundancy can be very complex, which can make the channel's transition probabilities be highly correlated and exhibit complex structures. Third, for file segments, the DNN's input can theoretically take $K = 2^k = 2^{4095}$ possible values, which is a huge number and makes the training data very sparse. So it is not simple to see whether the DNN can learn the transition probabilities well given the sparsity of training data.

In this subsection, we design DNNs for the soft decoding of noisy file segments. Our DNN architecture is related to auto-encoders. It consists of convolution layers followed by deconvolution layers. (Deconvolution layers may be seen as reverse operations of convolution layers. Interested readers can refer to [47] for more details.) Auto-encoders are good choices for various applications related to denoising [145, 146].

We illustrate our DNN architecture through some examples. Let p be the BER of the binary-symmetric channel that adds errors to file segments. Consider $p = 0.8\%$, 1.2%, 1.6%. The DNN

architecture for HTML (respectively, LaTeX) files, for all the above values of p, is shown in Fig. 6.4 (a) (respectively, in Fig. 6.4 (b)). When $p = 0.8\%$, the DNN architecture for both PDF and JPEG files is shown in Fig. 6.4 (c). When $p = 1.2\%$, 1.6%, the DNN architecture for PDF (respectively, JPEG) files is shown in Fig. 6.4 (d) (respectively, in Fig. 6.4 (e).

Since the transition probabilities corresponding to file segments are unknown, we cannot measure the performance of the DNN directly using the KL divergence. However, the experiments for error correction, which are to be presented in the next section, will confirm that the soft decoding result from the DNN is very useful for error correction.

6.4 Error Correction for Noisy File Segments

In this section, we combine the soft decoding output of the DNN – which was presented in the previous section – with an LDPC code for enhanced error correction performance. We adopt a *robust scheme* here: the DNNs for file-type recognition and for soft decoding have been trained with a constant BER p_{DNN}, but they are used for a wide range of BERs p for the BSC channel. (For example, the DNNs may be trained just for $p_{DNN} = 1.2\%$, but are used for any BER p from 0.2% to 1.6% in experiments here.) We choose this robust scheme because when DNNs are designed, the future BER in data can be highly unpredictable. That is exactly why errors may exceed ECC's thresholds for long-term storage, and why natural redundancy can become useful for error correction.

Given a noisy systematic LDPC codeword, we first use a DNN to recognize its file type based on its k noisy information bits. Then a second DNN for that file type is used to do soft decoding for the k noisy information bits, and output k probabilities: for $i = 1, 2, \cdots, k$, the i-th output p_i represents the estimated probability for the i-th information bit to be 1. Those k probabilities can be readily turned into LLRs (log-likelihood ratios) for the information bits using the formula $LLR_i^{DNN} = \log(\frac{1-p_i}{p_i})$. For $i = 1, 2, \cdots, n$, let $LLR_i^{channel}$ be the LLR for the i-th codeword bit (with $1 \leq i \leq k$ for information bits, and $k+1 \leq i \leq n$ for parity-check bits) derived for the binary-symmetric channel, which is either $\log(\frac{1-p}{p})$ (if the received codeword bit is 0) or $\log(\frac{p}{1-p})$

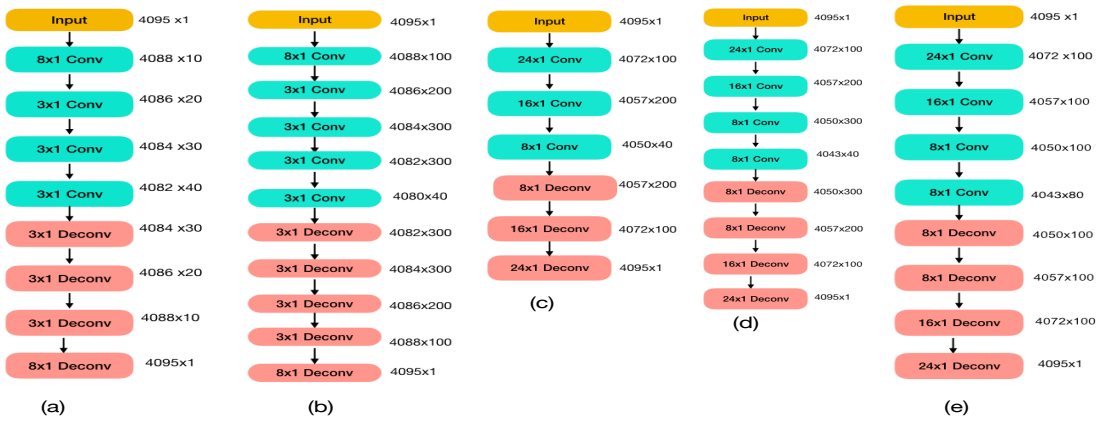

Figure 6.4: Architectures of deep neural networks (DNNs) for soft decoding of noisy file segments. (a) DNN architecture for HTML files for $p = 0.8\%$, 1.2%, 1.6%, (b) DNN architecture for LaTex files for $p = 0.8\%$, 1.2%, 1.6%, (c) DNN architecture for PDF and JPEG files when $p = 0.8\%$, (d) DNN architecture for PDF files when $p = 1.2\%$, 1.6%, (e) DNN architecture for JPEG files when $p = 1.2\%$, 1.6%.

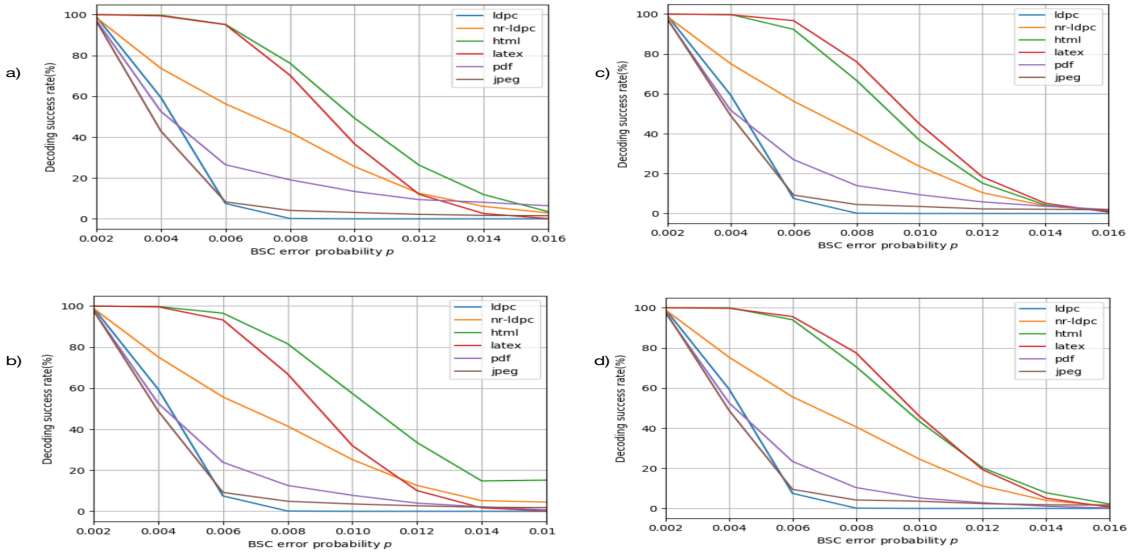

Figure 6.5: Decoding success rate vs bit error rate for (a) p_{DNN} = 1.0% , (b) p_{DNN} = 1.2%, (c) p_{DNN} = 1.4%, (d) p_{DNN} = 1.6%.

(if the received codeword bit is 1). Then we let the *initial LLR* for the i-th codeword bit be

$$LLR_i^{int} = LLR_i^{channel} + LLR_i^{DNN}$$

for $1 \leq i \leq k$, and $LLR_i^{int} = LLR_i^{channel}$ for $k+1 \leq i \leq n$. We then do belief-propagation (BP) decoding using the initial LLRs, and get the final result.

Note that there is a positive – although very small – chance that the file type will be recognized incorrectly. In that case, the incorrect soft-decoding DNN will be used. And that is accounted for in the overall decoding performance. We measure the performance of the error correction scheme by the percentage of codewords that are decoded correctly, which we call *Decoding Success Rate*. (Let us call the scheme the *NR-LDPC decoder*, since it combines decoding based on natural redundancy and the LDPC code.) We focus on BERs that are beyond the decoding threshold of the LDPC code, because natural redundancy becomes helpful in such cases. Note that the $(4376, 4095)$ LDPC code used in our experiments has a decoding threshold of $BER = 0.2\%$. In our experiments, we focus on BERs p that are not only beyond the decoding threshold, but also can be significantly larger: $p \in [0.2\%,\ 1.6\%]$.

The experimental results for $p_{DNN} = 1.0\%$ is presented in Fig. 6.5 (a). Here the x-axis is the channel error probability p, and the y-axis is the decoding success rate. (For each p, 1000 file segments with independent random error patterns have been used in experiments.) The curve for "ldpc" is the performance of the LDPC decoder alone, and the curve for "nr-lpdc" is for the NR-LDPC decoder. It can be seen that the NR-LDPC decoder achieves significantly higher performance. For example, as $p = 0.6\%$, the decoding success rate of the NR-LDPC decoder is approximately 4 times as high as the LDPC decoder.

The figure also shows the performance for each of the 4 file types. (The 4 curves are labelled by html, latex, pdf, jpeg, respectively. Their average value becomes the curve for "nr–ldpc".) It shows that the error correction performance for HTML and LaTex files are significantly better than for PDF and JPEG files. It is probably because the former two mainly consist of languages, for which

the soft-decoding DNNs are better at finding their patterns and mining their natural redundancy, while PDF is a mixture of languages and images and JPEG is image only. It is interesting to notice that even for JPEG files, when $p > 0.6\%$, the NR-LDPC decoder again performs better than the LDPC decoder, which means the DNNs can extract natural redundancy from images, too. Fig. 6.5 (b) to Fig. 6.5 (d) show the performance for p_{DNN} = 1.2%, 1.4% and 1.6%, respectively. The NR-LDPC decoder performs equally well in those cases, which proves the value of natural redundancy for decoding.

6.5 Conclusion

This section presents a new scheme that combines natural redundancy with LDPC codes for error correction. It is applied to noisy file segments of initially unknown file types, – which is the first of its kind to the best of our knowledge, — and shows substantial performance improvement compared to the original LDPC decoding scheme. The study can be extended to more types of natural redundancy in various types of files, more DNN architectures, and more ways to combine the NR decoder and ECC decoder, such as through iterative decoding between the two. When perfect error correction is not necessary (e.g., for images), non-binary LDPC codes and neural networks can also be combined naturally[147]. Those remain as our future research directions.

7. EXTERNALLY CODED NEURAL NETWORKS

7.1 Introduction

Deep Learning, as an important tool in AI, has found many applications in computer vision, speech recognition, robotics etc. This has led to increasing interest in the hardware implementation of Deep Neural Networks (DNNs) and embedded neuromorphic systems. Neural networks in hardware enjoy advantages in faster speed, low power consumption and easy integration with embedded systems [13, 148]. However, they suffer from noise that accumulates over time. A typical DNN consists of millions or more weights, where every weight is usually a real number. A natural approach is to store these real-valued weights as analog numbers in NVM cells. For example, memristors have been shown to be suitable for implementing synaptic weights of neural networks [149, 150]. Such an implementation can make the hardware highly compact. However, it is known that NVM cells usually suffer from various types of noise [3, 22]. The accumulating noise in NVM cells will make the DNN weights less accurate, and degrade the DNN's performance. Therefore, a better balance between the storage redundancy for weights and the overall DNN performance needs to be achieved.

In this work, we explore a natural solution, which uses linear analog error correcting codes to protect the analog weights. We start by exploring the performance of DNNs under different levels of noise. We then study *systematic linear analog codes*. Linear analog codes map real or complex valued source data to real or complex valued codewords. Our work is an extension of the study of non-systematic linear analog codes in [151, 152], which analyzes the theoretical performance of linear analog codes and derives lower bounds for minimum squared error (MSE) under maximum likelihood (ML) decoding. We present a systematic linear code design, whose code rate is at most $1/2$. We then show that although the analog ECC does not fully correct noise, the DNN's performance can be improved substantially. Other alternative approaches are to use a *systematic non-linear analog code* and *digital error correcting codes* for binarized neural networks for error

correction, which has also been explored in [33].

7.2 Deterioration of DNN Performance with Noise

In this section, we study the performance of DNNs when the real-valued weights are susceptible to noise from the Additive White Gaussian Noise (AWGN) channel. In classification problems, the perfomance of the DNN is measured by their *accuracy*, i.e. the fraction of outputs that they classify correctly. We first explain the different datasets used in our experiments [47], and then show results for the effect of noise on DNNs' performance.

7.2.1 Datasets

We use the following datasets in our experiments :

1) MNIST: MNIST is a widely used dataset of 60000 images that has handwritten digits as input and the actual digit as the label. We use a trained convolutional neural network (CNN) with 1.2 million weights for the MNIST dataset.

2) CIFAR-10: CIFAR-10 is a dataset used for classification of an image into one of the ten classes (e.g. cats, dogs, airplanes etc.). We also use another trained CNN with around 1.2 million weights for the CIFAR-10 dataset.

3) IMDB: IMDB is a collection of 50000 short (text) movie reviews, where each one is labeled as positive or negative. We use a trained Long Short Term Memory (LSTM) network for the IMDB dataset, which has around 2.7 million weights.

The above datasets and DNNs are representative of a large class of deep learning applications.

7.2.2 Results

When larger and larger AWGN noise is added to the weights of a DNN, its performance (i.e. the classification accuracy for classification tasks) degrades. The experimental performance of DNNs with noise is shown in Fig. 7.1, Fig. 7.2 and Fig. 7.3. In those figures, the accuracy of DNNs is represented by the black curves (the other curves represent the performance of DNNs after error correction, which will be discussed later). (In the figures, the x-axis is the signal to noise ratio (SNR) in dB, defined as $10 \log_{10} \frac{\sigma_s}{\sigma_n}$, where σ_s is the variance of the weights, and σ_n is the variance

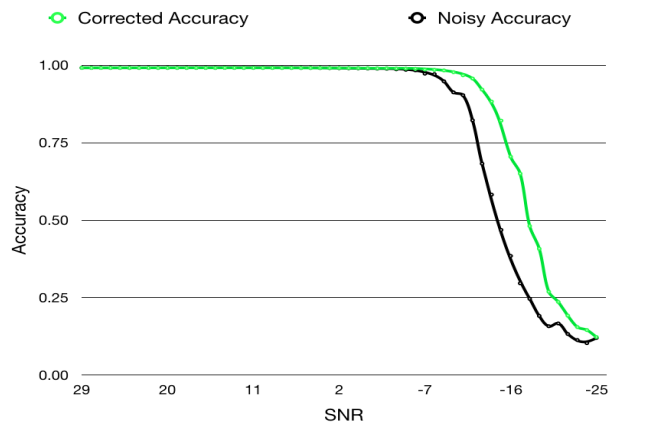

Figure 7.1: Classification accuracy (fraction of correct classification) vs Signal to Noise Ratio (SNR) in dB (high to low) for noisy weights (black lines) and weights corrected by linear analog codes (green lines) for MNIST dataset.

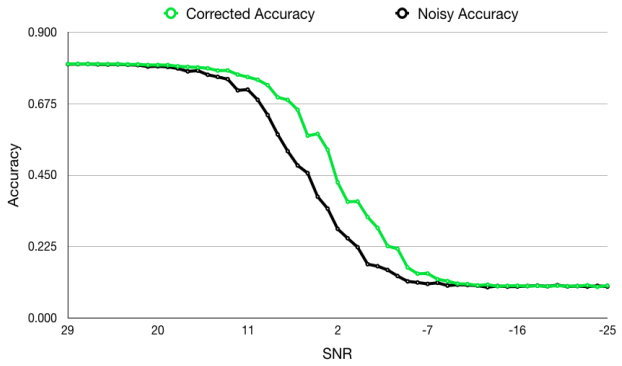

Figure 7.2: Accuracy vs SNR in dB(high to low) for noisy weights (black lines) and weights corrected by linear analog codes (green lines) for CIFAR-10 dataset.

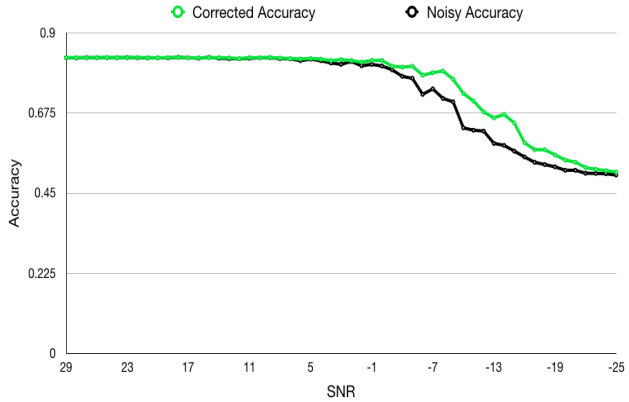

Figure 7.3: Accuracy vs SNR in dB(high to low) for noisy weights (black lines) and weights corrected by linear analog codes (green lines) for IMDB dataset.

of the noise. The y-axis is the classification accuracy)

It can be observed that when noise is quite small, DNNs have some natural tolerance for noise (namely, their performance degrades gracefully). However, when the noise exceeds a certain threshold the DNN's performance starts to drop sharply until it converges to some known value.

7.3 Analog Codes for Noisy DNN

In typical DNNs, the weights are real (i.e. analog) numbers. Each analog code converts a vector of analog symbols to a longer codeword for error correction. We use a systematic linear analog code in our experiments. In such systematic codes, the information symbols are still stored as before as network weights. The additional symbols (i.e. redundant symbols generated by the analog code) are stored in additional NVM cells. The hardware system can use either hardware or software to perform encoding and decoding.

7.3.1 Linear Analog Codes

In this subsection, we study an existing non-systematic linear analog code, and transform it into systematic form. We first present a brief overview of linear analog codes explored in [151]. A linear analog code converts a real vector of messages $\mathbf{u} \in \mathcal{R}^{K\times 1}$ to codeword vector $\mathbf{v} \in \mathcal{R}^{N\times 1}$, by the following transformation

$$\mathbf{v} = \mathbf{G}^T\mathbf{u}$$

Each coordinate u_i in $\mathbf{u}$ has mean 0 and variance D_u. $\mathbf{G}$ is called a $K \times N$ generator matrix. The codeword $\mathbf{v}$ passes through a channel, characterized by the noise vector $\mathbf{n} \in \mathcal{R}^{N\times 1}$, where each i.i.d. coordinate $n_i \sim N(0, \sigma_n^2)$. The received signal is given by

$$\mathbf{r} = \mathbf{v} + \mathbf{n}$$

The decoder takes the noisy message $\mathbf{r}$ to produce $\hat{\mathbf{u}}$, which is an estimate of $\mathbf{u}$ by the following equation

$$\hat{\mathbf{u}} = \mathbf{A}\mathbf{r}$$

where $\mathbf{A}$ is called the decoding matrix. For a ML decoder,

$$\mathbf{A} = (\mathbf{G}\mathbf{G}^T)^{-1}\mathbf{G}$$

$\mathbf{G}$ is designed to ensure that the energy per information symbol of the signal $\mathbf{v}$ is E_b. Also, for an ML decoder $\mathbf{A}$, the optimal $\mathbf{G}$ to reduce the mean squared error is given by

$$\mathbf{G}\mathbf{G}^T = diag\{\frac{E_b}{D_u}, \frac{E_b}{D_u}, ..., \frac{E_b}{D_u}\}$$

$\mathbf{G}$ can be formed by deleting $(N-K)$ rows of an *orthogonal matrix*, and scaling it appropriately by a factor $\sqrt{\frac{E_b}{D_u}}$.

We now transform the non-systematic analog code to a systematic form. Let us assume we have a vector $\mathbf{w}$ of K weights in a DNN, where weights w_i come from some distribution with mean μ and variance D_u. We first convert the vector $\mathbf{w}$ to the message vector $\mathbf{u}$ where

$$u_i = w_i - \mu$$

for $1 \leq i \leq K$. To obtain a systematic code, we would like $\mathbf{G}$ to be of the form $(\mathbf{I}_K \mid \mathbf{P})$ where $\mathbf{P}$ is a $K \times N-K$ matrix and $\mathbf{I}_K$ is a $(K \times K)$ identity matrix. The matrix $\mathbf{P}$ can be constructed as follows. $\mathbf{P}$ can be constructed by by deleting K rows of an $(N-K \times N-K)$ *orthogonal matrix*, and scaling it appropriately by a factor $\sqrt{\frac{E_b}{D_u} - 1}$, for $K \leq N-K$ and $E_b \geq D_u$. The following theorem can be proved:

Theorem 73. *A matrix* $\mathbf{G} = (\mathbf{I}_K \mid \mathbf{P})$ *constructed above satisfies the condition*

$$\mathbf{G}\mathbf{G}^T = diag\{\frac{E_b}{D_u}, \frac{E_b}{D_u}, ..., \frac{E_b}{D_u}\}$$

Proof.

$$\mathbf{G}\mathbf{G}^T = (\mathbf{I}_K \mid \mathbf{P})(\mathbf{I}_K \mid \mathbf{P})^T$$

$$= (\mathbf{I}_K \mid \mathbf{P})(\frac{\mathbf{I}_K}{\mathbf{P}^T}) = \mathbf{I}_K + \mathbf{P}\mathbf{P}^T = \mathbf{I}_K + (\frac{E_b}{D_u} - 1)\mathbf{I}_K = \frac{E_b}{D_u}\mathbf{I}_K$$

□

The matrix $\mathbf{G}$ in the above theorem can be used as a generating matrix to construct a systematic analog code. The matrix $\mathbf{G}$ satisfies the constraints on equation (13) of [151]. Consequently, it satisfies the optimality condition as well as the bound on performance in that paper.

7.3.2 Experimental Performance of Linear Analog Codes in DNNs

In this subsection, we show how analog codes can substantially improve the performance of DNNs. We measure the performance of these networks in terms of the classification accuracy. The x-axis denotes the signal to noise ratio (SNR) in dB, defined as $10\log_{10}\frac{\sigma_s}{\sigma_n}$, where σ_s is the variance of the codeword signal, and σ_n is the variance of the noise. The y-axis is the classification accuracy.

We first show the improvement in performance of different networks when when the weights are protected by linear analog codes. The experimental results are shown in Fig. 7.1, Fig. 7.2 and Fig. 7.3 as green curves. It can be seen that when SNR is high (namely, when noise is low) the accuracy of the networks degrades slowly, which implies that they have some inherent tolerance to noise. However, as the noise level exceeds a certain level, the accuracy of the networks begin to degrade substantially. Error correcting codes can be used to improve the overall performance effectively in this region. It can be seen from the figures that the linear analog codes used here (of rate 1/2) can improve the performance of the DNNs up to 5 dB (e.g. CNN for IMDB).

7.4 Conclusion

This section studies the performance of DNNs under noise. Systematic linear analog error correction codes are used to protect the analog weights of DNNs and improve the overall performance. The results in the paper can be deepened by exploring algorithms for optimal rate allocation of ECCs, the fundamental trade-off between a DNN's performance and the extra redundancy added to protect it, as well as the implementation of the schemes in hardware systems.

8. INTERNALLY CODED NEURAL NETWORKS

8.1 Introduction

Deep Neural Networks (DNNs) have become a dominating force in Artificial Intelligence (AI), bringing revolutions in science and technology. A massive amount of academic and industrial research is being devoted to implementing DNNs in hardware [12]. Hardware-implemented DNNs are appearing in phones, sensors, healthcare devices, and more, which will revolutionize every sector of society [20], and make AI systems increasingly energy-efficient and ubiquitous.

In parallel, DNNs are known to be highly susceptible to adversarial interventions. In a recent line of works which followed [153], it was shown that by adding a small (and often indistinguishable to humans) amount of noise to the *inputs* of a DNN, one can cause it to reach nonsensical conclusions. More recently, it was shown [154] that in some DNN architectures one can attain similar effects by changing as little as one or two entries of the input. This reveals an orthogonal concern of a similar nature from the adversarial machine learning perspective: performance degradation due to malicious attacks.

There exists a rich body of research which studies how to make DNNs robust to noise. This includes noise that is injected into the neurons/synapses, or into the inputs. Even though computation under noise has been studied since the 1950's [155], solutions have been almost exclusively heuristic.

To combat adversarial attacks to the inputs, much focus was given on adjusting the *training* process to produce more robust DNNs, e.g., by adjusting the regularization expression [156], or the loss function [157]. These approaches usually involve intractable optimization problems, and succeed insofar as the underlying optimization succeeds.

Combating noise in neurons/synapses has also enjoyed a recent surge of interest [158], which builds upon the previous wave of interest in DNNs in the early 1990's [159]. Most of this line of research focuses on replication methods (called augmentation), retraining, and providing statistical

frameworks for testing fault tolerance of DNNs (e.g., training a DNN to remember a coded version of all possible outputs [160]). It is also worth mentioning that to a certain degree, DNNs tend to present some natural fault-tolerance without any intervention. This phenomenon is conjectured to be connected to *over-provisioning* [161], i.e., the fact that in most cases one uses more neurons than necessary, but rigorous guarantees remain elusive.

In this work, we present a new scheme for robust DNNs called *Coded Deep Neural Network* (CodNN). It transforms the internal structure of DNNs by adding redundant neurons and edges to increase its reliability. The added redundancy can be seen as a new type of error-correcting codes customized for machine learning.

8.2 Construction of Coded Neural Networks

Consider a DNN, which usually has many layers of neurons. Consider two groups of neurons in two adjacent layers, as shown in Fig. 8.1a. The outputs of the n neurons in the $(\ell-1)$-th layer $v_{\ell-1,1}, v_{\ell-1,2}, \cdots, v_{\ell-1,n}$ are transmitted via the edges to the k neurons in the ℓ-th layer $v_{\ell,1}, v_{\ell,2}, \cdots, v_{\ell,k}$ (multiplied by the edge weights in the process), where they are summed and passed through activation function $\sigma(\cdot)$ to become the k outputs.

When errors/erasures appear in the edge weights (e.g., because of errors/erasures in the memory cells that store the weights, or due to circuit faults), the values passed to the k neurons $v_{\ell,1}, v_{\ell,2}, \cdots, v_{\ell,k}$ will be erroneous and their k outputs can be wrong. To make the DNN more fault tolerant, we transform the architecture to a new form, as shown in Fig. 8.1b, where a new middle layer is added. The middle layer has m neurons, whose m outputs are a coded version of the n inputs from the previous layer. One important way to use this new architecture is to make the m outputs a codeword of the n inputs with redundancy for fault tolerance; then the m outputs are transmitted to the next layer of k neurons.

The fault-tolerance performance of the architecture in Fig. 8.1a and 8.1b can be defined as follows. Assume that the k neurons $v_{\ell,1}, v_{\ell,2}, \cdots, v_{\ell,k}$ perform a classification task, and each of their outputs $z_1, z_2, \cdots, z_k$ takes its value in the set $\{-1, 1\}$. Let $\mathcal{X}$ denote the set of values that the inputs $\mathbf{x} = (x_1, x_2, \cdots, x_n)$ can take with positive probability (the definitions here can be

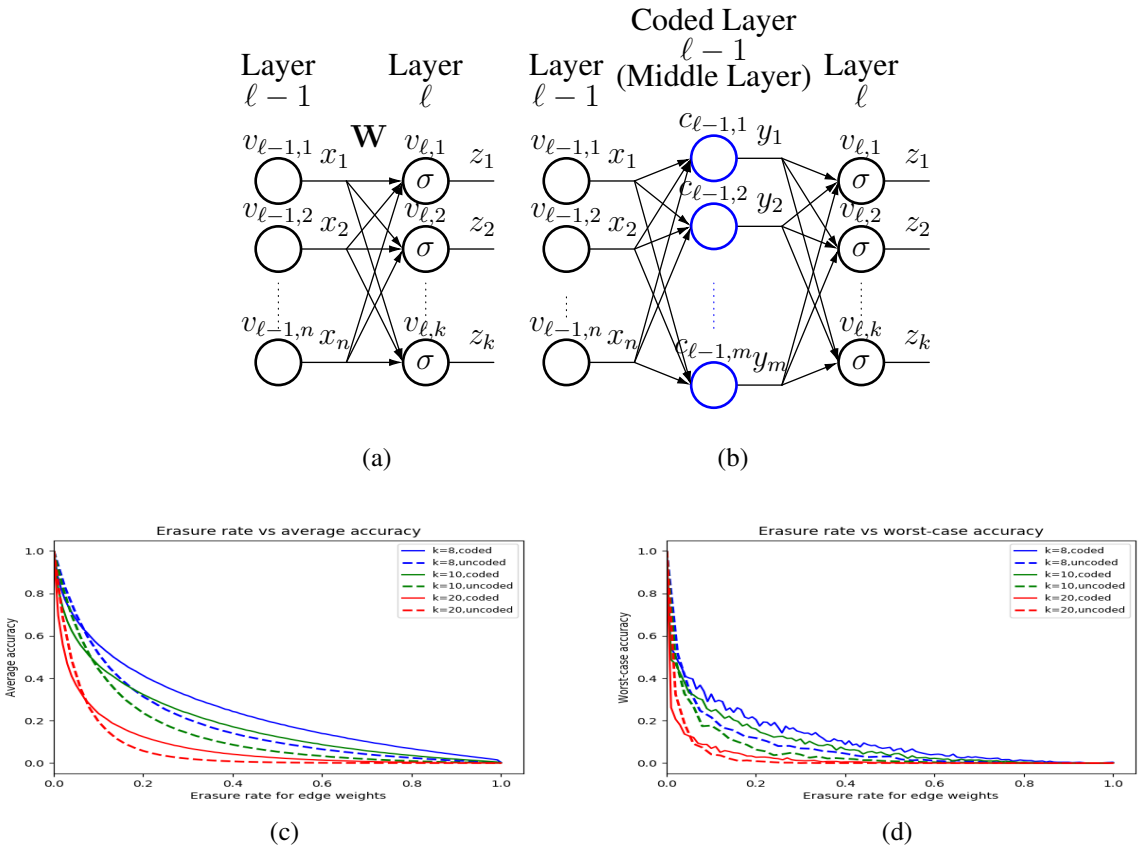

Figure 8.1: (a)-(b) An illustration of the Coded Deep Neural Network (CodNN) scheme, where (a) shows neurons in two adjacent layers, and (b) shows a CodNN scheme that adds a new middle layer (which is a coded version of its previous layer). (c)-(d) Improvement in *average accuracy* and *worst-case accuracy* by using CodNN schemes. Here the input layer has $n = 10$ neurons, the middle layer has $m = 20$ neurons, and the output layer has k neurons (for $k = 8, 10, 20$). The solid curves (denoted by "coded") are for CodNN, and the dashed curves (denoted by "uncoded") are for the original neural network component.

extended from discrete cases to continuous cases).

Let $\mathcal{E}$ denote the set of error patterns for the edge weights. Let $f(\mathbf{x})$ denote the correct outputs of the neurons $v_{l,1}, \cdots, v_{\ell,k}$ when the edge weights have no errors; and let $\bar{f}(\mathbf{x}, e)$ denote their outputs when the edge weights have the error pattern $e \in \mathcal{E}$. Let $\mathbb{1}(a, b)$ be the indicator function, which equals 1 if $a = b$ and 0 otherwise. We define two metrics for fault-tolerance performance as follows:

(1) Define the *average accuracy* as

$$\mathbb{E}_{\mathbf{x}\in\mathcal{X},e\in\mathcal{E}}\left[\mathbb{1}(f(\mathbf{x}),\bar{f}(\mathbf{x},e))\right],$$

where $\mathbb{E}(\cdot)$ denotes the expectation of a random variable.

(2) Define the *worst-case accuracy* as

$$\min_{e\in\mathcal{E}}\left\{\mathbb{E}_{\mathbf{x}\in\mathcal{X}}\left[\mathbb{1}(f(\mathbf{x}),\bar{f}(\mathbf{x},e))\right]\right\}$$

.

We present designs and analysis for CodNN next.

8.3 Coded Neural Network Construction By Analog Codes

In this section, we present a construction for CodNN based on analog error-correcting codes [136]. Let $\mathbf{W} = (w_{ij})_{n\times k}$ be the weight matrix for Fig. 8.1a, whose elements are the $n \times k$ edge weights. Let $\mathbf{G} = (g_{ij})_{n\times m}$ be a matrix constructed as follows: from a randomized $m \times m$ orthogonal matrix, delete any $m - n$ rows, then multiply the matrix by a scaling factor $\sqrt{2}$. Let $\mathbf{A} = (a_{ij})_{n\times m} = (\mathbf{G}\mathbf{G}^T)^{-1}\mathbf{G}$. Then, in the fault-tolerant architecture in Fig. 8.1b, we let $(y_1, y_2, \cdots, y_m) = (x_1, x_2, \cdots, x_n)\mathbf{G}$, and let $(z_1, z_2, \cdots, z_k) = \sigma((y_1, y_2, \cdots, y_m)\mathbf{A}^T\mathbf{W})$, where $\sigma(\cdot)$ is the activation function.

The coded layer $(y_1, y_2, \cdots, y_m)$ is an analog error correcting code of $(x_1, x_2, \cdots, x_n)$, which contains redundancy and can help correct errors/erasures. As an important case, consider erasures for edge weights, which can be caused by stuck-at errors in memories, by circuit disconnections or signal delay, *etc.* An edge-weight erasure can be seen as changing the edge weight to 0, thus removing the edge from computation.

Let the error model be i.i.d. edge-weight erasures, with erasure probability $p \in [0, 1]$. Let the activation functions of the k output neurons be $\mathrm{sign}(\cdot)$. We study how the accuracies of the network in Fig. 8.1a and 8.1b change with the erasure probability p. Assume that the n inputs $x_1, x_2, \cdots, x_n$

are real values with a uniform distribution in the range $[-1, 1]$. The results (based on extensive experiments) are shown in Fig. 8.1c and 8.1d . It can be seen that for a wide range of p, for both average and worst-case accuracies, the CodNN construction outperforms the original neural network significantly.

8.4 Related Work

In addition to these results, work by Raviv *et al*. [34] has looked into this problem in a more fundamental way for binarized inputs and real-valued weights. In this work, a tight connection between ℓ_1 metric and binary classification is studied. This paper also explores solutions based on Fourier analysis and replication. The most interesting result is the proof that a well-known parity code can guarantee successful classification under noise. These results are applicable to a large family of DNNs. The work in this section and the above paper are an extension of works on *coded computation* (typically with applications to distribution systems) to neural computation.

8.5 Conclusion

In this section, we studied a novel approach for combating noise in DNNs with error correcting codes using analog error correction codes, and shown that coding can improve both the average and worst case performance. In addition to the fundamental work as shown in the paper [34], extension of those results to more generic activation functions, and construction of coded neural networks using finite frame theory need to be studied more deeply.

9. CONCLUSION AND FUTURE WORK

This dissertation looks at some practical applications in the area of error correction for data storage and uses ideas at the intersection of information theory and machine learning to suggest effective solutions. Machine learning techniques, which can supplement existing techniques that use error-correcting codes, have been used for enhanced error correction. On the other hand, we use ideas from information theory and error-correcting codes to design robust neural networks, which perform reliable computation in the presence of noise.

A lot of other interesting areas exist at the intersection of machine learning and information theory. Existing results, including our own, seem like the tip of the iceberg. In the future, some interesting problems in the broader intersection of machine learning and information theory are discussed below :

- *Robust Neural Network Design:* Hardware implementation of neural networks is a key research area. However, hardware systems inevitably suffer from reliability issues. The devices which store weights and activations of neural networks are susceptible to various kinds of noise and faults. In addition, there might be adversarial attacks, in which minor perturbations on the input or the model of neural networks make them yield absurd results. These attacks have been studied extensively, but there is still a need to understand these phenomena theoretically and come up with better strategies to mitigate such adversarial noise. For example, different weights of a neural network have a disproportionate effect on its performance [162]. Ideas from information theory can be useful in studying this asymmetric behavior and use it to design more robust neural networks. Furthermore, areas such as robust training of neural networks also require further research.

- *Optimal Neural Network Design:*One useful follow-up to designing robust neural networks would be to design networks and training algorithms which satisfy multiple constraints such as robustness to random and adversarial noise, reduced storage space, the ability to learn

new tasks in the future, *etc* without significant reduction in performance. These methods are similar in the sense that information-theoretic methods can be used to find the asymmetric importance of various weights with respect to different optimization goals, and the neural network design can be optimized accordingly. For example, model compression and robustness are can be seen as complementary approaches. Model compression might lead to smaller DNN models, but such networks are also less tolerant of noise as compared to over-provisioned networks. Therefore, there is a need to protect the most important weights while doing away with less-important weights. Thus, after compression, some redundancy has to be added in a systematic manner to improve the performance of DNNs in the presence of noise.

- *Learning from Small Data:* One disadvantage of deep neural networks is that they usually require large amounts of data to train. However, in many critical applications, such data is not available. In areas such as healthcare, there are a lot of records that might not be voluminous but carry very useful information. Many useful problems in day-to-day life can be learned using less number of data samples. It also helps in cases where the neural network is not expected to have critically high levels of precision. Moreover, in many applications, generating labels might not be easy, or labels might be missing. Ideas such as few-shot (including one-shot and zero-shot) learning, representation learning, *etc.* have been proposed. However, there is a need to look into this problem with an information-theoretic lens. Novel methods, which look into the derivation of approximate sufficient statistics from data, can be explored.

REFERENCES

[1] A. Jiang, Y. Li, and J. Bruck, "Error correction through language processing," in *Proc. IEEE Information Theory Workshop (ITW)*, 2015.

[2] K. Fukuda, Y. Shimizu, K. Amemiya, M. Kamoshida, and C. Hu, "Random telegraph noise in flash memories-model and technology scaling," in *Proc. IEEE International Electron Devices Meeting*, pp. 169–172, IEEE, 2007.

[3] Q. Li, A. Jiang, and E. F. Haratsch, "Noise modeling and capacity analysis for NAND flash memories," in *Proc. IEEE International Symposium on Information Theory (ISIT)*, 2014.

[4] A. P. Ferreira, M. Zhou, S. Bock, B. Childers, R. Melhem, and D. Mossé, "Increasing PCM main memory lifetime," in *Proc. Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 914–919, IEEE, 2010.

[5] A. Jiang and J. Bruck, "Joint coding for flash memory storage," in *Proc. IEEE International Symposium on Information Theory*, pp. 1741–1745, IEEE, 2008.

[6] A. Jiang and J. Bruck, "Information representation and coding for flash memories," in *Proc. IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, pp. 920–925, IEEE, 2009.

[7] S. Vembu, S. Verdu, and Y. Steinberg, "The source-channel separation theorem revisited," *IEEE Transactions on Information Theory*, vol. 41, no. 1, pp. 44–54, 1995.

[8] C. E. Shannon, "Prediction and entropy of printed English," *Bell System Technical Journal*, vol. 30, no. 1, pp. 50–64, 1951.

[9] M. Li, W. Zuo, S. Gu, D. Zhao, and D. Zhang, "Learning convolutional networks for content-weighted image compression," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3214–3223, 2018.

[10] A. Prakash, N. Moran, S. Garber, A. DiLillo, and J. Storer, "Semantic perceptual image compression using deep convolution networks," in *Proc. Data Compression Conference (DCC)*, pp. 250–259, IEEE, 2017.

[11] M. N. Bojnordi and E. Ipek, "Memristive boltzmann machine: A hardware accelerator for combinatorial optimization and deep learning," in *Proc. IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 1–13, IEEE, 2016.

[12] K. Guo, S. Han, S. Yao, Y. Wang, Y. Xie, and H. Yang, "Software-hardware codesign for efficient neural network acceleration," *IEEE Micro*, vol. 37, no. 2, pp. 18–25, 2017.

[13] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," in *Proc. International Conference on Learning Representations*, 2016.

[14] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and < 0.5 MB model size," *arXiv preprint arXiv:1602.07360*, 2016.

[15] Y. Lin and J. R. Cavallaro, "Energy-efficient convolutional neural networks via statistical error compensated near threshold computing," in *Proc. IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, IEEE, 2018.

[16] X. Liu, M. Cheng, H. Zhang, and C.-J. Hsieh, "Towards robust neural networks via random self-ensemble," in *Proc. European Conference on Computer Vision (ECCV)*, pp. 369–385, 2018.

[17] Y. Wang, J. Shen, T.-K. Hu, P. Xu, T. Nguyen, R. Baraniuk, Z. Wang, and Y. Lin, "Dual dynamic inference: Enabling more efficient, adaptive and controllable deep inference," *arXiv preprint arXiv:1907.04523*, 2019.

[18] Y. Zhang, X. Wang, and E. G. Friedman, "Memristor-based circuit design for multilayer neural networks," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 2, pp. 677–686, 2017.

[19] M. Chincoli and A. Liotta, "Self-learning power control in wireless sensor networks," *Sensors*, vol. 18, no. 2, p. 375, 2018.

[20] M. Mohammadi, A. Al-Fuqaha, S. Sorour, and M. Guizani, "Deep learning for IoT big data and streaming analytics: A survey," *IEEE Communications Surveys & Tutorials*, vol. 20,

no. 4, pp. 2923–2960, 2018.

[21] B. Feinberg, S. Wang, and E. Ipek, "Making memristive neural network accelerators reliable," in *Proc. IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 52–65, IEEE, 2018.

[22] P. S. Georgiou, I. Köymen, and E. M. Drakakis, "Noise properties of ideal memristors," in *Proc. IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1146–1149, IEEE, 2015.

[23] P. Pouyan, E. Amat, and A. Rubio, "Reliability challenges in design of memristive memories," in *Proc. 5th European Workshop on CMOS Variability (VARI)*, pp. 1–6, IEEE, 2014.

[24] I. Vourkas, D. Stathis, G. C. Sirakoulis, and S. Hamdioui, "Alternative architectures toward reliable memristive crossbar memories," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 1, pp. 206–217, 2015.

[25] Y. Cassuto, S. Kvatinsky, and E. Yaakobi, "Sneak-path constraints in memristor crossbar arrays," in *Proc. IEEE International Symposium on Information Theory*, pp. 156–160, IEEE, 2013.

[26] M. Stanisavljevic, A. Athmanathan, N. Papandreou, H. Pozidis, and E. Eleftheriou, "Phase-change memory: Feasibility of reliable multilevel-cell storage and retention at elevated temperatures," in *Proc. IEEE International Reliability Physics Symposium*, pp. 5B–6, IEEE, 2015.

[27] D. B. Strukov and R. S. Williams, "Exponential ionic drift: fast switching and low volatility ofáthin-film memristors," *Applied Physics A*, vol. 94, no. 3, pp. 515–519, 2009.

[28] A. Jiang, P. Upadhyaya, E. F. Haratsch, and J. Bruck, "Error correction by natural redundancy for long term storage," in *Proc. Non-Volatile Memories Workshop (NVMW)*, 2017.

[29] A. Jiang, P. Upadhyaya, E. F. Haratsch, and J. Bruck, "Correcting errors by natural redundancy," in *Proc. Information Theory and Applications Workshop (ITA)*, pp. 1–8, IEEE, 2017.

[30] P. Upadhyaya and A. Jiang, "On LDPC decoding with natural redundancy," in *Proc. 55th*

Allerton Conference on Communication, Control and Computing, pp. 680–687, 2017.

[31] A. A. Jiang, P. Upadhyaya, Y. Wang, K. R. Narayanan, H. Zhou, J. Sima, and J. Bruck, "Stopping set elimination for LDPC codes," in *Proc. 55th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 700–707, IEEE, 2017.

[32] P. Upadhyaya and A. A. Jiang, "Representation-oblivious error correction by natural redundancy," in *Proc. IEEE International Conference on Communications (ICC)*, pp. 1–7, IEEE, 2019.

[33] P. Upadhyaya, X. Yu, J. Mink, J. Cordero, P. Parmar, and A. Jiang, "Error correction for noisy neural networks," in *Proc. Information Theory and Applications Workshop (ITA)*, 2019.

[34] N. Raviv, S. Jain, P. Upadhyaya, J. Bruck, and A. A. Jiang, "Codnn–robust neural networks from coded classification," in *Proc. International Symposium on Information Theory (ISIT)*, IEEE, 2020.

[35] N. Raviv, P. Upadhyaya, S. Jain, J. Bruck, and A. A. Jiang, "Coded deep neural networks for robust neural computation," in *Proc. Non Volatile Memories Workshop (NVMW)*, 2020.

[36] D. MacKay, *Information theory, Inference and Learning Algorithms*. Cambridge University Press, Cambridge, UK, 2003.

[37] C. E. Shannon, "Xxii. programming a computer for playing chess," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 41, no. 314, pp. 256–275, 1950.

[38] N. Wiener, *Cybernetics or Control and Communication in the Animal and the Machine*. MIT press, Cambridge, MA, USA, 1948.

[39] J. Bruck and M. Blaum, "Neural networks, error-correcting codes, and polynomials over the binary n-cube," *IEEE Transactions on Information Theory*, vol. 35, no. 5, pp. 976–987, 1989.

[40] J. Bruck and J. W. Goodman, "A generalized convergence theorem for neural networks," *IEEE Transactions on Information Theory*, vol. 34, no. 5, pp. 1089–1092, 1988.

[41] S. S. Venkatesh and D. Psaltis, "Linear and logarithmic capacities in associative neural networks," *IEEE Transactions on Information Theory*, vol. 35, no. 3, pp. 558–568, 1989.

[42] M. Bichsel and P. Seitz, "Minimum class entropy: A maximum information approach to layered networks," *Neural Networks*, vol. 2, no. 2, pp. 133–141, 1989.

[43] N. Murata, S. Yoshizawa, and S. I. Amari, "Network information criterion-determining the number of hidden units for an artificial neural network model," *IEEE Transactions on Neural Networks*, vol. 5, no. 6, pp. 865–872, 1994.

[44] I. Kanter, "Information theory of a multilayer neural network with discrete weights," *EPL (Europhysics Letters)*, vol. 17, no. 2, p. 181, 1992.

[45] A. Borst and F. E. Theunissen, "Information theory and neural coding," *Nature Neuroscience*, vol. 2, no. 11, pp. 947–957, 1999.

[46] A. G. Hoffmann, "General limitations on machine learning.," in *Proc. European Conference on Artificial Intelligence (ECAI)*, pp. 345–347, 1990.

[47] F. Chollet, *Deep Learning with Python*. Manning Publications Co, Shelter Island, NY, USA, 2017.

[48] P. A. Estevez, M. Tesmer, C. A. Perez, and J. M. Zurada, "Normalized mutual information feature selection," *IEEE Transactions on Neural Networks*, vol. 20, no. 2, pp. 189–201, 2009.

[49] C. Doersch, "Tutorial on variational autoencoders," *arXiv preprint arXiv:1606.05908*, 2016.

[50] W. Xu, X. Tan, Y. Lin, X. You, C. Zhang, and Y. Be'ery, "On the efficient design of neural networks in communication systems," in *Proc. 53rd Asilomar Conference on Signals, Systems, and Computers*, pp. 522–526, IEEE, 2019.

[51] S. Zheng, S. Chen, and X. Yang, "Deepreceiver: A deep learning-based intelligent receiver for wireless communications in the physical layer," *arXiv preprint arXiv:2003.14124*, 2020.

[52] M. van Lier, A. Balatsoukas-Stimming, H. Corporaaal, and Z. Zivkovic, "Optcomnet: Optimized neural networks for low-complexity channel estimation," *arXiv preprint arXiv:2002.10493*, 2020.

[53] S. Park, O. Simeone, and J. Kang, "End-to-end fast training of communication links without a channel model via online meta-learning," *arXiv preprint arXiv:2003.01479*, 2020.

[54] S. Dörner, M. Henninger, S. Cammerer, and S. t. Brink, "WGAN-based autoencoder training over-the-air," *arXiv preprint arXiv:2003.02744*, 2020.

[55] C. T. Leung, R. V. Bhat, and M. Motani, "Low-latency neural decoders for linear and non-linear block codes," in *Proc. IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, IEEE, 2019.

[56] W. Zhang, S. Zhou, and Y. Liu, "Iterative soft decoding of reed-solomon codes based on deep learning," *IEEE Communications Letters (Early Access)*, pp. 1–1, 2020.

[57] Y. Wei, M.-M. Zhao, M.-J. Zhao, and M. Lei, "ADMM-based decoder for binary linear codes aided by deep learning," *IEEE Communications Letters*, 2020.

[58] X. Xiao, B. Vasić, R. Tandon, and S. Lin, "Designing finite alphabet iterative decoders of ldpc codes via recurrent quantized neural networks," *IEEE Transactions on Communications*, 2020.

[59] L. Li, G. Yu, J. Xu, and L. Li, "Channel decoding based on complex-valued convolutional neural networks," in *Proc. 2nd 6G Wireless Summit (6G SUMMIT)*, pp. 1–5, IEEE, 2020.

[60] M. Lian, F. Carpi, C. Häger, and H. D. Pfister, "Reinforcement learning for channel coding," in *Proc. International Zurich Seminar on Information and Communication (IZS)*, p. 89, ETH Zurich, 2020.

[61] A. Dhok and S. Bhole, "ATRNN: Using seq2seq approach for decoding polar codes," in *Proc. International Conference on COMmunication Systems & NETworkS (COMSNETS)*, pp. 662–665, IEEE, 2020.

[62] Y. Qin and F. Liu, "Convolutional neural network-based polar decoding," in *Proc. 2nd World Symposium on Communication Engineering (WSCE)*, pp. 189–194, IEEE, 2019.

[63] M. Benammar and P. Piantanida, "On robust deep neural decoders," in *Proc. 53rd Asilomar Conference on Signals, Systems, and Computers*, pp. 527–531, IEEE, 2019.

[64] C. Teng and Y. Chen, "Syndrome enabled unsupervised learning for neural network based

polar decoder and jointly optimized blind equalizer," *Journal on Emerging and Selected Topics in Circuits and Systems*, pp. 1–1, 2020.

[65] H. Lee, E. Y. Seo, H. Ju, and S.-H. Kim, "On training neural network decoders of rate compatible polar codes via transfer learning," *Entropy*, vol. 22, no. 5, p. 496, 2020.

[66] E. Balevi and J. G. Andrews, "Deep learning-based encoder for one-bit quantization," in *2019 IEEE Global Communications Conference (GLOBECOM)*, pp. 1–6, IEEE, 2019.

[67] Y. Jiang, H. Kim, H. Asnani, S. Oh, S. Kannan, and P. Viswanath, "Feedback turbo autoencoder," in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 8559–8563, IEEE, 2020.

[68] K. Vedula, R. Paffenroth, and D. R. Brown, "Joint coding and modulation in the ultra-short blocklength regime for Bernoulli-Gaussian impulsive noise channels using autoencoders," in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5065–5069, IEEE, 2020.

[69] Z. Zhang, D. Yao, L. Xiong, B. Ai, and S. Guo, "A convolutional neural network decoder for convolutional codes," in *Proc. International Conference on Communications and Networking in China*, pp. 113–125, Springer, 2019.

[70] N. Farsad, N. Shlezinger, A. J. Goldsmith, and Y. C. Eldar, "Data-driven symbol detection via model-based machine learning," *arXiv preprint arXiv:2002.07806*, 2020.

[71] N. Katz, "Communet: U-net decoder for convolutional codes in communication," *arXiv preprint arXiv:2004.10057*, 2020.

[72] J. Shen, A. Aboutaleb, K. Sivakumar, B. J. Belzer, K. S. Chan, and A. James, "Deep neural network a posteriori probability detector for two-dimensional magnetic recording," *IEEE Transactions on Magnetics*, vol. 56, no. 6, pp. 1–12, 2020.

[73] P. Henarejos and M. Ángel Vázquez, "Decoding 5G-NR communications via deep learning," in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 3782–3786, 2020.

[74] M. E. Morocho-Cayamcela, J. N. Njoku, J. Park, and W. Lim, "Learning to communicate

with autoencoders: Rethinking wireless systems with deep learning," in *Proc. International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, pp. 308–311, IEEE, 2020.

[75] D. Burth Kurka and D. Gündüz, "Joint source-channel coding of images with (not very) deep learning," in *Proc. International Zurich Seminar on Information and Communication (IZS 2020)*, pp. 90–94, ETH Zurich, 2020.

[76] K. Ullrich, F. Viola, and D. J. Rezende, "Neural communication systems with bandwidth-limited channel," *arXiv preprint arXiv:2003.13367*, 2020.

[77] F. A. Aoudia and J. Hoydis, "Joint learning of probabilistic and geometric shaping for coded modulation systems," *arXiv preprint arXiv:2004.05062*, 2020.

[78] D. Mu, W. Meng, S. Zhao, and X. Wang, "UAV intelligent optical communication based on conditional generation against network," *IOP Conference Series: Materials Science and Engineering*, vol. 768, p. 072022, mar 2020.

[79] D.-D. Le, D.-P. Nguyen, T.-H. Tran, and Y. Nakashima, "Run-length limited decoding for visible light communications: A deep learning approach," in *Proc. 25th Asia-Pacific Conference on Communications (APCC)*, pp. 496–501, IEEE, 2019.

[80] J. Fang, M. Bi, S. Xiao, G. Yang, H. Yang, Z. Chen, Z. Liu, and W. Hu, "Neural network decoder of polar codes with tanh-based modified LLR over FSO turbulence channel," *Optics Express*, vol. 28, no. 2, pp. 1679–1689, 2020.

[81] V. G. Satorras and M. Welling, "Neural enhanced belief propagation on factor graphs," *arXiv preprint arXiv:2003.01998*, 2020.

[82] A. Askri, G. R. Othman, and H. Ghauch, "Counting lattice points in the sphere using deep neural networks," in *Proc. 53rd Asilomar Conference on Signals, Systems, and Computers*, pp. 2053–2057, 2019.

[83] J. Zhang, O. Simeone, Z. Cvetkovic, E. Abela, and M. Richardson, "Itene: Intrinsic transfer entropy neural estimator," *arXiv preprint arXiv:1912.07277*, 2019.

[84] S. Ali, W. Saad, N. Rajatheva, K. Chang, D. Steinbach, B. Sliwa, C. Wietfeld, K. Mei,

H. Shiri, H.-J. Zepernick, *et al.*, "6G white paper on machine learning in wireless communication networks," *arXiv preprint arXiv:2004.13875*, 2020.

[85] N. Tishby and N. Zaslavsky, "Deep learning and the information bottleneck principle," in *Proc. IEEE Information Theory Workshop (ITW)*, pp. 1–5, IEEE, 2015.

[86] R. Shwartz-Ziv and N. Tishby, "Opening the black box of deep neural networks via information," *arXiv preprint arXiv:1703.00810*, 2017.

[87] A. M. Saxe, Y. Bansal, J. Dapello, M. Advani, A. Kolchinsky, B. D. Tracey, and D. D. Cox, "On the information bottleneck theory of deep learning," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2019, no. 12, p. 124020, 2019.

[88] M. Gabrié, A. Manoel, C. Luneau, J. Barbier, N. Macris, F. Krzakala, and L. Zdeborová, "Entropy and mutual information in models of deep neural networks," in *Advances in Neural Information Processing Systems*, pp. 1821–1831, 2018.

[89] Z. Liao, T. Drummond, I. Reid, and G. Carneiro, "Approximate fisher information matrix to characterize the training of deep neural networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 1, pp. 15–26, 2020.

[90] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel, "Infogan: Interpretable representation learning by information maximizing generative adversarial nets," in *Advances in Neural Information Processing Systems*, pp. 2172–2180, 2016.

[91] Y. Roh, K. Lee, S. E. Whang, and C. Suh, "Fr-train: A mutual information-based approach to fair and robust training," *arXiv preprint arXiv:2002.10234*, 2020.

[92] R. V. Zarcone, J. H. Engel, S. B. Eryilmaz, W. Wan, S. Kim, M. BrightSky, C. Lam, H.-L. Lung, B. A. Olshausen, and H.-S. P. Wong, "Analog coding in emerging memory systems," *Scientific Reports*, vol. 10, no. 1, pp. 1–13, 2020.

[93] J. Kosaian, K. V. Rashmi, and S. Venkataraman, "Learning-based coded computation," *IEEE Journal on Selected Areas in Information Theory*, pp. 1–1, 2020.

[94] R. Bauer and J. Hagenauer, "On variable length codes for iterative source/channel decoding," *Proceedings of Data Compression Conference*, pp. 273–282, 2001.

[95] M. Fresia and G. Caire, "Combined error protection and compression with turbo codes for image transmission using a JPEG2000-like architecture," in *Proc. International Conference on Image Processing (ICIP)*, pp. 821–824, 2006.

[96] L. Guivarch, J. Carlach, and P. Siohan, "Joint source-channel soft decoding of Huffman codes with Turbo-codes," *Proceedings of Data Compression Conference (DCC)*, pp. 83–92, 2000.

[97] J. Hagenauer, "Source-controlled channel decoding," *IEEE Transactions on Communications*, vol. 43, no. 9, pp. 2449–2457, 1995.

[98] M. Jeanne, J. Carlach, and P. Siohan, "Joint source-channel decoding of variable-length codes for convolutional codes and Turbo codes," *IEEE Transactions on Communications*, vol. 53, no. 1, pp. 10–15, 2005.

[99] A. N. Kim, S. Sesia, T. Ramstad, and G. Caire, "Combined error protection and compression using turbo codes for error resilient image transmission," in *Proc. International Conference on Image Processing (ICIP)*, 2005.

[100] Z. Peng, Y. Huang, and D. Costello, "Turbo codes for image transmission – a joint channel and source decoding approach," *IEEE Journal on Selected Areas in Communications (JSAC)*, vol. 18, no. 6, pp. 868–879, 2000.

[101] C. Poulliat, D. Declercq, C. Lamy-Bergot, and I. Fijalkow, "Analysis and optimization of irregular LDPC codes for joint source-channel decoding," *IEEE Communications Letters*, vol. 9, no. 12, pp. 1064–1066, 2005.

[102] L. Pu, Z. Wu, A. Bilgin, M. Marcellin, and B. Vasic, "LDPC-based iterative joint source-channel decoding for JPEG," *IEEE Transactions on Image Processing*, vol. 16, no. 2, pp. 577–581, 2007.

[103] L. Alvarez, P. Lions, and J. Morel, "Image selective smoothing and edge detection by nonlinear diffusion II," *SIAM Journal on Numerical Analysis*, vol. 29, no. 3, pp. 845–866, 1992.

[104] A. Buades, B. Coll, and J. Morel, "A review of image denoising algorithms, with a new one," *Multiscale Modeling and Simulation*, vol. 4, no. 2, pp. 490–530, 2005.

[105] P. Chatterjee and P. Milanfar, "Is denoising dead?," *IEEE Transactions on Image Processing*, vol. 19, no. 4, pp. 895–911, 2010.

[106] R. R. Coifman and D. L. Donoho, "Translation-invariant de-noising," in *Wavelets and Statistics*, pp. 125–150, Springer, 1995.

[107] M. Lindenbaum, M. Fischer, and A. Bruckstein, "On Gabor's contribution to image enhancement," *Pattern Recognition*, vol. 27, no. 1, pp. 1–8, 1994.

[108] E. Ordentlich, G. Seroussi, S. Verdu, and K. Viswanathan, "Universal algorithms for channel decoding of uncompressed sources," *IEEE Transactions on Information Theory*, vol. 54, pp. 2243–2262, May 2008.

[109] E. Ordentlich, G. Seroussi, S. Verdu, M. Weinberger, and T. Weissman, "A discrete universal denoiser and its application to binary images," in *Proc. International Conference on Image Processing (ICIP)*, vol. 1, pp. I–117, IEEE, 2003.

[110] L. Rudin, S. Osher, and E. Fatemi, "Nonlinear total variation based noise removal algorithms," *Physica D: Nonlinear Phenomena*, vol. 60, no. 1, pp. 259–268, 1992.

[111] L. Yaroslavsky and M. Eden, *Fundamentals of Digital Optics: Digital Signal Processing in Optics and Holography*. Springer, Verlag New York, 1996.

[112] Y. Li, Y. Wang, A. Jiang, and J. Bruck, "Content-assisted file decoding for nonvolatile memories," in *Proc. 46th Asilomar Conference on Signals, Systems and Computers*, pp. 937–941, 2012.

[113] J. Luo, Q. Huang, S. Wang, and Z. Wang, "Error control coding combined with content recognition," in *Proc. 8th International Conference on Wireless Communications and Signal Processing*, pp. 1–5, 2016.

[114] Y. Wang, M. Qin, K. R. Narayanan, A. Jiang, and Z. Bandic, "Joint source-channel decoding of polar codes for language-based sources," in *Proc. IEEE Global Communications Conference (Globecom)*, December 2016.

[115] D. Mackay, "Encyclopedia of sparse graph codes." `http://www.inference.org.uk/mackay/codes/data.html#l132`, 2015. (Accessed on 05/19/2020).

[116] J. Lin, "Vector quantization for image compression: Algorithms and performance," Ph.D. thesis, Brandeis University, 1992.

[117] J. Lin, J. Storer, and M. Cohn, "Optimal pruning for tree-structured vector quantization," *Information Processing and Management*, vol. 28, p. 6, 1992.

[118] M. Ruhl and H. Hartenstein, "Optimal fractal coding is NP-hard," in *Proc. Data Compression Conference (DCC)*, April 1997.

[119] J. Bruck and M. Naor, "The hardness of decoding linear codes with preprocessing," *IEEE Transactions on Information Theory*, vol. 36, pp. 381–385, March 1990.

[120] I. Dumer, D. Micciancio, and M. Sudan, "Hardness of approximating the minimum distance of a linear code," *IEEE Transactions on Information Theory*, vol. 49, pp. 22–37, January 2003.

[121] U. Feige and D. Micciancio, "The inapproximability of lattice and coding problems with preprocessing," *Journal of Computer and Systems Sciences*, vol. 69, pp. 45–67, August 2004.

[122] A. Vardy, "Algorithmic complexity in coding theory and the minimum distance problem," in *Proc. 29th ACM Symposium on Theory of Computing (STOC)*, (Texas), pp. 92–109, El Paso, May 1997.

[123] A. Vardy, "The intractability of computing the minimum distance of a code," *IEEE Transactions on Information Theory*, vol. 43, pp. 1757–1766, November 1997.

[124] J. Lansky, K. Chernik, and Z. Vlckova, "Syllable-based Burrows-Wheeler transform," in *Proc. Annual International Workshop on Databases,Texts, Specifications and Objects*, pp. 1–10, 2007.

[125] T. Weissman, E. Ordentlich, G. Seroussi, S. Verdu, and M. Weinberger, "Universal discrete denoising: Known channel," *IEEE Transactions on Information Theory*, vol. 51, no. 1, pp. 5–28, 2005.

[126] Y. Wang, K. R. Narayanan, and A. Jiang, "Exploiting source redundancy to improve the rate of polar codes," in *Proc. IEEE International Symposium on Information Theory (ISIT)*

(G. Aachen, ed.), June 2017.

[127] C. D. Manning and H. Schutze, *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, 1999.

[128] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *Proc. IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, 2010.

[129] G. Maral, M. Bousquet, and Z. Sun, *Satellite Communications Systems: Systems, Techniques and Technology*. Wiley, Hoboken, NJ, 5th ed., 2010.

[130] B. Addis, M. Di Summa, and A. Grosso, "Removing critical nodes from a graph: complexity results and polynomial algorithms for the case of bounded treewidth," *Optimization online (www. optmization-online. org)*, 2011.

[131] T. Fujito, "Approximating node-deletion problems for matrodial properties," *Journal of Algorithms*, vol. 31, pp. 211–227, 1999.

[132] M. Kumar, S. Mishra, N. S. Devi, and S. Saurabh, "Approximation algorithms for node deletion problems on bipartite graphs with finite forbidden subgraph characterization," *Theoretical Computer Science*, vol. 526, pp. 90–96, 2014.

[133] M. Yannakakis, "Node-deletion problems on bipartite graphs," *SIAM Journal on Computing*, vol. 10, pp. 310–327, May 1981.

[134] T. J. Schaefer, "The complexity of satisfiability problems," in *Proc. 10th Annual ACM Symposium on Theory of Computing (STOC)*, pp. 216–226, 1978.

[135] K. Cai, "Vertical constrained coding for phase-change memory with thermal crosstalk," in *International Conference on Computing, Networking and Communications*, 2014.

[136] J. Xie, L. Xu, and E. Chen, "Image denoising and inpainting with deep neural networks," *Advances in Neural Information Processing Systems*, pp. 341–349, 2012.

[137] Y. Ichiki, G. Song, K. Cai, S. Lu, and J. Cheng, "Neural network detection of ldpc-coded random access cdma systems," in *Proc. International Symposium on Information Theory and Its Applications*, (Singapore), 2018.

[138] M. Qin, C. Sun, and D. Vucinic, "Robustness of neural networks against storage media

errors," *arXiv preprint arXiv:1709.06173*, 2017.

[139] H. Kim, Y. Jiang, R. Rana, S. Kannan, S. Oh, and P. Viswanath, "Communication algorithms via deep learning," in *Proc. International Conference on Representation Learning (ICLR)*, (Vancouver), 2018.

[140] E. Nachmani, E. Marciano, L. Lugosch, W. J. Gross, D. Burshtein, and Y. Be'ery, "Deep learning methods for improved decoding of linear codes," *IEEE Journal of Selected Topics in Signal Processing*, vol. 12, no. 1, pp. 119–131, 2018.

[141] W. C. Calhoun and D. Coles, "Predicting the types of file fragments," *Digital Investigation*, vol. 5, no. supplement, pp. S14–S20, 2008.

[142] S. Fitzgerald, G. Mathews, C. Morris, and O. Zhulyn, "Using NLP techniques for file fragment classification," *Digital Investigation*, vol. 9, no. supplement, pp. S44–S49, 2012.

[143] M. Amirani, M. Toorani, and A. Beheshti, "A new approach to content-based file type detection," in *Proc. IEEE Symposium on Computers and Communications*, pp. 1103–1108, 2008.

[144] T. M. Cover and J. A. Thomas, *Elements of Information Theory*. Wiley, Hoboken, NJ, USA, 2nd ed., 2006.

[145] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3431–3440, 2015.

[146] P. Vincent, H. Larochelle, Y. Bengio, and P. A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *Journal of Machine Learning Research*, vol. 11, pp. 3371–3408, 2010.

[147] M. Zhang, K. Cai, Q. Huang, and S. Yuan, "On bit-level decoding of non-binary ldpc codes," *IEEE Transactions on Communications*, vol. 66, no. 9, pp. 3736–3748, 2018.

[148] S. Narayanan, A. Shafiee, and R. Balasubramonian, "Inxs: Bridging the throughput and energy gap for spiking neural networks," in *Proc. International Joint Conference on Neural Networks (IJCNN)*, pp. 2451–2459, 2017.

[149] P. Chi, S. Li, Z. Qi, P. Gu, C. Xu, T. Zhang, J. Zhao, Y. Liu, Y. Wang, and Y. Xie, "PRIME: A novel processing-in-memory architecture for neural network computation in ReRAM-based main memory," in *Proc. International Symposium on Computer Architecture (ISCA)*, 2016.

[150] H. Kim, M. P. Sah, C. Yang, T. Roska, and L. O. Chua, "Neural synaptic weighting with a pulse-based memristor circuit," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 59, no. 1, pp. 148–158, 2012.

[151] K. Xie, J. Li, and Y. Liu, "Analysis of performance of linear analog codes," *arXiv preprint arXiv:1511.05509*, 2015.

[152] K. Xie, J. Li, and Y. Liu, "Linear analog codes: The good and the bad," in *Proc. 46th Annual Conference on Information Sciences and Systems (CISS)*, pp. 1–6, IEEE, 2012.

[153] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.

[154] A. Shamir, I. Safran, E. Ronen, and O. Dunkelman, "A simple explanation for the existence of adversarial examples with small hamming distance," *arXiv preprint arXiv:1901.10861*, 2019.

[155] J. Von Neumann, "Probabilistic logics and the synthesis of reliable organisms from unreliable components," *Automata Studies*, vol. 34, pp. 43–98, 1956.

[156] I. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *International Conference on Learning Representations (ICLR)*, 2015.

[157] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," in *Proc. International Conference on Learning Representations (ICLR)*, 2018.

[158] C. Torres-Huitzil and B. Girau, "Fault and error tolerance in neural networks: A review," *IEEE Access*, vol. 5, pp. 17322–17341, 2017.

[159] D. S. Phatak and I. Koren, "Complete and partial fault tolerance of feedforward neural nets," *IEEE Transactions on Neural Networks*, vol. 6, no. 2, pp. 446–456, 1995.

[160] H. Ito and T. Yagi, "Fault tolerant design using error correcting code for multilayer neural

networks," *IEEE International Workshop on Defect and Fault Tolerance in VLSI Systems*, pp. 177–184, 1994.

[161] E. M. E. Mhamdi and R. Guerraoui, "When neurons fail," *arXiv preprint arXiv:1706.08884*, 2017.

[162] K. Huang, P. Siegel, and A. Jiang, "Functional error correction for robust neural networks," *arXiv preprint arXiv:2001.03814*, 2020.