LDPC CODES OVER LARGE ALPHABETS AND THEIR APPLICATIONS TO COMPRESSED SENSING AND FLASH MEMORY

A Dissertation

by

FAN ZHANG

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

DOCTOR OF PHILOSOPHY

August 2010

Major Subject: Electrical Engineering

LDPC CODES OVER LARGE ALPHABETS AND THEIR APPLICATIONS TO COMPRESSED SENSING AND FLASH MEMORY

A Dissertation

by

FAN ZHANG

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

DOCTOR OF PHILOSOPHY

Approved by:

Chair of Committee,	Henry D. Pfister
Committee Members,	Krishna R. Narayanan
	Sebastian Hoyos
	Guergana Petrova
Head of Department,	Costas N. Georghiades

August 2010

Major Subject: Electrical Engineering

ABSTRACT

LDPC Codes over Large Alphabets and Their
Applications to Compressed Sensing and Flash Memory. (August 2010)
Fan Zhang, B.S., University of Science and Technology of China;
M.S., University of Science and Technology of China
Chair of Advisory Committee: Prof. Henry D. Pfister

This dissertation is mainly focused on the analysis, design and optimization of Low-density parity-check (LDPC) codes over channels with large alphabet sets and the applications on compressed sensing (CS) and flash memories. Compared to belief-propagation (BP) decoding, verification-based (VB) decoding has significantly lower complexity and near optimal performance when the channel alphabet set is large. We analyze the verification-based decoding of LDPC codes over the q-ary symmetric channel (q-SC) and propose the list-message-passing (LMP) decoding which offers a good tradeoff between complexity and decoding threshold. We prove that LDPC codes with LMP decoding achieve the capacity of the q-SC when q and the block length go to infinity.

CS is a newly emerging area which is closely related to coding theory and information theory. CS deals with the sparse signal recovery problem with small number of linear measurements. One big challenge in CS literature is to reduce the number of measurements required to reconstruct the sparse signal. In this dissertation, we show that LDPC codes with verification-based decoding can be applied to CS system with surprisingly good performance and low complexity.

We also discuss modulation codes and error correcting codes (ECC's) design for flash memories. We design asymptotically optimal modulation codes and discuss their improvement by using the idea from load-balancing theory. We also design LDPC

codes over integer rings and fields with large alphabet sets for flash memories.

To my parents

ACKNOWLEDGMENTS

In my four years' stay in Texas A&M University, I have learned one thing - I could have never done any of this, particularly the research and the writing that went into this dissertation without the support and encouragement of a lot of people.

First, my foremost acknowledgement is to my advisor, Professor Henry Pfister. When I joined Texas A&M in 2006, Henry also joined the WCL that fall as a new faculty. I was so lucky to meet Henry and became his research assistant. Henry is the most important reason that my academic experience at TAMU was so enjoyable and fulfilling. Henry was always willing to help, encourage me and give valuable suggestions. I am always deeply impressed by his insightful thoughts, solid mathematical background, enthusiasm and persistence for solving complicated problems. Beyond the research problems we solved, Henry also taught me the knowledge and philosophy of being an independent researcher. I also would like to thank Henry for his generous help and support with looking for a job.

I am also grateful to Professor Krishna Narayanan for many valuable discussions on my research problems. Also, I would like to thank Krishna for his teaching of two courses, Information Theory and Advanced Channel Coding. His deep understanding on coding theory and information theory, and his innovative teaching style really made every lecture very interesting and unforgetable. I am impressed by his ability to express abstract and complicated concepts in a simple but precise way. Krishna also helped me a lot during my job hunting.

My sincere thanks to Professor Krishna Narayanan, Professor Sebastian Hoyos and Professor Guergana Petrova for being my committee members and helpful discussions on compressed sensing. Also I would like to thank Professor Ronald Devore for his wonderful course on compressed sensing. Ronald always described abstract prob-

lems in a precise and easy-to-understand way. His innovative and interactive style of teaching really made every class a happy journey. I would like to thank Professor Anxiao (Andrew) Jiang for his wonderful course on flash memories and many helpful discussions on the topic of ECC deisgn for flash memories. I also would like to thank Professor Jean-Francois Chamberland, Professor Tie Liu, Professor Shuguang Cui, Professor Costas Georghiades, Professor Zixiang Xiong, Professor Alex Sprintson, Professor Scott Miller and Professor Serap Savari for their wonderful courses and technical discussions on my research topics.

I would like to thank to all my colleagues Arvind Yedla, Wei-Yu Chen, Makesh Pravin Wilson, Zhuizhuan Yu, Chiawen Wang, Phong Sy Nguyen, Yung-Yih Jian, Byung Hak Kim, Lingjia Liu and many other names for their support and company. Their kindness made my four years at TAMU so memorable.

My special thanks to Paula Evans, Salim Rouyaheb, Jing Jiang, Tao Lang, Guodong Zhou and Yingning Wang for taking care of me in my first year at TAMU. They were always willing to help on everything. Jing also offered generous help during my job-hunting.

Finally, I would like to thank my parents, Jinlian Qiu and Rongmao Zhang and my sister Yuyun Qiu for their unconditional love and support. This work is dedicated to them.

TABLE OF CONTENTS

Page

Page

Page

Page

LIST OF TABLES

LIST OF FIGURES

FIGURE Page

FIGURE Page

CHAPTER I

INTRODUCTION

A. Introduction

The mathematical study of probability theory began in the 16th century. As a fundamental building block of modern science, probability theory has affected many areas in a very fundamental way, and also created many new areas of research. Shannon's seminal work on information theory is one of the most important examples.

There are two central topics in Shannon's original work on information theory: the compression and transmission of information [1]. Shannon formalizes these two problems and derives their ultimate limits. The goal of coding theory is to find practical schemes to achieve these two ultimate limits. Shannon's genius lies in several parts. First, Shannon proposed a schematic diagram of a general communication system. In the diagram, Shannon represents the communication system as the source, communication channel and the receiver. Second, Shannon started a revolutionary approach to the communication of information that revolutionized digital communications. If we transmit information using analog waveforms, as the signal is transmitted over long distance, it gets weaker. If we just simply amplify the analog signal with the noise, the signal becomes very noisy after a few stages.

Instead of dealing with the analog signal directly, Shannon showed that any information can be represented in binary digits and the receiver of a digital communication system can recover the binary digits with probability arbitrarily close to 1, if the amount of information transmitted per second is less than a limit. This limit is now called the Shannon limit, the Shannon capacity, or simply channel ca-

---

This dissertation follows the style of *IEEE Trans. on Information Theory.*

pacity. Shannon's work provides an ultimate limit for the information transmission rate, which means that no matter what method we use, the information rate cannot be more than Shannon capacity. Shannon shows, if we are allowed to jointly use the channel multiple times to transmit the same piece of information, the channel capacity is achievable. Unfortunately, Shannon's proof does not provide a practical way to achieve the channel capacity. In the past 60 years, researchers in channel coding theory have spent tremendous efforts to find practical schemes to achieve channel capacity using channel codes. The idea is, instead of transmitting the information itself, we also transmit some redundant information which can help the receiver combat the noise and recover the original information. The channel capacity gives the minimum fraction of redundancy needed for perfect recovery at the receiver.

The way how the redundancy is added and removed is call a channel code. In particular, the way how the transmitter adds the redundancy is called encoding and the way how the receiver recovers the information is called decoding. Channel coding theory is mainly focused on finding practically implementable channel codes which achieve the minimum fraction of redundancy, or channel capacity.

In the theoretical study of many information theoretic problems, choosing the channel code randomly usually simplifies the proof significantly. But the decoding complexity of random channel codes increases exponentially in the block-length of the code. To achieve channel capacity, the block-length must go to infinity. To make the codes practically implementable, the general idea is introduce some algebraic structure in the codewords instead of randomly generating them. By utilizing the structure of the codes, the codes can have encoders/decoders with complexity polynomial in the block-length and hopefully still achieve the channel capacity. For example, the structure of a linear code can be described by its generator matrix or parity-check matrix. Finding codes with both good structure and good error correcting capability

is always one of the most important problems in the research of coding theory.

There are two major frameworks when we consider code design: Hamming's framework and Shannon's framework. In Hamming's framework, the minimum Hamming distance, or the worst case error correction capability is considered. While in Shannon's framework, the average error correction capability is considered. Shannon's genius is to consider the whole communication system from a probabilistic viewpoint and to focus on the average performance. This characteristic of Shannon's information theory allows it to produce many beautiful results. For example, this allows strictly positive channel capacity even one requires the probability of decoding error to be arbitrarily small.

For most algebraic codes such as Hamming codes and Reed-Solomon (RS) codes, it is easy to analyze and optimize the performance of the codes in Hamming's framework. But they are usually sub-optimal in Shannon's framework. For example, RS codes, are maximum distance separable (MDS) codes that achieve the singleton bound. Therefore, they are optimum in Hamming's framework. But they are sub-optimal, or non-capacity-achieving, in Shannon's framework. On the other hand, low-density parity-check (LDPC) codes [4] are capacity-achieving over the binary-erasure channel (BEC) but they are not MDS codes [22]. Actually, there are no long MDS binary codes found so far. In this dissertation, we consider the performance of the codes using Shannon's framework.

It remains an open problem to find what structure allows a channel code to be capacity-achieving and encodable/decodable in polynomial complexity (in both time and memory). A further question would be, if the structure exists, is it unique? So far, two structures are shown to be potentially capacity-achieving or capacity-approaching. The first one is the trellis structure, such as the trellis structure of Turbo codes [2],[3]. The second structure is the tree-like graph structure such as

the bipartite graph of LDPC codes. The random interleaver in Turbo codes and the random permutation in the bipartite graph of LDPC codes actually give these codes some properties similar to random codes while still maintain a well structure of the codes. However, this brings significant difficulty to the analysis of these codes in the Hamming's framework. LDPC codes have been proved to be capacity-achieving over the BEC and suggested to be capacity-achieving over the q-ary symmetric channel (q-SC) [22],[14]. In this dissertation, we provide the proof for the capacity-achieving of the LDPC codes over the q-SC. Recently, Arikan shows that Polar codes are provably capacity-achieving over any symmetric binary-input discrete memoryless channel [16]. Polar code construction is based on a phenomenon called channel polarization. Interestingly, the channel polarization can also be represented by a tree structure.

B. Outline of Dissertation

In this dissertation, we mainly consider LDPC codes over large alphabet sets and their applications to compressed sensing and flash memories. In Chapter II, we consider verification-based decoding of LDPC codes over $GF(q)$. We discuss the difference of node-based and message-based verification-based decoding algorithms and we provide the differential equation analysis for the node-based algorithms. We also propose the list-message-passing (LMP) decoding which provides a smooth trade-off between complexity and performance of verification-based decoding. In the density evolution analysis of the LMP decoder, we show that LDPC codes are capacity-achieving over the q-SC when q and n go to infinity. The false-verification probability and error floors are also analyzed. In Chapter III, we consider the application of LDPC codes with verification-based decoding to compressed sensing systems. First, we show the connections between coding theory and compressed sensing problem. Then we discuss

how to apply LDPC codes with verification-based decoding algorithms to compressed sensing systems. It turns out that compressed sensing system based on LDPC codes with verification-based decoding only requires linear (in k) number of measurements to recover a k-sparse signal with high probability. The proof is based on the scaling law analysis of the density evolution equations and the stopping set analysis. In Chapter IV and Chapter V, we consider code design for flash memories. In Chapter IV, we discuss the modulation codes design for flash memories and in Chapter V, we design LDPC codes with verification-based decoding algorithms for flash memories. The conclusion is in Chapter VI.

CHAPTER II

VERIFICATION-BASED DECODING OF LDPC CODES OVER THE Q-SC

A. Introduction

Low-density parity-check (LDPC) codes are linear codes that were introduced by Gallager in 1962 [4] and re-discovered by MacKay in 1995 [5]. The ensemble of LDPC codes that we consider (e.g. see [22] and [6]) is defined by the edge degree distribution (d.d.) functions $\lambda(x) = \sum_{k\geq 2} \lambda_k x^{k-1}$ and $\rho(x) = \sum_{k\geq 2} \rho_k x^{k-1}$. The standard encoding and decoding algorithms are based on the bit-level operations. However, when applied to the transmission of data packets, it is natural to perform the encoding and decoding algorithm at the packet level rather than the bit level. For example, if we are going to transmit 32 bits as a packet, then we can use error-correcting codes over the, rather large, alphabet with 2^{32} elements.

Let Y be the r.v. which is the output of the the q-SC given the transmitted r.v. X. Then, the channel transition probability can be described as

$$\Pr(Y = y|X = x) = \begin{cases} 1-p & \text{if } x = y \\ p/(q-1) & \text{if } x \neq y \end{cases}$$

where x (resp. y) is the transmitted (resp. received) symbol and $x, y \in GF(q)$. The capacity of the q-SC is $1 + (1-p)\log_q(1-p) + p\log_q p - p\log_q(q-1)$ which is approximately equal to $1-p$ symbols per channel use for large q. This implies the number of symbols which can be reliably transmitted per channel use of the q-SC with large q is approximately equal to that of the BEC with erasure probability p. Moreover, the behavior of the q-SC with large q is similar to the BEC in the sense that: i) incorrectly received symbols from the q-SC provide almost no information about the transmitted symbol and ii) error detection (e.g., a CRC) can be added to

each symbol with negligible overhead [7].

Binary LDPC codes for the q-SC with moderate q are proposed and optimized based on EXIT charts in [8] and [9]. It is known that the complexity of the FFT-based belief-propagation algorithm for q-ary LDPC codes scales like $O(q \log q)$. Even for moderate sizes of q, such as $q = 256$, this renders such algorithms ineffective in practice. However, when q is large, an interesting effect can be used to facilitate decoding: if a symbol is received in error, then it is essentially a randomly chosen element of the alphabet, and the parity-check equations involving this symbol is very unlikely to be valid.

Based on this idea, Luby and Mitzenmacher develop an elegant algorithm for decoding LDPC codes on the q-SC for large q [10]. However, their paper did not present simulation results and left capacity-achieving ensembles as an interesting open problem. Metzner presented similar ideas earlier in [11] and [12], but the focus and analysis is quite different. Davey and MacKay also develop and analyze a symbol-level message-passing decoder over small finite fields in [71]. A number of approaches to the q-SC (for large q) based on interleaved Reed-Solomon codes are also possible [7] [13]. In [14], Shokrollahi and Wang discuss two ways of approaching capacity. The first uses a two-stage approach where the first stage uses a Tornado code and verification decoding. The second is, in fact, equivalent to one of the decoders we discuss in this paper.[1] When we discovered this, the authors were kind enough to send us an extended abstract [15] which contains more details. Still, the authors did not consider the theoretical performance with a maximum list size constraint, the actual performance of the decoder via simulation, or false verification (FV) due to cycles in the decoding graph. In this paper, we describe the algorithm in detail and

[1]The description of the second method in [14] is very brief and we believe its capacity-achieving nature deserves further attention.

consider those details.

Inspired by [10], we introduce list-message-passing (LMP) decoding with verification for LDPC codes on the q-SC. Instead of passing a single value between symbol and check nodes, we pass a list of candidates to improve the decoding threshold. This modification also increases the probability of FV. So, we analyze the causes of FV and discuss techniques to mitigate FV. It is worth noting that the LMP decoder we consider is somewhat different than the list extension suggested in [10]. Also, the algorithms in [10] are proposed in a node-based (NB) style but analyzed using message-based (MB) decoders. It is implicitly assumed that the two approaches are equivalent. In fact, this is not always true. In this paper, we consider the differences between NB and MB decoders and derive an asymptotic analysis for NB decoders.

The chapter is organized as follows. In Section B, we describe the LMP algorithm for bounded and unbounded list size and use density evolution (DE) [17] to analyze its performance. The difference between NB and MB decoders for the first (LM1) and second algorithm (LM2) in [10] is discussed and the NB decoder analysis is derived in Section C, respectively. The error floor of the LMP algorithms is considered in Section D. In Section E, we use differential evolution to optimize code ensembles. We describe the simulation of these codes and compare the results with the theoretical thresholds. We also compare our results with previously published results in this area [10] and [14]. In Section F, simulation results are shown. Applications of the LMP algorithm are discussed.

B. List-Message-Passing (LMP) Decoding Algorithm

1. Description of the Decoding Algorithm

The LMP decoder we discuss is designed mainly for the q-SC and is based on local decoding operations applied to lists of messages containing probable codeword symbols. The list messages passed in the graph have three types: *verified* (V), *unverified* (U) and *erasure* (E). Every V-message has a symbol value associated with it. Every U-message has a list of symbols associated with it. Following [10], we mark messages as verified when they are very likely to be correct. In particular, we will find that the probability of FV approaches zero as q goes to infinity.

The LMP decoder works by passing list-messages around the decoding graph. Instead of passing a single code symbol (e.g., Gallager A/B algorithm [4]) or a probability distribution over all possible code symbols (e.g., [71]), we pass a list of values that are more likely to be correct than the other messages. At a check node, the output list contains all symbols which could satisfy the check constraint for the given input lists. At the check node, the output message will be verified if and only if all the incoming messages are verified. At a node of degree d, the associativity and commutativity of the node-processing operation allow it to be decomposed into $(d-1)$ basic[2] operations (e.g., $a{+}b{+}c{+}d{=}(a{+}b){+}(c{+}d)$). In such a scheme, the computational complexity of each basic operation is proportional to s^2 at the check node and $s \ln s$ at the variable node[3], where s is the list size of the input list. The list size grows rapidly as the number of iterations increases. In order to make the algorithm practical, we have to truncate the list to keep the list size within some maximum value, denoted

[2]Here we use "basic" to emphasize that it maps two list-messages to a single list message.

[3]The basic operation at the variable node can be done by s binary searches of length s and the complexity of a binary search of length s is $O(\ln s)$

S_{max}. In our analysis, we also find that, after the number of iterations exceeds half the girth of the decoding graph, the probability of FV increases very rapidly. We analyze the reasons of FV and classify the FV's into two types. We find that the codes described in [10] and [14] both suffer from type-II FV. In Section D, we analyze these FV's and propose a scheme that reduces the probability of FV.

The message-passing decoding algorithm using list messages (or LMP) applies the following simple rules to calculate the output messages for a check node:

- If all the input messages are verified, then the output becomes verified with the value which makes all the incoming messages sum to zero.
- If any input message is an erasure, then the output message becomes an erasure.
- If there is no erasure on the input lists, then the output list contains all symbols which could satisfy the check constraint for the given input lists.
- If the output list size is larger than S_{max}, then the output message is an erasure.

It applies the following rules to calculate the output messages of a variable node:

- If all the input messages are erasures or there are multiple verified messages which disagree, then output message is the channel received value.
- If any of the input messages are verified (and there is no disagreement) or a symbol appears more than once, then the output message becomes verified with the same value as the verified input message or the symbol which appears more than once.
- If there is no verified message on the input lists and no symbol appears more than once, then the output list is the union of all input lists.

- If the output message has list size larger than S_{max}, then the output message is the received value from the channel.

2. Decoding with Unbounded List Size

To apply DE to the LMP decoder with unbounded list sizes, denoted LMP-∞ (i.e., $S_{max} = \infty$), we consider three quantities which evolve with the iteration number i. Let x_i be the probability that the correct symbol is not on the list passed from a variable node to a check node. Let y_i be the probability that the message passed from a variable node to a check node is not verified. Let z_i be the average list size passed from a variable node to a check node. The same variables are "marked" $(\tilde{x}_i, \tilde{y}_i, \tilde{z}_i)$ to represent the same values for messages passed from the check nodes to the variable nodes (i.e., the half-iteration value). We also assume all the messages are independent, that is, we assume that the bipartite graph has girth greater than twice the number of decoding iterations.

First, we consider the probability, x_i, that the correct symbol is not on the list. For any degree-d check node, the correct message symbol will only be on the edge output list if all of the other $d-1$ input lists contain their corresponding correct symbols. This implies that $\tilde{x}_i = 1 - \rho(1 - x_i)$. For any degree-$d$ variable node, the correct message symbol is not on the edge output list only if it is not on any of the other $d-1$ edge input lists. This implies that $x_{i+1} = p\lambda(\tilde{x}_i)$. This behavior is very similar to erasure decoding of LDPC codes on the BEC and gives the identical update equation

$$x_{i+1} = p\lambda\left(1 - \rho(1 - x_i)\right) \tag{2.1}$$

where p is the q-SC error probability. Note that throughout the DE analysis, we assume that q is sufficiently large. Next, we consider the probability, y_i, that the

message is not verified. For any degree-d check node, an edge output message is verified only if all of the other $d-1$ edge input messages are verified. For any degree-d variable node, an edge output message is verified if any symbol on the other $d-1$ edge input lists is verified or occurs twice which implies $\tilde{y}_i = 1-\rho(1-y_i)$. The event that the output message is not verified can be broken into the union of two disjoint events: (i) the correct symbol is not on any of the input lists, and (ii) the symbol from the channel is incorrect and the correct symbol is on exactly one of the input lists and not verified. For a degree-d variable node, this implies that

$$\Pr(\text{not verified}) = (\tilde{x}_i)^{d-1} + p(d-1)\left(\tilde{y}_i - \tilde{x}_i\right)(\tilde{x}_i)^{d-2}. \tag{2.2}$$

Summing over the d.d. gives the update equation

$$y_{i+1} = \lambda\left(1-\rho(1-x_i)\right) + p\left(\rho(1-x_i) - \rho(1-y_i)\right)\lambda'\left(1-\rho(1-x_i)\right). \tag{2.3}$$

It is important to note that (2.1) and (2.3) were published first in [14, Thm. 2] (by mapping $x_i = p_i$ and $y_i = p_i + q_i$), but were derived independently by us.

Finally, we consider the average list size z_i. For any degree-d check node, the output list size is equal[4] to the product of the sizes of the other $d-1$ input lists. Since the mean of the product of i.i.d. random variables is equal to the product of the means, this implies that $\tilde{z}_i = \rho(z_i)$. For any degree-d variable node, the output list size is equal to one[5] plus the sum of the sizes of the other $d-1$ input lists if the output is not verified and one otherwise. Again, the mean of the sum of $d-1$ i.i.d. random variables is simply $d-1$ times the mean of the distribution, so the average

[4]It is actually upper bounded because we ignore the possibility of collisions between incorrect entries, but the probability of this occurring is negligible as q goes to infinity.

[5]A single symbol is always received from the channel.

output list size is given by

$$1+\left(\left(\tilde{x}_i\right)^{d-1}+p(d-1)\left(\tilde{y}_i-\tilde{x}_i\right)\left(\tilde{x}_i\right)^{d-2}\right)(d-1)\tilde{z}_i.$$

This gives the update equation

$$z_{i+1}=1+\left[\tilde{x}_i\lambda'\left(\tilde{x}_i\right)+p\left(\tilde{y}_i-\tilde{x}_i\right)\left(\lambda'\left(\tilde{x}_i\right)+\tilde{x}_i\lambda''\left(\tilde{x}_i\right)\right)\right]\rho(z_i).$$

For the LMP decoding algorithm, the threshold of an ensemble $(\lambda(x),\rho(x))$ is defined to be

$$p^* \triangleq \sup\left\{p\in(0,1]\middle| p\lambda(1-\rho(1-x))<x\ \forall\ x\in(0,1]\right\}.$$

Next, we show that some codes can achieve channel capacity using this decoding algorithm.

Theorem 1. *Let p^* be the threshold of the d.d. pair $(\lambda(x),\rho(x))$ and assume that the channel error rate p is less than p^*. In this case, the probability y_i that a message is not verified in the i-th decoding iteration satisfies $\lim_{i\to\infty} y_i \to 0$. Moreover, for any $\epsilon>0$, there exists a $q<\infty$ such that LMP decoding of a long random (λ,ρ) LDPC code, on a q-SC with error probability p, results in a symbol error rate of less than ϵ.*

Proof. See Appendix B. □

Remark 1. *Note that the convergence condition, $p^*\lambda(1-\rho(1-x))<x$ for $x\in(0,1]$, is identical to the BEC case but that x has a different meaning. In the DE equation for the q-SC, x is the probability that the correct value is not on the list. In the DE equation for the BEC, x is the probability that the message is an erasure. This tells us any capacity-achieving ensemble for the BEC is capacity-achieving for the q-SC with LMP-∞ algorithm and large q. This also gives some intuition about the behavior of the q-SC for large q. For example, when q is large, an incorrectly received value behaves like an erasure [7].*

Corollary 1. *The code with d.d. pair* $\lambda(x) = x$ *and* $\rho(x) = (1-\epsilon)x + \epsilon x^2$ *has a threshold of* $1 - \frac{\epsilon}{1+\epsilon}$ *and a rate of* $r > \frac{\epsilon}{3(1+\epsilon)}$. *Therefore, it achieves a rate of* $\Theta(\delta)$ *for a channel error rate of* $p = 1 - \delta$.

Proof. Follows from $\left(1 - \frac{\epsilon}{1+\epsilon}\right) \lambda\left(1 - \rho(1-x)\right) < x$ for $x \in (0, 1]$ and Theorem 1. □

Remark 2. *We believe that Corollary 1 provides the first linear-time decodable construction of rate* $\Theta(\delta)$ *for a random-error model with error probability* $1-\delta$. *A discussion of linear-time encodable/decodable codes, for both random and adversarial errors, can be found in [18]. The complexity also depends on the required list size which may be extremely large (though independent of the block length). Unfortunately, we do not have explicit bounds on the required alphabet size or list size for this construction.*

In practice, we cannot implement a list decoder with unbounded list size. Therefore, we also evaluate the LMP decoder under a bounded list size assumption.

3. Decoding with Bounded List Size

First, we introduce some definitions and notation for the DE analysis with bounded list size decoding algorithm. Note that, in the bounded list-size LMP algorithm, each list may contain at most S_{max} symbols. For convenience, we classify the messages into four types:

(V) *Verified*: message is verified and has list size 1.

(E) *Erasure*: message is an erasure and has list size 0.

(L) *Correct on list*: message is not verified or erased and the correct symbol is on the list.

(N) *Correct not on list*: message is not verified or erased, and the correct symbol is not on the list.

For the first two message types, we only need to track the fraction, V_i and E_i, of message types in the i-th iteration. For the third and the fourth types of messages, we also need to track the list sizes. Therefore, we track the characteristic function of the list size for these messages, given by $L_i(x)$ and $N_i(x)$. The coefficient of x^j represents the probability that the message has list size j. Specifically, $L_i(x)$ is defined by

$$L_i(x) = \sum_{j=1}^{S_{max}} l_{i,j} x^j,$$

where $l_{i,j}$ is the probability that, in the i-th decoding iteration, the correct symbol is on the list and the message list has size j. The function $N_i(x)$ is defined similarly. This implies that $L_i(1)$ is the probability that the list contains the correct symbol and that it is not verified. For the same reason, $N_i(1)$ gives the probability that the list does not contain the correct symbol and that it is not verified. For the simplicity of expression, we denote the overall density as $P_i = [V_i, E_i, L_i(x), N_i(x)]$. The same variables are "marked" ($\tilde{V}, \tilde{E}, \tilde{L}, \tilde{N}$ and $\tilde{P}$) to represent the same values for messages passed from the check nodes to the variable nodes (i.e., the half-iteration value).

Using these definitions, we find that DE can be computed efficiently by using arithmetic of polynomials. For the convenience of analysis and implementation, we use a sequence of basic operations plus a separate truncation operator to represent a multiple-input multiple-output operation. We use $\boxplus$ to denote the check-node operator and $\otimes$ to denote the variable-node operator. Using this, the DE for the variable-node basic operation $P^{(3)} = \tilde{P}^{(1)} \otimes \tilde{P}^{(2)}$ is given by

$$V^{(3)} = \tilde{V}^{(1)} + \tilde{V}^{(2)} - \tilde{V}^{(1)}\tilde{V}^{(2)} + \tilde{L}^{(1)}(1)\tilde{L}^{(2)}(1) \tag{2.4}$$

$$E^{(3)} = \tilde{E}^{(1)}\tilde{E}^{(2)} \tag{2.5}$$

$$L^{(3)}(x) = \tilde{L}^{(1)}(x)\left(\tilde{E}^{(2)} + \tilde{N}^{(2)}(x)\right) + \tilde{L}^{(2)}(x)\left(\tilde{E}^{(1)} + \tilde{N}^{(1)}(x)\right) \tag{2.6}$$

$$N^{(3)}(x) = \tilde{N}^{(1)}(x)\tilde{E}^{(2)} + \tilde{N}^{(2)}(x)\tilde{E}^{(1)} + \tilde{N}^{(1)}(x)\tilde{N}^{(2)}(x). \tag{2.7}$$

Note that (2.4) to (2.7) do not yet consider the list size truncation and the channel value. For the basic check-node operation $\tilde{P}^{(3)} = P^{(1)} \boxplus P^{(2)}$, the DE is given by

$$\tilde{V}^{(3)} = V^{(1)}V^{(2)} \tag{2.8}$$

$$\tilde{E}^{(3)} = E^{(1)} + E^{(2)} - E^{(1)}E^{(2)} \tag{2.9}$$

$$\tilde{L}^{(3)}(z) = \left[V^{(1)}L^{(2)}(z) + V^{(2)}L^{(1)}(z) + L^{(1)}(x)L^{(2)}(y)\right]_{x^j y^k \to z^{jk}} \tag{2.10}$$

$$\tilde{N}^{(3)}(z) = \Big[N^{(1)}(x)N^{(2)}(y) + N^{(1)}(x)\left(V^{(2)}y + L^{(2)}(y)\right) + N^{(2)}(x)\left(V^{(1)}y + L^{(1)}(y)\right)\Big]_{x^j y^k \to z^{jk}} \tag{2.11}$$

where the subscript $x^j y^k \to z^{jk}$ means the replacement of variables. Finally, the truncation of lists to size S_{max} is handled by truncation operators which map densities to densities. We use $\mathcal{T}$ and $\mathcal{T}'$ to denote the truncation operation at the check and variable nodes. Specifically, we truncate terms with degree higher than S_{max} in the polynomials $L(x)$ and $N(x)$. At check nodes, the truncated probability mass is moved to E.

At variable nodes, lists longer than S_{max} are replaced by the channel value. Let $P_i' = \left(\tilde{P}_i^{\otimes k-1}\right)$ be the an intermediate density which is the result of applying the basic operation $k-1$ times on $\tilde{P}_i$. The correct symbol node message density after considering the channel value and truncation would be $\mathcal{T}'(P_i')$. To analyze this, we separate $L_i'(x)$ into two terms: $A_i'(x)$ with degree less than S_{max} and $x^{S_{max}}B_i'(x)$

with degree at least S_{max}. Likewise, we separate $N_i'(x)$ into $C_i'(x)$ and $x^{S_{max}} D_i'(x)$. The inclusion of the channel symbol and the truncation are combined into a single operation

$$P_i = \mathcal{T}' \left(\left[V_i', E_i', A_i'(x) + x^{S_{max}} B_i'(x), C_i'(x) + x^{S_{max}} D_i'(x) \right] \right)$$

defined by

$$V_i = V_i' + (1-p)\left(A_i'(1) + B_i'(1)\right) \tag{2.12}$$

$$E_i = 0 \tag{2.13}$$

$$L_i(x) = (1-p)x\left(E_i' + C_i'(x) + D_i'(1)\right) + pxA_i'(x) \tag{2.14}$$

$$N_i(x) = px\left(E_i' + B_i'(1) + C_i'(x) + D_i'(1)\right). \tag{2.15}$$

Note that in (2.12), the term $(1-p)\left(A_i'(1) + B_i'(1)\right)$ is due to the fact that messages are compared for possible verification before truncation.

The overall DE recursion is easily written in terms of the forward (symbol to check) density P_i and the backward (check to symbol) density $\tilde{P}_i$ by taking the irregularity into account. The initial density is $P_0 = [0, 0, (1-p)x, px]$, where p is the error probability of the q-SC channel, and the recursion is given by

$$\tilde{P}_i = \sum_{k=2}^{d_c} \rho_k \, \mathcal{T}\left(P_i^{\boxplus k-1}\right) \tag{2.16}$$

$$P_{i+1} = \sum_{k=2}^{d_v} \lambda_k \, \mathcal{T}'\left(\tilde{P}_i^{\otimes k-1}\right). \tag{2.17}$$

Note that the DE recursion is not one-dimensional. This makes it difficult to optimize the ensemble analytically. It remains an open problem to find the closed-form expression of the threshold in terms of the maximum list size, d.d. pairs, and the alphabet size q. In section E, we will fix the maximum variable and check degrees,

code rate, q and maximum list size and optimize the threshold over the d.d. pairs by using a numerical approach.

C. Analysis of Node-Based Algorithms

1. Differential Equation Analysis of LM1-NB

We refer to the first and second algorithms in [10] as LM1 and LM2, respectively. Each algorithm can be viewed either as message-based (MB) or node-based (NB). The first and second algorithms in [14] and [15] are referred to as SW1 and SW2. These algorithms are summarized in Table I. Note that, if no verification occurs, the variable node (VN) sends the ("channel value", U) and the check node (CN) sends the ("expected correct value",U) in all these algorithms. The algorithms SW1, SW2 and LMP are all MB algorithms, but can be modified to be NB algorithms.

a. Motivation

In [10], the algorithms are proposed in the node-based (NB) style [10, Section III-A and IV], but analyzed in the message-based (MB) style [10, Section III-B and IV]. It is easy to verify that the LM1-NB and LM1-MB have identical performance, but this is not ture for the NB and MB LM2 algorithms. In this section, we will show the differences between the NB decoder and MB decoder and derive a precise analysis for LM1-NB.

First, we show the equivalence between LM1-MB and LM1-NB.

Theorem 2. *Any verification that occurs in LM1-NB also occurs in LM1-MB and vice versa. Therefore, LM1-NB and LM1-MB are equivalent.*

Proof. See Appendix B. □

Table I. Brief Description of Message-Passing Algorithms for the q-SC

Alg.	**Description**
LMP-S_{max}	LMP as described in Section 1 with maximum list size S_{max}
LM1-MB	MP decoder that passes (value, U/V). [10, III.B] At VN's, output is V if any input is V or message matches channel value, otherwise pass channel value. At CN's, output is V if all inputs are V.
LM1-NB	Peeling decoder with VN state (value, U/V). [10, III.B] At CN's, if all neighbors sum to 0, then all neighbors get V. At CN's, if all neighbors but one are V, then last gets V.
LM2-MB	The same as LM1-MB with one additional rule. [10, IV.A]. At VN's, if two input messages match, then output V.
LM2-NB	The same as LM1-NB with one additional rule. [10, IV.A]. At VN's, if two neighbor values same, then VN gets V.
SW1	Identical to LM2-MB
SW2	Identical to LMP-∞. [14, Thm. 2]

Remark 3. *The theorem shows the equivalence between LM1-NB and LM1-MB. This also implies the stable error patterns or stopping sets of LM1-NB and LM1-MB are the same.*

In the NB decoder, the verification status is associated with the node. Once a node is verified, all the outgoing messages are verified. In the MB decoder, the status is associated with the edge/message and the outgoing messages may have different verification status. NB algorithms cannot, in general, be analyzed using DE because the independence assumption between messages does not hold. Therefore, we develop peeling-style decoders which are equivalent to LM1-NB and LM2-NB and

use differential equations to analyze them.

Following [22], we analyze the peeling-style decoder using differential equations to track the average number of edges (grouped into types) in the graph as decoding progresses. From the results from [23] and [22], we know that the actual number of edges (of any type), in any particular decoding realization is tightly concentrated around the average over the lifetime of the random process. In a peeling-style decoder for $GF(q)$, a variable node and its edges are removed after verification. The check node keeps track of the new parity constraint (i.e., the value to which the attached variables must sum) by subtracting values associated with the removed edges.

b. Analysis of Peeling-Style Decoding

First, we introduce some notation and definitions for the analysis. A variable node (VN) whose channel value is correctly received is called a correct variable node (CVN), otherwise it is called an incorrect variable node (IVN). A check node (CN) with i edges connected to the CVN's and j edges connected to the IVN's will be said to have C-degree i and I-degree j, or type (i, j).

We also define the following quantities:

- t: decoding time or the fraction of VNs removed from graph
- $L_i(t)$: the number of edges connected to CVN's with degree i at time t.
- $R_j(t)$: the number of edges connected to IVN's with degree j at time t.
- $N_{i,j}(t)$: the number of edges connected to CN's with C-degree i and I-degree j.
- $E_l(t)$: the remaining number of edges connected to CVN's at time t.
- $E_r(t)$: the remaining number of edges connected to IVN's at time t.

- $a(t)$: the average degree of CVN's,

$$a(t) = \sum_{i \geq 0} L_i(t) i / E_l(t)$$

- $b(t)$: the average degree of IVN's,

$$b(t) = \sum_{i \geq 0} R_i(t) i / E_r(t)$$

- E: number of edges in the original graph,

$$E = E_l(0) + E_r(0)$$

Counting edges in three ways gives the following equations:

$$\sum_{i \geq 0} L_i(t) + \sum_{i \geq 0} R_i(t) = E_l(t) + E_r(t) = \sum_{i \geq 0} \sum_{j \geq 0} N_{i,j}(t).$$

These r.v.'s represent a particular realization of the decoder. The differential equations are defined for the normalized (i.e., divided by E) expected values of these variables. We use lower-case notation (e.g., $l_i(t)$, $r_i(t)$, $n_{i,j}(t)$, etc.) for these deterministic trajectories. For a finite system, the decoder removes exactly one variable node in one time step of Δt.

The description of peeling-style decoder is as follows. The Tanner graph of an LDPC codes can be represented as Fig. 1. The peeling-style decoder removes one CVN or IVN in each time step by the following rules:

CER: If any CN has its edges all connected to CVN's, pick one of the CVN's and remove it and all its edges.

IER1: If any IVN has at least one edge connected to a CN of type $(0, 1)$, then the value of the IVN is given by the attached CN and we remove the IVN and all

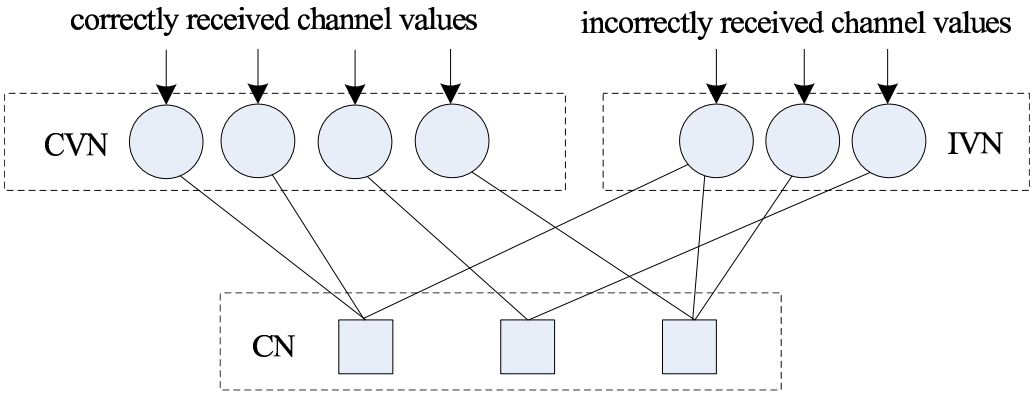

Fig. 1. Tanner graph for differential equation analysis.

its outgoing edges.

If both CER and IER1 can be applied, then one is chosen randomly as described below.

Since both rules remove exactly one VN, the decoding process either finishes in exactly N steps or stops early and cannot continue. The first case occurs only when either the IER1 or CER condition is satisfied in every time step. When the decoder stops early, the pattern of CVNs and IVNs remaining is known as a stopping set. We also note that the rules above, though described differently, are equivalent to the first node-based algorithm (LM1-NB) introduced in [10].

Recall that in the node-based algorithm for LM1 we have two verification rules. The first rule is that if all messages but one are verified at a CN, then all messages are verified. We call this type-I incorrect-edge-removal (IER1) and this is only possible when $n_{0,1}(t) > 0$. The second rule is: if all messages sum to zero at a CN, then all messages are verified. We call this as correct-edge-removal (CER) in the peeling-style decoder and this requires $n_{i,0} > 0$ for some $i \geq 1$. The peeling-style decoder performs one operation in time step. The operation is random and can be either CER or IER1. When both operations are possible, we choose randomly between these two rules by

picking CER with probability $c_1(t)$ and IER1 with probability $c_2(t)$, where

$$c_1(t) = \frac{\sum_{i\geq 1} n_{i,0}(t)}{\sum_{i\geq 1} n_{i,0}(t) + n_{0,1}(t)}$$
$$c_2(t) = \frac{n_{0,1}(t)}{\sum_{i\geq 1} n_{i,0}(t) + n_{0,1}(t)}.$$

Therefore, the differential equations can be written as

$$\frac{\mathrm{d}l_i(t)}{\mathrm{d}t} = c_1(t)\frac{\mathrm{d}l_i^{(1)}(t)}{\mathrm{d}t} + c_2(t)\frac{\mathrm{d}l_i^{(2)}(t)}{\mathrm{d}t}$$
$$\frac{\mathrm{d}r_i(t)}{\mathrm{d}t} = c_1(t)\frac{\mathrm{d}r_i^{(1)}(t)}{\mathrm{d}t} + c_2(t)\frac{\mathrm{d}r_i^{(2)}(t)}{\mathrm{d}t}$$
$$\frac{\mathrm{d}n_{i,j}(t)}{\mathrm{d}t} = c_1(t)\frac{\mathrm{d}n_{i,j}^{(1)}(t)}{\mathrm{d}t} + c_2(t)\frac{\mathrm{d}n_{i,j}^{(2)}(t)}{\mathrm{d}t},$$

where $^{(1)}$ and $^{(2)}$ denote, respectively, the effects of CER and IER1.

c. CER Analysis

If the CER operation is picked, then we choose randomly an edge attached to a CN of type $(i, 0)$ with $i \geq 1$. This VN endpoint of this edge is distributed uniformly across the CVN edge sockets. Therefore, it will be attached to a CVN of degree k with probability $\frac{l_k(t)}{e_l(t)}$. Therefore, one has the following differential equations for l_k and r_k

$$\frac{\mathrm{d}l_k^{(1)}(t)}{\mathrm{d}t} = \frac{l_k(t)}{e_l(t)}(-k), \text{ for } k \geq 1$$

and

$$\frac{\mathrm{d}r_k^{(1)}(t)}{\mathrm{d}t} = 0.$$

For the effect on check edges, we can think of removing a CVN with degree k as first randomly picking an edge of type $(k, 0)$ connected to that CVN and then removing all the other $k-1$ edges (called reflected edges) attached to the same CVN. The $k-1$ reflected edges are uniformly distributed over the $E_l(t)$ correct sockets of

the CN's. Averaging over all graphs, the $k-1$ reflected edges hit $\frac{n_{i,j}(t)i(k-1)}{(i+j)e_l(t)}$ CN's of type (i,j). Averaging over the degree k shows that the reflected edges hit $\frac{n_{i,j}(t)i(a(t)-1)}{(i+j)e_l(t)}$ CN's of type (i,j).

If a CN of type (i,j) is hit by a reflected edge, then we lose $i+j$ edges of type (i,j) and gain $i-1+j$ edges of type $(i-1,j)$. Hence, one has the following differential equation for $j>0$ and $i+j\leq d_c$

$$\frac{\mathrm{d}n_{i,j}^{(1)}(t)}{\mathrm{d}t}=\left(p_{i+1,j}^{(1)}(t)-p_{i,j}^{(1)}(t)\right)(i+j)$$

where

$$p_{i,j}^{(1)}(t)=\frac{n_{i,j}(t)i(a(t)-1)}{(i+j)e_l(t)}.$$

One should keep in mind that $n_{i,j}(t)=0$ for $i+j>d_c$.

For $n_{i,j}^{(1)}(t)$ with $j=0$, the effect from above must be combined with effect of the type-$(i,0)$ initial edge that was chosen. So the differential equation becomes

$$\frac{\mathrm{d}n_{i,0}^{(1)}(t)}{\mathrm{d}t}=\left(p_{i+1,0}^{(1)}(t)-p_{i,0}^{(1)}(t)\right)i+\left(q_{i+1}^{(1)}(t)-q_{i}^{(1)}(t)\right)i$$

where

$$q_i^{(1)}(t)=\frac{n_{i,0}(t)}{\sum_{m\geq 1}n_{m,0}(t)}.$$

Note that $p_{d_c+1,0}^{(1)}(t)\triangleq 0$ and $q_{d_c+1}^{(1)}(t)\triangleq 0$

d. IER1 Analysis

If the IER1 operation is picked, then we choose a random CN of type $(0,1)$ and follow its only edge to the set of IVNs. The edge is attached uniformly to this set, so the differential equations for IER1 can be written as

$$\frac{\mathrm{d}l_k^{(2)}(t)}{\mathrm{d}t}=0,$$

$$\frac{\mathrm{d}r_k^{(2)}(t)}{\mathrm{d}t} = \frac{r_k(t)}{e_r(t)}(-k), \text{ for } k \geq 1$$

and

$$\frac{\mathrm{d}n_{i,j}^{(2)}(t)}{\mathrm{d}t} = \left(p_{i,j+1}^{(2)}(t) - p_{i,j}^{(2)}(t)\right)(i+j), \text{ for } (i,j) \neq (0,1)$$

where

$$p_{i,j}^{(2)}(t) = \frac{n_{i,j}(t)j(b(t)-1)}{(i+j)e_r(t)}.$$

For $n_{i,j}(t)$ with $(i,j) = (0,1)$, the differential equation must also account for the initial edge and becomes

$$\frac{\mathrm{d}n_{0,1}^{(2)}(t)}{\mathrm{d}t} = \left(p_{0,2}^{(2)}(t) - p_{0,1}^{(2)}(t)\right) - 1.$$

Notice that even for (3,6) codes, there are 30 differential equations[6] to solve. So we solve the differential equations numerically and the threshold for (3,6) code with LM1 is $p^* = 0.169$. This coincides with the result from density evolution analysis for LM1-MB in [10] and hints at the equivalence between LM1-NB and LM1-MB. In the proof of Theorem 2 we make this equivalence precise by showing that the stopping sets of LM1-NB and LM1-MB are the same.

2. Differential Equation Analysis of LM2-NB

Similar to the analysis of LM1-NB algorithm, we analyze LM2-NB algorithm by analyzing a peeling-style decoder which is equivalent to the LM2-NB decoding algorithm. The peeling-style decoder removes one CVN or IVN in each time unit according to the following rules:

CER: If any CN has all its edges connected to CVN's, pick one of the CVN's and

[6]There are 28 for $n_{i,j}$ $(i,j \in [0,\cdots,6]$ such that $i+j \leq 6)$, 1 for $r_k(t)$, and 1 for $l_k(t)$.

remove it.

IER1: If any IVN has any messages from CN's with type $n_{0,1}$, then the IVN and all its outgoing edges can be removed and we track the correct value by subtracting the value from the check node.

IER2: If any IVN is attached to more than one CN with I-degree 1, then it will be verified and all its outgoing edges can be removed.

For the reason of simplicity, we first introduce some definitions and short-hand notations.

- Correct Edges: edges which are connected to CVN's
- Incorrect Edges: edges which are connected to IVN's
- CER edges: the edges which are connected to check nodes with type $n_{i,0}$ for $i \geq 1$
- IER1 edges: the edges which are connected to check nodes with type $n_{0,1}$
- IER2 edges: the edges which connect IVN's and the check nodes with type $n_{i,1}$ for $i \geq 1$
- NIE edges: normal incorrect edges, which are incorrect edges but neither IER1 edges nor IER2 edges
- CER nodes: CVN's which have at least one CER edge
- IER1 nodes: IVN's which have at least one IER1 edge
- IER2 nodes: IVN's which have at least two IER2 edge
- NIE nodes: IVN's which contain at most 1 IER2 edge and no IER1 edges.

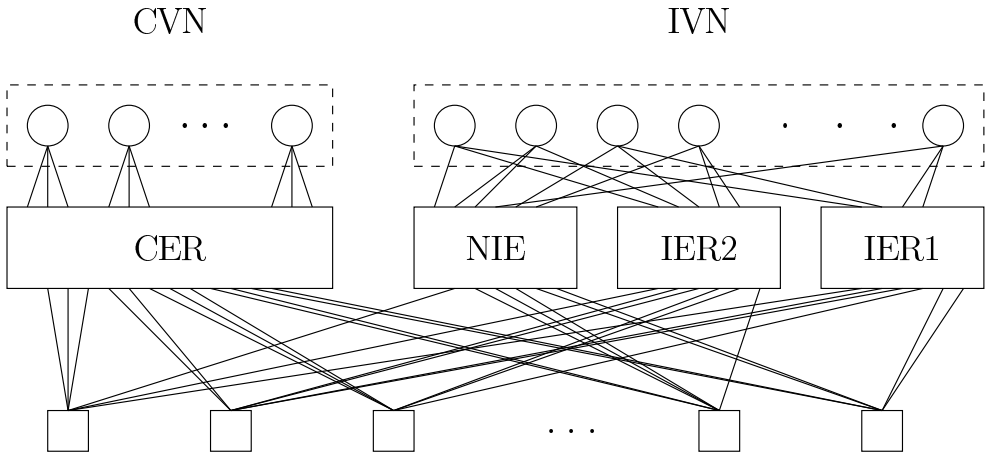

Fig. 2. Graph structure of an LDPC code with LM2-NB decoding algorithm

Note that an IVN can be both an IER1 node and an IER2 node at the same time.

The analysis of LM2-NB is much more complicated than LM1-NB because the IER2 operation makes the distribution of IER2 edges dependent on each other. In each IER2 operation, one IVN with more than 2 IER2 edges can be removed, therefore the rest of the IER2 edges are more likely to land on different IVN's. The idea to analyze LM2-NB decoder is to separate the incorrect edges into types such that the edges in each type imply an uniform permutation. In detail, we model the structure of an LDPC codes with LM2-NB decoder as shown in Fig. 2. There is one type of correct edge and three types of incorrect edges. There is a random permutation for each type of edges. We say a random permutation is uniform if each input socket is mapped to all the output sockets with equal probability. As we will see soon, the permutation of each type is uniform which allows us to calculate the probabilities of certain events.

The peeling-style decoder randomly chooses one VN from the set of CER IER1 and IER2 nodes and removes this node and all its edges at each step. The idea of the analysis is to first calculate the probability of choosing a VN with a certain type,

i.e., CER, IER1 or IER2, and the node degree. We then analyze how removing this VN affects the system parameters.

In the analysis, we will track the evolution of the following system parameters.

- $l_k(t)$: the fraction of edges connected to CVN's with degree k, $0 < k \leq d_v$ at time t. [7]

- $r_{i,j,k}(t)$: the fraction of edges connected to IVN's with i NIE edges, j IER2 edges and k IER1 edges at time t, $i, j, k \in \{0, 1, \ldots, d_v\}$ and $0 < i + j + k \leq d_v$. [8]

- $n_{i,j}(t)$: the fraction of edges connected to check nodes with i correct edges and j incorrect edges at time t, $i, j \in \{0, 1, \ldots, d_c\}$ and $0 < i + j \leq d_c$. [9]

We note that, when we say "fraction", we mean the number of a certain type of edges/nodes normalized by the number of edges/nodes in the *original graph*.

The following quantities can be calculated by $l_k(t)$, $r_{i,j,k}(t)$ and $n_{i,j}(t)$.

- $e_l(t) \triangleq \sum_{k=1}^{d_v} l_k(t)$: the fraction of correct edges

- $e_r(t) \triangleq \sum_{i=0}^{d_v} \sum_{j=0}^{d_v-i} \sum_{k=0}^{d_v-i-j} r_{i,j,k}(t)$: the fraction of correct edges

- $\eta_0(t) \triangleq \sum_{j=2}^{d_c} \sum_{i=0}^{d_c-j} \frac{j n_{i,j}(t)}{i+j} = \sum_{i=1}^{d_v} \sum_{j=0}^{d_v-i} \sum_{k=0}^{d_v-i-j} \frac{i r_{i,j,k}(t)}{i+j+k}$: the fraction of NIE edges

- $\eta_1(t) \triangleq n_{0,1}(t) = \sum_{k=1}^{d_v} \sum_{j=0}^{d_v-k} \sum_{i=0}^{d_v-i-j} \frac{k r_{i,j,k}(t)}{(i+j+k)}$: the fraction of IER1 edges

- $\eta_2(t) \triangleq \sum_{i=1}^{d_c} \frac{n_{i,1}(t)}{(i+1)} = \sum_{j=1}^{d_v} \sum_{i=0}^{d_v-j} \sum_{k=0}^{d_v-i-j} \frac{j r_{i,j,k}(t)}{(i+j+k)}$: the fraction of IER2 edges

---

[7] We don't track $l_0(t)$ and simply set $l_0(t) = 0$.

[8] We don't track $r_{0,0,0}(t)$ and simply set $r_{0,0,0}(t) = 0$.

[9] We don't track $n_{0,0}(t)$ and simply set $n_{0,0}(t) = 0$.

- $s_0(t) \triangleq \sum_{j=0}^{1} \sum_{i=1}^{d_v-j} \frac{r_{i,j,0}}{i+j}$: the fraction of NIE nodes
- $s_1(t) \triangleq \sum_{k=1}^{d_v} \frac{n_{k,0}(t)}{k}$: the fraction of CER nodes
- $s_2(t) \triangleq \sum_{k=1}^{d_v} \sum_{i=0}^{d_v-k} \sum_{j=0}^{d_v-i-k} \frac{r_{i,j,k}}{i+j+k}$: the fraction of IER1 nodes
- $s_3(t) \triangleq \sum_{j=2}^{d_v} \sum_{i=0}^{d_v-j} \sum_{k=0}^{d_v-i-j} \frac{r_{i,j,k}}{i+j+k}$: the fraction of IER2 nodes

As in the LM1-NB analysis, we use superscript $^{(1)}$ to denote the contribution by the CER operations. We use $^{(2)}$ to denote the contribution by the IER1 operations and $^{(3)}$ to denote the contribution by the IER2 operations. Since we assume that the decoder randomly chooses a VN from the set of CER, IER1 and IER2 nodes and removes all its edges in each time unit, the differential equations of the system parameters can be written as the weighted sum of the contributions by CER, IER1 and IER2 operations. The weights are chosen as

$$\begin{aligned} c_1(t) &= \frac{s_1(t)}{(s_1(t)+s_2(t)+s_3(t))} \\ c_2(t) &= \frac{s_2(t)}{(s_1(t)+s_2(t)+s_3(t))} \\ c_3(t) &= \frac{s_3(t)}{(s_1(t)+s_2(t)+s_3(t))}. \end{aligned}$$

Next, we will show how CER, IER1 and IER2 operations affect the system parameters.

Given the d.d. pair (λ, ρ) and the channel error probability p, we initialize the state as follows. Since a fraction $(1-p)\lambda_k$ of the edges are connected to CVN's of degree k, we initialize $l_k(t)$ with

$$l_k(0) = (1-p)\lambda_k,$$

for $k = 1, 2, \ldots d_v$. Noticing that each CN socket is connected to a correct edge with probability $(1-p)$ and incorrect edge with probability p, we initialize $n_{i,j}(t)$ with

$$n_{i,j}(0) = \rho_{i+j} \binom{i+j}{i} (1-p)^i p^j,$$

for $i + j \in \{1, 2, \ldots, d_c\}$. Since the probability an IVN socket is connected to an NIE is $\frac{\sum_{j'=2}^{d_c} \sum_{i'=0}^{d_c - j'} \frac{j' n_{i',j'}(0)}{i'+j'}}{p}$ and the probability an IVN socket is connected to an IER2 edge is $\frac{n_{0,1}(0)}{p}$, we initialize $r_{i,j,k}(t)$ with

$$r_{i,j,k}(0) = p\lambda_{i+j+k} \binom{i+j+k}{i,j,k} \left(\frac{\sum_{j'=2}^{d_c} \sum_{i'=0}^{d_c - j'} \frac{j' n_{i',j'}(0)}{i'+j'}}{p} \right)^i \left(\frac{\sum_{i'=1}^{d_c} \frac{n_{i',1}(0)}{(i'+1)}}{p} \right)^j \left(\frac{n_{0,1}(0)}{p} \right)^k,$$

for $i + j + k \in \{1, 2, \ldots, d_v\}$.

a. CER Analysis

The analysis for $\frac{\mathrm{d}l_k^{(1)}(t)}{\mathrm{d}t}$ is the same as LM1-NB analysis. In the CER operation, the decoder randomly selects a CER edge. With probability $\frac{l_k(t)}{e_l(t)}$, a CVN with degree k is chosen, this decreases the number of edges of type l_k by k. This gives

$$\frac{\mathrm{d}l_k^{(1)}(t)}{\mathrm{d}t} = \frac{-k l_k(t)}{e_l(t)}, \ k \geq 1.$$

For $j \geq 1$ and $i + j \leq d_c$

$$\frac{\mathrm{d}n_{i,j}^{(1)}(t)}{\mathrm{d}t} = \left(p_{i+1,j}^{(1)}(t) - p_{i,j}^{(1)}(t) \right) (i+j)$$

where $a(t) = \frac{\sum_{k=1}^{d_v} k l_k(t)}{e_l(t)}$ is the average degree of the CVN's which are hit by the initially chosen CER edge and $p_{i,j}^{(1)} = \frac{n_{i,j}(t) i (a(t)-1)}{(i+j) e_l(t)}$ is the average number of CN's with type $n_{i,j}$ hit by the $a(t) - 1$ reflecting edges.

For $j = 0$ and $i \geq 1$, we also have to consider the initially chosen CER edge.

This gives

$$\frac{\mathrm{d}n_{i,0}^{(1)}(t)}{\mathrm{d}t} = \left(p_{i+1,0}^{(1)}(t) - p_{i,0}^{(1)}(t)\right) i + \left(q_{i+1}^{(1)}(t) - q_{i}^{(1)}(t)\right) i$$

where $q_i^{(1)}(t) = \frac{n_{i,0}(t)}{\sum_{m\geq 1} n_{m,0}(t)}$ is the probability that the initially chosen CER edge is of type $n_{i,0}$.

When one of the reflecting edge of the removed CER node hits a CN of type $n_{1,1}$, an IER2 edge becomes an IER1 edge. This is the only way the CER operation can affect $r_{i,j,k}$. On average, each CER operation generates $a(t) - 1$ reflecting edges. For each reflecting edge, the probability that it hits a CN of type $n_{1,1}$ is $\frac{n_{1,1}(t)}{2e_l(t)}$. Once a reflecting edge hits a CN with type $n_{1,1}$, one IER2 edge is changed to IER1 edge, but not removed. By considering this, when $j \neq d_v$ and $k \neq 0$, we have

$$\begin{aligned}\frac{\mathrm{d}r_{i,j,k}^{(1)}(t)}{\mathrm{d}t} &= (a(t)-1)\frac{n_{1,1}(t)}{2e_l(t)} \\ &\quad \left(\frac{jr_{i,j,k}(t)}{(i+j+k)\eta_2(t)}(-(i+j+k)) - \frac{(j+1)r_{i,j+1,k-1}(t)}{(i+j+k)\eta_2(t)}(-(i+j+k))\right) \\ &= (a(t)-1)\frac{n_{1,1}(t)}{2e_l(t)}\left(\frac{-jr_{i,j,k}(t)}{\eta_2(t)} + \frac{(j+1)r_{i,j+1,k-1}(t)}{\eta_2(t)}\right).\end{aligned}$$

If $k = 0$ or $k \neq d_v$, then the IVN's with type $r_{i,j,k}$ can only lose edges and

$$\frac{\mathrm{d}r_{i,j,k}^{(1)}(t)}{\mathrm{d}t} = (a(t)-1)\frac{n_{1,1}(t)}{2e_l(t)}\left(\frac{-jr_{i,j,k}(t)}{\eta_2(t)}\right).$$

b. IER2 Analysis

Since IER2 operation does not affect $l_k(t)$, we have

$$\frac{\mathrm{d}l_k^{(3)}(t)}{\mathrm{d}t} = 0.$$

To analyze how IER2 operation changes $n_{i,j}(t)$ and $r_{i,j,k}(t)$, we first calculate the probability that a randomly chosen IER2 node is of type $r_{i,j,k}$ as follows

$$\Pr\left(\text{type } r_{i,j,k}|\text{IER2 node}\right) = \frac{\frac{r_{i,j,k}(t)}{i+j+k}}{s_2(t)}$$

if $j \geq 2$ and $i+j+k \neq 0$. Otherwise, $\Pr\left(\text{type } r_{i,j,k}|\text{IER2 node}\right) = 0$.

Let's denote $\frac{\mathrm{d}n^{(3)}_{i',j'}(t)}{\mathrm{d}t}$ caused by removing one NIE edge as $u_{i',j'}(t)$, $\frac{\mathrm{d}n^{(3)}_{i',j'}(t)}{\mathrm{d}t}$ caused by removing one IER2 edge as $v_{i',j'}(t)$ and $\frac{\mathrm{d}n^{(3)}_{i',j'}(t)}{\mathrm{d}t}$ caused by removing one IER1 edge as $w_{i',j'}(t)$. Then, we can write $\frac{\mathrm{d}n^{(3)}_{i',j'}(t)}{\mathrm{d}t}$ as

$$\frac{\mathrm{d}n^{(3)}_{i',j'}(t)}{\mathrm{d}t} = \sum_{i=0}^{d_v}\sum_{j=0}^{d_v}\sum_{k=0}^{d_v}\Pr\left(\text{type } r_{i,j,k}|\text{IER2 node}\right)\left(iu_{i',j'}(t) + jv_{i',j'}(t) + kw_{i',j'}(t)\right).$$

First, we consider $u_{i',j'}(t)$. If an NIE edge is chosen from the IVN side, it hits a CN of type $n_{i,j}$ with probability $\frac{jn_{i,j}(t)}{\eta_0(i+j)}$ if $j \geq 2$ and with probability 0 otherwise. When $j \geq 2$ and $j \leq d_v - 1$, we have

$$\begin{aligned} u_{i',j'}(t) &= \frac{j'n_{i',j'}(t)}{(i'+j')\eta_0(t)}(-(i'+j')) + \frac{(j'+1)n_{i',j'+1}(i'+j')}{(i'+j'+1)\eta_0(t)} \\ &= \frac{-j'n_{i',j'}(t)}{\eta_0(t)} + \frac{(j'+1)n_{i',j'+1}(i'+j')}{(i'+j'+1)\eta_0(t)} \end{aligned}$$

and, when $j = d_v$, we have

$$u_{i',j'}(t) = \frac{-j'n_{i'j'}(t)}{\eta_0(t)}.$$

Since an NIE edge cannot be connected to a CN with type $n_{i',1}$, we must treat $j' = 1$ separately. Notice that $n_{i',1}$ can still gain edges from $n_{i',2}$, we have

$$u_{i',1}(t) = \frac{2n_{i',2}(t)(i'+1)}{(i'+2)\eta_0(t)}.$$

When $j' = 0$, CN's with type $n_{i',0}$ do not have any NIE edges. So we have

$$u_{i',0}(t) = 0.$$

Now we consider $v_{i',j'}(t)$. Since edges of type $n_{i',j'}$ with $j \geq 2$ cannot be IER2 edges, $n_{i'j'}$ with $j \geq 2$ is not affected by removing IER2 edges. The IER2 edge removal reduces the number of edges of type $n_{i',1}$, $i \geq 1$. So we have

$$v_{i',1} = -\frac{n_{i',1}}{\eta_2(t)}.$$

When $j' = 0$ and $i' \geq 1$, we have

$$v_{i',0} = \frac{n_{i',1}}{\eta_2(t)}.$$

The CN's with only type $n_{0,1}$ are affected when we remove an IER1 edge on the IVN side. So we have

$$w_{0,1} = 1$$

and $w_{i',j'} = 0$ when $(i', j') \neq (0, 1)$.

Next, we derive the differential equation for $r_{i,j,k}$ caused by removing an IER2 node. If the decoder removes an IER2 node with type $r_{i',j',k'}$, we need to study how this affects $r_{i,j,k}(t)$. There are two effects caused by removing an IER2 node of type $r_{i',j',k'}$. When we remove an IER2 node of type $r_{i',j',k'}$, we remove i' NIE edges, j' IER2 edges and k' IER1 edges. For each removed edge, if we look at the CN side, it may cause the types of some other edges on the same CN to change and therefore affect $r_{i,j,k}(t)$. We also call the edges other than the one coming from the removed IER2 node as "CN reflecting edges". Let $u'_{i,j,k}(t)$ be the contribution to $\frac{\mathrm{d}r_{i,j,k}(t)}{\mathrm{d}t}$ caused by the CN reflecting edges of an NIE edge on the CN. Let $v'_{i,j,k}(t)$ be the contribution to $\frac{\mathrm{d}r_{i,j,k}(t)}{\mathrm{d}t}$ caused by the CN reflecting edges of an IER2 edge on the CN. Let $w'_{i,j,k}(t)$ be the contribution to $\frac{\mathrm{d}r_{i,j,k}(t)}{\mathrm{d}t}$ caused by the CN reflecting edges of an IER1 edge on

the CN. Then we can write $\frac{\mathrm{d}r^{(3)}_{i,j,k}(t)}{\mathrm{d}t}$ as

$$\frac{\mathrm{d}r^{(3)}_{i,j,k}(t)}{\mathrm{d}t} = -\Pr\left(\text{type } r_{i,j,k}|\text{IER2 node}\right)(i'+j'+k')+ \sum_{i'=0}^{d_v}\sum_{j'=0}^{d_v}\sum_{k'=0}^{d_v}\Pr\left(\text{type } r_{i',j',k'}|\text{IER2 node}\right)\left(i'u'_{i,j,k}(t)+j'v'_{i,j,k}(t)+k'w'_{i,j,k}(t)\right).$$

There are two ways that the CN reflecting edges of an NIE edge can affect $r_{i,j,k}(t)$. The first one is when the CN is of type $n_{i,2}$, $1 \le i \le d_c-2$. Removing an NIE can change the type of the other incorrect edge from NIE to IER2. The second way is when the CN is of type $n_{0,2}$. Removing an NIE can change the type of the other incorrect edge from NIE to IER1. Notice that the probability that an NIE hits a CN of type $n_{i,2}$, $1 \le i \le d_c-2$ is $\frac{\sum_{i=1}^{d_c-2}\frac{2n_{i,2}(t)}{i+2}}{\eta_0(t)}$, the probability that an NIE hits a CN of type $n_{0,2}$ is $\frac{n_{0,2}(t)}{\eta_0(t)}$ and the probability that an NIE edge is connected to an IVN of type $r_{i,j,k}$ is $\frac{ir_{i,j,k}(t)}{(i+j+k)\eta_0(t)}$. Therefore, we can write

$$u'_{i,j,k}(t) = \frac{\sum_{i=1}^{d_c-2}\frac{2n_{i,2}(t)}{i+2}}{\eta_0(t)}\left(-\frac{ir_{i,j,k}(t)}{\eta_0(t)}+\frac{(i+1)r_{i+1,j-1,k}(t)}{\eta_0(t)}\right) + \frac{n_{0,2}(t)}{\eta_0(t)}\left(-\frac{ir_{i,j,k}(t)}{\eta_0(t)}+\frac{(i+1)r_{i+1,j,k-1}(t)}{\eta_0(t)}\right)$$

if $i \neq d_v$, $j \neq 0$ and $k \neq 0$. When $i \neq d_v$, $j \neq 0$ and $k = 0$, we have

$$u'_{i,j,k}(t) = \frac{\sum_{i=1}^{d_c-2}\frac{2n_{i,2}(t)}{i+2}}{\eta_0(t)}\left(-\frac{ir_{i,j,k}(t)}{\eta_0(t)}+\frac{(i+1)r_{i+1,j-1,k}(t)}{\eta_0(t)}\right) + \frac{n_{0,2}(t)}{\eta_0(t)}\left(-\frac{ir_{i,j,k}(t)}{\eta_0(t)}\right).$$

When $i \neq d_v$, $j = 0$ and $k \neq 0$, we have

$$u'_{i,j,k}(t) = \frac{\sum_{i=1}^{d_c-2}\frac{2n_{i,2}(t)}{i+2}}{\eta_0(t)}\left(-\frac{ir_{i,j,k}(t)}{\eta_0(t)}\right) + \frac{n_{0,2}(t)}{\eta_0(t)}\left(-\frac{ir_{i,j,k}(t)}{\eta_0(t)}+\frac{(i+1)r_{i+1,j,k-1}(t)}{\eta_0(t)}\right).$$

When $i \neq d_v$, we have

$$u'_{i,j,k}(t) = \frac{\sum_{i=1}^{d_c-2} \frac{2n_{i,2}(t)}{i+2}}{\eta_0(t)} \left(-\frac{ir_{i,j,k}(t)}{\eta_0(t)} \right) + \frac{n_{0,2}(t)}{\eta_0(t)} \left(-\frac{ir_{i,j,k}(t)}{\eta_0(t)} \right).$$

Since there are no CN reflecting edges of type IER1 and IER2, $v'_{i,j,k}(t) = 0$ and $w'_{i,j,k}(t) = 0$.

Like IER2 operations, the IER1 operation does not affect $l_k(t)$. So, we have

$$\frac{\mathrm{d}l_k^{(2)}(t)}{\mathrm{d}t} = 0.$$

To analyze how IER1 changes $n_{i,j}(t)$ and $r_{i,j,k}(t)$, we first calculate the probability that a randomly chosen IER1 node is of type $r_{i,j,k}$ as follows

$$\Pr\left(\text{type } r_{i,j,k} | \text{IER1 node}\right) = \frac{\frac{r_{i,j,k}(t)}{i+j+k}}{s_1(t)}$$

when $k \geq 1$ and $\Pr\left(\text{type } r_{i,j,k} | \text{IER1 node}\right) = 0$ when $k = 0$.

For the same reason,

$$\frac{\mathrm{d}n_{i',j'}^{(2)}(t)}{\mathrm{d}t} = \sum_{i=0}^{d_v} \sum_{j=0}^{d_v} \sum_{k=0}^{d_v} \Pr\left(\text{type } r_{i,j,k} | \text{IER1 node}\right) \left(iu_{i',j'}(t) + jv_{i',j'}(t) + kw_{i',j'}(t) \right)$$

and

$$\frac{\mathrm{d}r_{i,j,k}^{(2)}(t)}{\mathrm{d}t} = -\Pr\left(\text{type } r_{i',j',k'} | \text{IER1 node}\right)(i'+j'+k') + \sum_{i'=0}^{d_v} \sum_{j'=0}^{d_v} \sum_{k'=0}^{d_v} \Pr\left(\text{type } r_{i',j',k'} | \text{IER1 node}\right) \left(i'u'_{i,j,k}(t) + j'v'_{i,j,k}(t) + k'w'_{i,j,k}(t) \right).$$

We note that we provide a Matlab program for the LM2-NB analysis which is available online at http://www.ece.tamu.edu/~hpfister/software/lm2nb_threshold.m

3. Proof of Correctness of Differential Equation Analysis for LM2-NB

Consider the peeling decoder for the BEC introduced in [10]. Throughout the decoding process, one reveals and then removes edges one at a time from a hidden random graph. The analysis of this decoder is simplified by the fact that, given the current residual degree distribution, the unrevealed portion of the graph remains uniform for every decoding trajectory. In fact, one can build a finite-length decoding simulation never constructs the actual decoding graph. Instead, it tracks only the residual degree distribution of the graph and implicitly chooses a random decoding graph one edge at a time.

For asymptotically long codes,[10] used this approach leads to derive an analysis based on differential equations. This analysis is actually quite general and can also be applied to other peeling-style decoder in which the unrevealed graph is not uniform. One may observe this from its proof of correctness, which depends on two important observations. First, the distribution of all decoding paths is concentrated very tightly around its average as the system size increases. Second, the expected change in the decoder state can be written as a Lipschitz function of the current decoder state. If one augments the decoding state to include enough information so that the expected change can be computed from the augmented state (even for non-uniform residual graphs), then the theorem still applies.

The differential equation gives the average (over all the random bipartite graph, or the code ensemble) evolution of the system parameters as the block length n goes to infinity. While the numerical simulation of long codes gives the evolution of the system parameters of a particular code (a particular bipartite graph) as n goes to infinity. To prove that the differential equation analysis precisely predicts the evolution of the system parameters of a particular code, we need to show the concentration of the

evolution of the system parameters of a particular code around the ensemble average as n goes to infinity.

In the LM2-NB algorithm, one edge is removed at a time. The main difference for LM2-NB algorithm is that we have more edge types and we track some details of the edge types on both the check nodes and the variable nodes. This causes a significant problem in the analysis because updating the exact effect of edge removal requires revealing some edges before they will be removed. For example, the CER operation can cause an IER2 edge to become an IER1 edge, but revealing the adjacent symbol node (or type) renders the analysis intractable.

To prove the correctness of the differential equation analysis, we will put our analysis of the LM2-NB algorithm in the framework of [10, 23, 24] by indicating exactly the discrete-time random process associated with the decoding analysis.

We first introduce the definitions of the random process. In this subsection, we use t to represent the discrete time. We follow the same notation used in [10]. Let the life-span of the random process be m. Let Ω denote a probability space and S be a measurable space of observations. A discrete-time random process over Ω with observations S is a sequence $Q \triangleq (Q_0, Q_1, \dots)$ of random variables where Q_t contains the information revealed at t-th step. We denote the history of the process up to time t as $H_t \triangleq (Q_0, Q_1, \dots, Q_t)$. Let $S^+ := \cup_{i \geq 1} S^i$ denote the set of all histories and $\mathcal{Y}$ be the set of all decoder states. One typically uses a state space that tracks the number of edges of a certain type (e.g., the degree of the attached nodes).

We define the random process as follows. Let's define the total number of edges connected to IVN's with type $r_{i,j,k}$ at time t as $R_{i,j,k}(t)$ and the total number of edges connected to check nodes with type $n_{i,j}$ as $N_{i,j}(t)$. Let $\bar{R}_{i,j,k}(t) \triangleq E[R_{i,j,k}(t)|H_t]$. Let

$$R(t) \triangleq \{R_{i,j,k}(t), 0 \leq i \leq d_v, 0 \leq j \leq d_v, 0 \leq k \leq d_v, 0 \leq i+j+k \leq d_v\},$$

$$\bar{R}(t) \triangleq \{\bar{R}_{i,j,k}(t), 0 \leq i \leq d_v, 0 \leq j \leq d_v, 0 \leq k \leq d_v, 0 \leq i+j+k \leq d_v\}$$

and

$$N(t) \triangleq \{N_{i,j}(t), 0 \leq i \leq d_c, 0 \leq j \leq d_c, 0 \leq i+j \leq d_c\}.$$

Let the r.v. $Y_t \triangleq \{N(t), \bar{R}(t)\}$ be the decoder state at time t. We will show that Y_t/m concentrates around the solutions of our differential equation. In particular, The $\bar{R}_{i,j,k}(t)$ concentrates around $r_{i,j,k}(t/m)$ and $\bar{N}_{i,j}(t)$ concentrates around $n_{i,j}(t/m)$.

The history of the random process is defined as follows. In the beginning of the decoding, we label the variable/check nodes by their degrees. When the decoder removes an edge, the revealed information Q_t contains the degree of the variable node and type of the check node to which the removed edge is connected to. We note that sometimes the edge-removal operation changes the type of the un-removed edge on that check node. In this case, Q_t also contains the information about the type of the check node to which this CN-reflecting edge is connected to. But Q_t does not contain any information about the IVN that this CN-reflecting edge is connected to. By defining the history as this, Y_t is a deterministic function of H_t, and it satisfies the bounded change condition, i.e., condition (i) in Theorem 5.1 in [24]. The condition (iii) is also verified. We note that $E[Y_{t+1} - Y_t | H_t] = f\left(t/m, \frac{N(t)}{m}, \frac{R(t)}{m}\right)$. By the following conjecture and the Lipschitz condition of $f\left(t/m, \frac{N(t)}{m}, \frac{R(t)}{m}\right)$, we have $E[Y_{t+1} - Y_t | H_t] = f\left(t/m, \frac{N(t)}{m}, \frac{\bar{R}(t)}{m}\right) + o(1)$. This shows condition (ii) of Theorem 5.1 in [24] is also satisfied.

Conjecture 1. $\lim_{m\to\infty} \Pr\left(\sup_{0\leq t\leq m} \left|\bar{R}_{i,j,k}(t) - R_{i,j,k}(t)\right| \geq m^{5/6}\right) = 0$ *holds for all* $\{i,j,k : 0 \leq i \leq d_v, 0 \leq j \leq d_v, 0 \leq k \leq d_v, 0 \leq i+j+k \leq d_v\}$.

By Theorem 5.1 in [24], one can show the concentration of Y_t, or $N(t)$ and $\bar{R}(t)$. Since $R(t)$ and $\bar{R}(t)$ concentrate around the same function, we complete the proof.

D. Error Floor Analysis of LMP Algorithms

During the simulation of the optimized ensembles of Table II, we observed an error floor that warranted more attention. While one might expect a floor due to finite q effects, the simulation uses q large enough so that no FV's were observed in the error floor regime. Instead, the error floor is due to the event that some symbols remain unverified when the decoding terminates. This motivates us to analyze the error floor of LMP algorithms. We need to point out that, when q is relatively small, error floors are caused by a mixture of several reasons such as type-I FV, type-II FV (which we will discuss later) and the event that some symbols remain unverified when the decoding terminates. These reasons are coupled together and affect each other.

But this is not the case of our interest and there are three reasons for this. The first reason is that this is not the setting in our simulation, (e.g., the error floors observed in the simulation are not caused by FV). The second reason is that, in practice, one would like to let the assumption "the verified symbols are correct with high probability" hold to make the verification-based algorithms to work well and to make the analysis correct. This is can be done by picking large enough q as we did in the simulation. Note that, if FV has significant impact on the algorithms, then both the density evolution analysis and the differential equation analysis break down and the thresholds are not correct anymore. The last reason is for the simplicity of

Table II. Optimized Ensemble for LMP Algorithms (rate 1/2)

Alg.	$\boldsymbol{\lambda}(\mathbf{x})$	$\boldsymbol{\rho}(\mathbf{x})$	$\mathbf{p}^*$
LMP-1	$.1200x+.3500x^2+.0400x^4+.4900x^{14}$	x^8	.2591
LMP-1	$.1650x+.3145x^2+.2111x^{14}+.0265x^{24}+.2674x^{49}$	$.0030x^2+.9970x^{10}$	.2593
LMP-8	$.32x+.24x^2+.26x^8+.19x^{14}$	$.02x^4+.82x^6+.16x^8$	.288
LMP-32	$.40x+.20x^3+.13x^5+.04x^8+.23x^{14}$	$.04x^4+.96x^6$	.303
LMP-∞	$.34x+.16x^2+.21x^4+.29x^{14}$	x^7	.480
LM2-MB	$.2x+.3x^3+.05x^5+.45x^{11}$	x^8	.289

analysis. One can analyze the error floor caused by different reasons separately since they are not coupled.

We note that, even though the error floor is not caused by FV, we still provide an analysis of FV for sake of the completeness. The analysis actually helps us understand why the dominant error events caused by FV can be avoided by increasing q. The analysis is derived by considering each effect separately.

1. The Union Bound for ML Decoding

First, we derive the union bound on the probability of error with ML decoding for the q-SC. To match our simulations with the union bounds, we expurgate (i.e., ignore) all codeword weights that have an expected multiplicity less than 1.

First, we summarize a few results from [20, p. 497] that characterize the low-weight codewords of LDPC codes with degree-2 variable nodes. When the block length n is large, all of these low-weight codewords are caused, with high probability, by short cycles of degree-2 nodes. For binary codes, the number of codewords with weight k is a random variable which converges to a Poisson distribution with mean $\frac{\left(\lambda_2 \rho^{'}(1)\right)^k}{2k}$. When the channel quality is high (i.e., high SNR, low error/erasure rate), the probability of ML decoding error is mainly caused by low-weight codewords.

For non-binary $GF(q)$ codes, a codeword is supported on a cycle of degree-2 nodes only if the product of the edge weights is 1. This occurs with probability $1/(q-1)$ if we choose the i.i.d. uniform random edge weights for the code. Hence, the number of $GF(q)$ codewords of weight k is a random variable, denoted B_k, which converges to a Poisson distribution with mean $b_k = \frac{\left(\lambda_2 \rho^{'}(1)\right)^k}{2k(q-1)}$. After expurgating weights that have an expected multiplicity less than 1, $k_1 = \arg\min_{k \geq 1} b_k^{(n)} \geq 1$ becomes the minimum codeword weight. An upper bound on the pairwise error probability (PEP) of the q-SC with error probability p is given by the following lemma.

Lemma 1. *Let y be the received symbol sequence assuming the all-zero codeword was transmitted. Let u be any codeword with exactly k non-zero symbols. Then, the probability that the ML decoder chooses u over the all-zero codeword is upper bounded by*

$$p_{2,k} \leq \left(p\frac{q-2}{q-1} + \sqrt{\frac{4p(1-p)}{q-1}} \right)^k .$$

Proof. See Appendix C. □

Remark 4. *Notice that b_k is exponential in k and the PEP is also exponential in k. The union bound for the frame error rate, due to low-weight codewords, can be written as*

$$P_B \leq \sum_{k=k_1}^{\infty} b_k p_{2,k}.$$

It is easy to see $k_1 = \Omega(\log q)$ and the sum is dominated by the first term $b_{k_1} p_{2,k_1}$ which has the smallest exponent. When q is large, the PEP upper bound is on the order of $O\left(p^k\right)$. Therefore. the order of the union bound on frame error rate with ML decoding is

$$P_B = O\left(\frac{\left(\lambda_2 \rho^{'}(1) p\right)^{\log q}}{q \log q} \right)$$

and the expected number of symbols in error is

$$O\left(\frac{\left(\lambda_2 \rho^{'}(1) p\right)^{\log q}}{q} \right),$$

if $p\lambda_2\rho'(1) < 1$.

2. Error Analysis for LMP Algorithms

The error of LMP algorithm comes from two types of decoding failure. The first type of decoding failure is due to unverified symbols. The second one is caused by the FV. To understand the performance of LMP algorithms, we analyze these types of failure

separately. Note that when we analyze each error type, we neglect the interaction for the simplicity of analysis.

The FV's can be classified into two types. The first type is, as [10] mentions, when the error magnitudes in a single check sum to zero; we call this type-I FV. For single-element lists, it occurs with probability roughly $1/q$ (i.e., the chance that two uniform random symbols are equal). For multiple lists with multiple entries, we analyze the FV probability under the assumption that no list contains the correct symbol. In this case, each list is uniform on the $q-1$ incorrect symbols. For m lists of size $s_1, \ldots, s_m$, the type-I FV probability is given by $1 - \binom{q-1}{s_1, s_2, \cdots, s_m} \Big/ \prod_{i=1}^{m} \binom{q-1}{s_i}$. In general, the Birthday paradox applies and the FV probability is roughly $s^2 \binom{m}{2}/q$ for large q and equal size lists.

The second type of FV is that messages become more and more correlated as the number of iterations grows, so that an incorrect message may go through different paths and return to the same node. We denote this kind of FV as a type-II FV.

Note that these are two different types of FV and one does not affect another. We cannot avoid type-II FV by increasing q without randomizing the edge weights and we cannot avoid type-I FV by constraining the number of decoding iterations to be within half of the girth (or increasing the girth). Fig. 3 shows an example of type-II FV. In Fig. 3, there is an 8-cycle in the graph and we assume the variable node on the right has an incorrect incoming message "a". Assume that the all-zero codeword is transmitted, all the incoming messages at each variable node are not verified, the list size is less than S_{max}, and each incoming message at each check node contains the correct symbol. In this case, the incorrect symbol will travel along the cycle and cause FV's at all variable nodes along the cycle. If the characteristic of the field is 2, there are a total of $c/2$ FV's occurring along the cycle, where c is the length of the cycle. This type of FV can be reduced significantly by choosing each non-zero entry in the

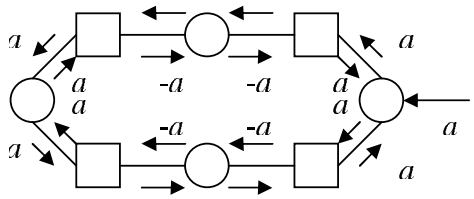

Fig. 3. An example of type-II FV's.

parity-check matrix randomly from the non-zero elements of Galois field. In this case, a cycle causes a type-II FV only if the product of the edge-weights along that cycle is 1. Therefore, we suggest choosing the non-zero entries of the parity-check matrix randomly to mitigate type-II FV. Recall that the idea to use non-binary elements in the parity-check matrix appears in the early works on the LDPC codes over $GF(q)$ [71].

3. An Upper Bound on the Probability of Type-II FV on Cycles

In this subsection, we analyze the probability of error caused by type-II FV. Note that type-II FV occurs only when the depth-$2k$ directed neighborhood of an edge (or a node) has cycles. But type-I FV occurs at every edge (or node). The order of the probability that type-I FV occurs is approximately $O(1/q)$ [10]. The probability of type-II FV is hard to analyze because it depends on q, S_{max} and k in a complicated way. But an upper bound of the probability of the type-II FV is derived in this section.

Since the probability of type-II FV is dominated by short cycles of degree-2 nodes, we only analyze type-II FV along cycles of degree-2 nodes. As we will soon see, the probability of type-II FV is exponential in the length of the cycle. So, the error caused by type-II FV on cycles is dominated by short cycles. We also assume S_{max} to be large enough such that an incorrectly received value can pass around a

cycle without being truncated. This assumption makes our analysis an upper bound. Another condition required for an incorrectly received value to participate in a type-II FV is that the product of the edge weights along the cycle is 1. If we assume that almost all edges not on the cycle are verified, then once any edge on the cycle is verified, all edges will be verified in the next k iterations. So we also assume that nodes along a cycle are either all verified or all unverified.

We note that there are three possible patterns of verification on a cycle, depending on the received values. The first case is that all the nodes are received incorrectly. As mentioned above, the incorrect value passes around the cycle without being truncated, comes back to the node again and falsely verifies the outgoing messages of the node. So all messages will be falsely verified (if they are all received incorrectly) after k iterations. Note that this happens with probability $\frac{1}{q-1}p^k$. The second case is that all messages are verified correctly, say, no FV. Note that this does not require all the nodes to have correctly received values. For example, if any pair of adjacent nodes are received correctly, it is easy to see all messages will be correctly verified. The last case is, there is at least 1 incorrectly received node in any pair of adjacent nodes and there is at least 1 node with correctly received value on the cycle. In this case, all messages will be verified after k iterations, i.e., messages from correct nodes are verified correctly and those from incorrect nodes are falsely verified. Then the verified messages will propagate and half of the messages will be verified correctly and the other half will be falsely verified. Note that this happens with probability $\frac{1}{q-1}2\left(p^{k/2}-p^k\right) \approx \frac{2p^{k/2}}{q-1}$ and this approximation gives an upper bound even if we combine the previous $\frac{1}{q-1}p^k$ term.

Recall that the number of cycles with length k converges to a Poisson with mean $\frac{(\lambda_2\rho'(1))^k}{2k}$. Using the union bound, we can upper bound on the ensemble average

probability of any type-II FV event with

$$\Pr(\text{any type-II FV}) \leq \sum_{k=k_1}^{\infty} \frac{\left(\lambda_2 \rho^{'}(1)\right)^k}{2k(q-1)} 2p^{\frac{k}{2}} = \sum_{k=k_1}^{\infty} \frac{\left(\lambda_2 \rho^{'}(1)\sqrt{p}\right)^k}{k(q-1)}.$$

The ensemble average number of nodes involved in type-II FV events is given by

$$\mathrm{E}\left[\text{symbols in type-II FV}\right] \leq \sum_{k=k_1}^{\infty} \frac{\left(\lambda_2 \rho^{'}(1)\right)^k}{2k(q-1)} 2kp^{\frac{k}{2}} = \sum_{k=k_1}^{\infty} \frac{\left(\lambda_2 \rho^{'}(1)\sqrt{p}\right)^k}{(q-1)}.$$

The upper bound on the frame error rate of type-II FV is on the order of $O\left(\frac{\left(\lambda_2 \rho^{'}(1)\sqrt{p}\right)^{\log q}}{(q-1)\log q}\right)$ and the upper bound on the ensemble average number of nodes in type-II FV symbol is on the order of $O\left(\frac{\left(\lambda_2 \rho^{'}(1)\sqrt{p}\right)}{(q-1)}\right)$. Notice that both bounds are decreasing functions of q.

4. An Upper Bound on the Probability of Unverification on Cycles

In the simulation of the optimized ensembles from Table II, we observe significant error floors and all the error events are caused by some unverified symbols when the decoding terminates. In this subsection, We derive the union bound for the probability of decoder failure caused by the symbols on short cycles which never become verified. We call this event as *unverification* and we denote it by UV. As described above, to match the settings of the simulation and simplify the analysis, we assume q is large enough to have arbitrarily small probability of both type-I and type-II FV. In this case, the error is dominated by the unverified messages because the following analysis shows that the union bound on the probability of unverification is independent of q.

In contrast to type-II FV, unverification event does not require cycles, i.e., unverification occurs even on subgraphs without cycles. But in the low error-rate regime, the dominant unverification events occur on short cycles of degree-2 nodes. Therefore, we only analyze the probability of unverification caused by short cycles of degree-2

nodes.

Consider a degree-2 cycle of length k and assume that no FV occurs in the neighborhood of this cycle. Assuming the maximum list size is S_{max}, the condition for UV is that there is at most one correctly received value along $S_{max}+1$ adjacent variable nodes. Note that we don't consider type-II FV since type-II FV occurs with probability $\frac{1}{q-1}$ and we can choose q to be arbitrarily large. On the other hand, unverification does not require the product of the edge weights on a cycle to be 1, so we cannot mitigate it by increasing q. So the union bound on the probability of unverification on a cycle with length k is

$$P_U \leq \sum_{k \geq k_2} \frac{\left(\lambda_2 \rho^{'}(1)\right)^k}{2k} \phi(S_{max}, p, k)$$

where $k_2 = \arg\min_{k \geq 1} \frac{\left(\lambda_2 \rho^{'}(1)\right)^k}{2k} \geq 1$ and $\phi(S_{max}, p, k)$ is the UV probability which is given by the following lemma.

Lemma 2. *Let the cycle have length k, the maximum list size be s, and the channel error probability be p. Then, the probability of an unverification event on a degree-2 cycle of length-k is $\phi(s,p,k) = Tr\left(B^k(p)\right)$ where $B(p)$ is the $(s+1)$ by $(s+1)$ matrix*

$$B(p) = \begin{bmatrix} p & 1-p & 0 & 0 & \cdots & 0 \\ 0 & 0 & p & 0 & \cdots & 0 \\ 0 & 0 & 0 & p & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & p \\ p & 0 & 0 & 0 & \cdots & 0 \end{bmatrix}. \tag{2.18}$$

Proof. See Appendix D. □

Finally, the union bound on the average number of symbols involved in unveri-

fication events is

$$\mathrm{E}\left[\text{unverified symbols}\right] \leq \sum_{k \geq k_2} \frac{\left(\lambda_2 \rho^{'}(1)\right)^k}{2} \phi(S_{max}, p, k). \tag{2.19}$$

Note that if we have to choose some small q and we need to consider type-II FV, then the union bound P_U can be easily rewritten as

$$P_U \leq \sum_{k \geq k_2} \frac{\left(\lambda_2 \rho^{'}(1)\right)^k (q-2)}{2k(q-1)} \phi(S_{max}, p, k)$$

since all symbols will always be verified if the product of the weights on the edges equals to 1 if s is larger than half of the length of the cycle,[10] the necessary conditions for unverification are the UV condition mentioned above and the product of the weights on the edges does not equal to 1. The union bound on the average number of symbols involved in unverification events is

$$\mathrm{E}\left[\text{unverified symbols}\right] \leq \sum_{k \geq k_2} \frac{\left(\lambda_2 \rho^{'}(1)\right)^k (q-2)}{2(q-1)} \phi(S_{max}, p, k). \tag{2.20}$$

Let's look at (2.19) and (2.20), we can see that the average number of unverified symbols scales exponentially with k. The ensemble with larger $\lambda_2 \rho^{'}(1)$ will have more short degree-2 cycles and more average unverified symbols. The average number of unverified symbols depends on the maximum list size S_{max} in a complicated way. Intuitively, if S_{max} is larger, then the constraint that "there is at most one correct symbol along S_{max} adjacent variable nodes" becomes stronger since we assume the probability of seeing a correct symbol is higher than that of seeing a incorrect symbol. Therefore, unverification is less likely to happen and the average number of unverified symbols will decrease as S_{max} increases. Note that (2.19) does not depend on q and (2.20) depends on q weakly.

[10] When s is not large enough, this analysis provides an upper bound.

One might expect that the stability condition of the LMP-S_{max} decoding algorithms can be used to analyze the error floor. Actually, one can show that the stability condition for LMP-S_{max} decoding of irregular LDPC codes is identical to that of the BEC, which is $p\lambda_2\rho'(1) < 1$. This is not much help for predicting the error floor though, because for codes with degree-2 nodes, the error floor is determined mainly by short cycles of degree-2 nodes instead. A finite number of degree-2 cycles is predicted instead by the condition $\lambda_2\rho'(1) < 1$.

E. Comparison and Optimization

In this section, we compare the proposed algorithm with maximum list size S_{max} (LMP-S_{max}) with other message-passing decoding algorithms for the q-SC. We note that the LM2-MB algorithm is identical to SW1 for any code ensemble because the decoding rules are the same. LM2-MB, SW1 and LMP-1 are identical for (3,6) regular LDPC codes because the list size is always 1 and erasures never happen in LMP-1 for (3,6) regular LDPC codes. The LMP-∞ algorithm is identical to SW2.

There are two important differences between the LMP algorithm and previous algorithms: (i) erasures and (ii) FV recovery. The LMP algorithm passes erasures because, with a limited list size, it is better to pass an erasure than to keep unlikely symbols on the list. The LMP algorithm also detects FV events and passes an erasure if they cause disagreement between verified symbols later in decoding, and can sometimes recover from a FV event. LM1-NB and LM2-NB fix the status of a variable node once it is verified and pass the verified value in all following iterations.

The results in [10] and [15] also do not consider the effects of type-II FV. These FV events degrade the performance in practical systems with moderate block lengths, and therefore we use random entries in the parity-check matrix to mitigate these

effects.

Using the DE analysis of the LMP-S_{max} algorithm, we can improve the threshold by optimizing the degree distribution pair (λ, ρ). Since the DE recursion is not one-dimensional, we use differential evolution to optimize the code ensembles [21]. In Table II, we show the results of optimizing rate-$\frac{1}{2}$ ensembles for LMP with a maximum list size of 1, 8, 32, and ∞. Thresholds for LM1 and LM2-NB/MB with rate 1/2 are also shown. In all but one case, the maximum variable-node degree is 15 and the maximum check-node degree is 9. The second table entry allowed for larger degrees (in order to improve performance) but very little gain was observed. We can also see that there is a gain of between 0.05 and 0.07 over the thresholds of (3,6) regular ensemble with the same decoder.

F. Simulation Results

In this part, we show the simulation results for (3,6) regular LDPC codes using various decoding algorithms as well as the simulation results for the optimized ensembles shown in Table II with LMP algorithms in Fig. 4. In the simulation of optimized ensembles, we try different maximum list sizes and different finite fields. We use notation "LMPS_{max},q,ensemble" to denote the simulation result of LMP algorithm with maximum list size S_{max}, finite field $GF(q)$ and the simulated ensemble. We choose the block length to be 100000. The parity-check matrices are chosen randomly without 4-cycles. Each non-zero entry in the parity-check matrix is chosen uniformly from GF$(q)\backslash 0$. This allows us to keep the FV probability low. The maximum number of decoding iterations is fixed to be 200 and more than 1000 blocks are run for each point. These results are compared with the theoretical thresholds. Table III shows the theoretical thresholds of $(3, 6)$ regular codes on the q-SC for different algorithms

Table III. Threshold vs. Algorithm for (3,6) Regular LDPC Codes

LMP-1	LMP-8	LMP-32	LMP-∞	LM1	LM2-MB	LM2-NB
.210	.217	.232	.429	.169	.210	.259

and Table II shows the thresholds for the optimized ensembles. The numerical results match the theoretical thresholds very well.

In the results of (3,6) regular codes simulation, we cannot see any error floor because there is almost no FV in the simulation. The LM2-NB performs much better than other algorithms with list size 1 for (3,6) regular ensemble. In the optimized ensembles, there are a large number of degree-2 variable nodes which cause the significant error floor. By evaluating (2.19), the predicted error floor caused by unverification is 1.6×10^{-5} for the optimized $S_{max} = 1$ ensemble, 8.3×10^{-7} for the optimized $S_{max} = 8$ ensemble, and 1.5×10^{-6} for the optimized $S_{max} = 32$ ensemble. From the results, we see the analysis of unverification events matches the numerical results very well.

In next chapter, we will see some applications of LDPC codes over large alphabet sets together with VB decoding algorithms in compressed sensing systems.

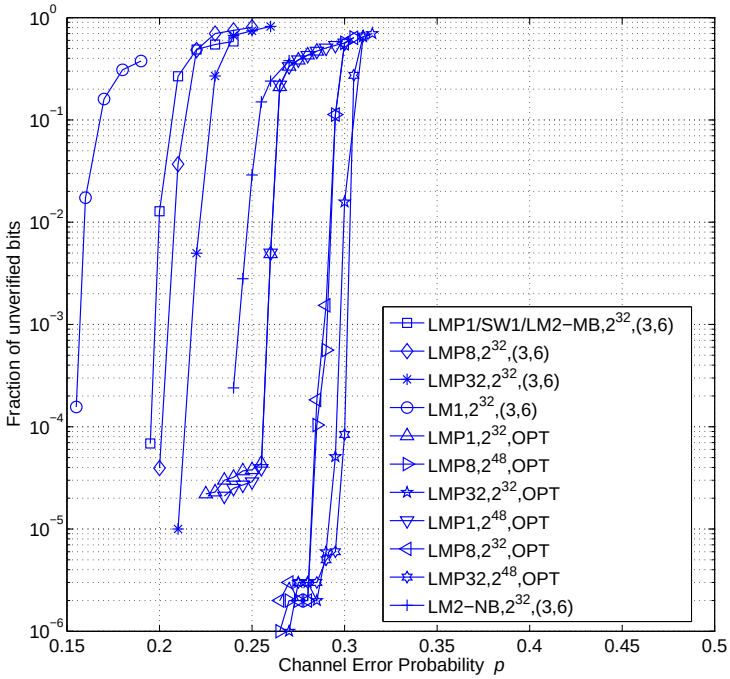

Fig. 4. Simulation results for (3,6) regular codes with block length 100000.

CHAPTER III

VERIFICATION DECODING OF HIGH-RATE LDPC CODES WITH APPLICATIONS IN COMPRESSED SENSING

A. Introduction

Compressed sensing (CS) is a relatively new area of signal processing that has recently received a large amount of attention. The main idea is that many real-world signals (e.g., those sparse in some transform domain) can be reconstructed from a relatively small number of linear dot-product measurements. Its roots lie in the areas of statistics and signal processing [27, 28, 29], but it is also very much related to previous work in computer science [30] and applied mathematics [31, 32, 33]. CS is also very closely related to error correcting codes, and can be seen as source coding using linear codes over real numbers [34, 35, 36, 37, 38].

In this chapter, we analyze the performance of low-density parity-check (LDPC) codes with verification decoding [68] as applied to CS. The resulting decoding algorithm is almost identical to that of Sudocodes [34], but a more suitable code ensemble is chosen and a more precise analysis is presented. Since most of the interesting applications of CS require very sparse (or compressible) signals, the natural mapping to coding implies high-rate codes. One new characteristic of this analysis, which is also interesting from a coding perspective, is that the performance estimates hold uniformly as the code rate approaches one. This allows us to explore the sparsity (or rate) regime that makes sense for compressed sensing.

An important implication of this work is that our randomized reconstruction system allows linear-time reconstruction of strictly-sparse signals with a constant oversampling ratio. In contrast, all previous reconstruction methods with moderate

reconstruction complexity have an oversampling ratio which grows logarithmically with the signal dimension. We note that the material of this chapter is mainly from [25].

1. Background on LDPC Codes

LDPC codes are linear codes introduced by Gallager in 1962 [4] and re-discovered by MacKay in 1995 [39]. Binary LDPC codes are now known to be capacity approaching on various channels when the block length tends to infinity. They can be represented by a Tanner graph, where the i-th variable node is connected to the j-th check node if the entry on the i-th column and j-th row of its parity-check matrix is non-zero.

LDPC codes can be decoded by an iterative *message-passing* (MP) algorithm which passes messages between the variable nodes and check nodes iteratively. If the messages passed along the edges are probabilities, then the algorithm is also called *belief propagation* (BP) decoding. The performance of the MP algorithm can be evaluated using density evolution (DE) [40] and stopping set analysis [41] [42]. Each method provides a decoding threshold for the code ensemble.

2. Structure of the Chapter

Section B provides background information on coding and CS. Section C summarizes the main results. In Section D, proofs and details are given for the main results based on DE. While in Section E, proofs and details are provided for the main results based on stopping-set analysis. Section F discusses a simple information-theoretic bound on the number of measurements required for reconstruction. Section D presents simulation results comparing the algorithms discussed in this chapter with a range of other algorithms.

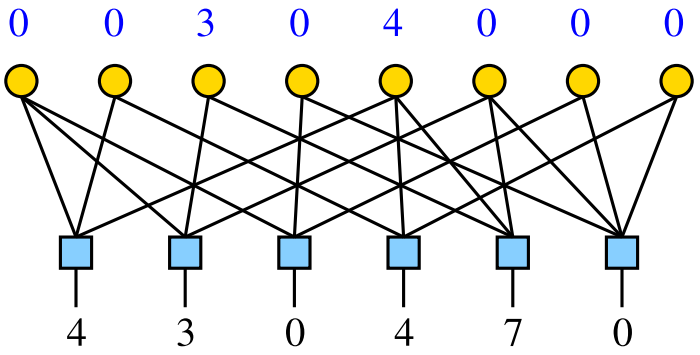

Fig. 5. Structure of the encoder.

B. Background on Coding and CS

1. Encoding and Decoding

The sparse graph representation of LDPC codes allows encoding and decoding algorithms to be implemented with linear complexity in the code length n. Since LDPC codes are usually defined over the finite field $GF(q)$ instead of the real numbers, we need to modify the encoding/decoding algorithm to deal with signals over real numbers. Each entry in the parity-check matrix is either 0 or a real number drawn from a continuous distribution. The parity-check matrix $\Phi \in \mathbb{R}^{m \times n}$ will be full-rank with high probability (w.h.p.) and is used as the measurement matrix in the CS system (e.g., the signal vector $x \in \mathbb{R}^n$ is observed as $y = \Phi x$).

The process of generating the observation symbols can also be seen in a bipartite Tanner graph representation. Fig. 5 shows the encoder structure. Each non-zero entry in Φ is the edge-weight of its corresponding edge in this graph. Therefore, the observation process associated with a degree d check node is as follows:

1. Encoding: The observation symbol is the weighted sum (using the edge weights) of the d neighboring signal components.

In this work, we only consider strictly sparse signals and we use two decoders based on verification which were proposed and analyzed in [68]. The second algorithm was also proposed independently for CS in [34]. The decoding process uses the following rules:

1. If a measurement is zero, then all the neighboring variable nodes are verified as zero.

2. If a check node is of degree one, verify the variable node with the value of the measurement.

3. [Enhanced verification] If two check nodes overlap in a single variable node and have the same measurement value, then verify that variable node to the value of the measurement.

4. Remove all verified variable nodes and the edges attached to them by subtracting out the verified values from the measurements.

5. Repeat steps 1-4 until decoding succeeds or makes no further progress.

Note the first algorithm follows steps 1, 2, 4 and 5. The second algorithm follows steps from 1 to 5. These two algorithms correspond to the first and second algorithms in [68] and are referred to as LM1 and node-based LM2 (LM2-NB) in this chapter[1]. Note that LDPC codes with regular check degree and Poisson symbol degree with LM2-NB decoding is identical to the Sudocodes introduced in [34]. In [69], the LM2-NB

[1]In [68], the second algorithm (which we refer to as LM2) was described in a node-based (NB) fashion (as above), but analyzed using a message-based (MB) density-evolution. There is an implicit assumption that the two algorithms perform the same. In fact, they perform differently and the LM2-NB algorithm is superior as observed in [43][69].

algorithm which is an enhanced version of message-based LM2 (LM2-MB) is analyzed precisely.

In general, the scheme described above does not guarantee that all verified symbols are actually correct. The event that a symbol is verified but incorrect is called false verification (FV). In order to guarantee there are no FVs, one can add a constraint on the signal such that the weighted sum, of any subset of a check node's non-zero neighbors, does not equal to zero [35] [34]. Another approach (e.g., taken in this chapter) is to consider random signals with continuous distributions so that FV occurs with probability zero. Finally, if the measured signal is assumed to be non-negative, then FV is impossible for the LM1 decoding algorithm.

Verification decoding was originally introduced and analyzed for the q-SC. It is based on the observation that, over large alphabets, the probability that "two independent random numbers are equal" is quite small. This leads to the *verification assumption* that any two matching values (during decoding) are generated by the same set of non-zero coefficients. The primary connection between CS, codes over real numbers, and verification decoding lies in the fact that:

> *The verification assumption applies equally well to both large discrete alphabets and the real numbers.*

2. Analysis Tools

Based on the bipartite graph structure, LDPC codes can be decoded efficiently using iterative MP algorithms. The average performance of MP decoding algorithms can be analyzed with density evolution (DE) [40] or extrinsic information transfer (EXIT) charts [44]. The concentration theorem [40] shows that random realizations of decoding are close to the average behavior w.h.p. as the block length goes to

infinity. DE analysis provides a threshold below which decoding (or reconstruction) succeeds w.h.p.. The decoding threshold can be improved by optimizing the edge degree distribution (d.d.) pair $\lambda(x)$ and $\rho(x)$.

Decoding can also be analyzed using combinatorial methods such as stopping set analysis [41] and [42]. Stopping set analysis gives a threshold below which *all* error patterns can be recovered with certainty under the assumption of no FV. Note that DE and stopping set analysis lead to different thresholds in general. Since stopping set analysis implies uniform recovery of all the error patterns, instead of just most of them, the threshold given by stopping set analysis is always lower than the one given by DE. For example, DE analysis of $(3, 6)$ regular codes on the BEC shows that erasure patterns with size less than 43% of the code length can be corrected w.h.p. [22]. On the other hand, the result from stopping set analysis guarantees that most codes correct all erasure patterns with size less than 1.8% of the code length when $n \to \infty$.

Likewise, in CS systems, there are two standard measures of reconstruction: *uniform* reconstruction and *randomized* (or *non-uniform*) reconstruction. A CS system achieves randomized reconstruction if *most* randomly chosen measurement matrices recover *most* of the signals in the signal set. While a CS system achieves uniform reconstruction if a measurement matrix and the decoder recover *all* the signals in the signal set with certainty. Another criterion, which is between uniform reconstruction and randomized reconstruction, is what we call *uniform-in-probability* reconstruction. A CS system achieves uniform-in-probability reconstruction if, for any signal in the signal set, *most* randomly chosen measurement matrices achieve successful decoding.

Since DE and the concentration theorem lead to w.h.p. statements for MP decoding over all signals and graphs, it is natural to adopt a DE analysis to evaluate the performance of randomized reconstruction CS systems based on LDPC codes.

For uniform reconstruction, a stopping set analysis of the MP decoder is the natural choice. While this works for the BEC, the possibility of FV prevents this type of strong statement for verification decoding. If the non-zero entries of Φ are chosen randomly from a continuous distribution, however, then the probability of FV is zero for all signals. Therefore, one can use stopping set analysis to analyze MP decoding of LDPC code ensembles and show that the LDPC codes with MP decoding achieves uniform-in-probability reconstruction in the CS system. The reader is cautioned that these results are somewhat brittle, however, because they rely on exact calculation and measurement of real numbers.

Understanding CS systems requires one to consider how the system parameters (e.g., the number of measurements and the sparsity of the signal) scale in the regime where the signal is both high-dimensional and extremely sparse. To compare results, we focus on the *oversampling ratio* (i.e., the number of measurements divided by the number of non-zero elements in the signal) required for reconstruction. This leads us to a scaling law approach to the standard DE and stopping set analysis.

3. Decoding Algorithms

In CS, optimal decoding (in terms of oversampling ratio) requires a combinatorial search that is known to be NP-Hard [45]. Practical reconstruction algorithms tend to either be based on linear programming (e.g., basis pursuit (BP) [27]) or low-complexity iterative algorithms (e.g., Orthogonal Matching Pursuit (OMP) [46]). A wide range of algorithms allow one to trade-off the oversampling ratio for reconstruction complexity. In [34], LDPC codes are used in the CS system and the algorithm is essentially identical to the verification based decoding proposed in [68]. The scaling law analysis shows the oversampling ratio for LDPC codes based CS system can be quite good. Encoding/decoding complexity is also a consideration. LDPC codes have

sparse bipartite graph representation so that encoding and decoding algorithms with complexity linearly with the code length can be developed.

There are several existing MP decoding algorithms for LDPC codes over non-binary fields. In [22] and [47], an analysis is introduced to find provably capacity-achieving codes for erasure channels under MP decoding. Metzner presents a modified majority-logic decoder in [11] that is similar to verification decoding. Davey and MacKay develop and analyze a symbol-level MP decoder over small finite fields [71]. Two verification decoding algorithms for large discrete alphabets are proposed by Luby and Mitzenmacher in [68] and called LM1 and LM2 in this chapter. The two capacity-achieving algorithms presented by Shokrollahi and Wang in [15] and are denoted as SW1 and SW2 in this chapter. The list-message-passing (LMP) algorithm [69] provides a smooth trade-off between the performance and complexity of SW1 and SW2. These algorithms are summarized in [69].

One can get a rough idea of the performance of these algorithms by comparing their performance for the standard $(3, 6)$-regular LDPC code. A standard performance measure is the noise threshold (or sparsity threshold for CS) below which decoding succeeds with high probability. The threshold of the LM1 algorithm in this case is 0.169 . This means that a long random $(3, 6)$-regular LDPC code will correct a q-SC error pattern with high probability as long as the error rate is less than 0.169. Likewise, it means that using the same code for LM1 reconstruction of a strictly-sparse signal with succeed w.h.p. as long as the sparsity rate (i.e., fraction of non-zero elements) of the signal vector is less than 0.169. The LM2-MB algorithm improves this threshold to 0.210 and the LM2-NB algorithm further improves this threshold to 0.259 [69].

Likewise, the stopping-set analysis of the LM1 algorithm in Section V shows that a $(3, 6)$-regular code exists where LM1 succeeds (ignoring FV) for all error (or sparsity)

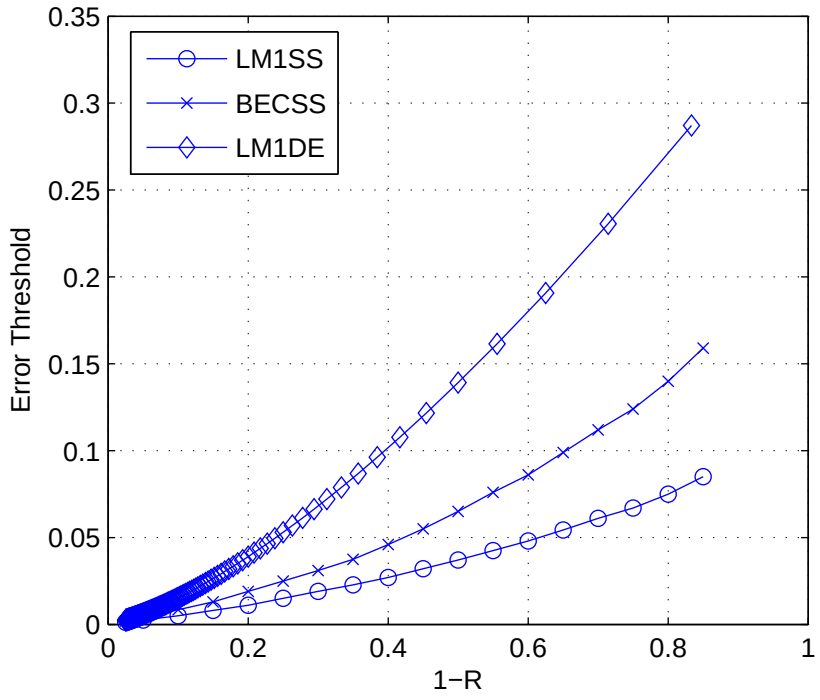

Fig. 6. Thresholds vs 1-R, where R is the code rate, for LM1 stopping set/DE analysis and the BEC stopping set analysis.

patterns whose fraction of non-zero entries is less than 0.0055. In comparison, the BEC stopping-set threshold of the $(3, 6)$ code is 0.018 for erasure patterns. However, both of these thresholds can be increased significantly (for the same code rate) by increasing the variable node degree. In fact, the $(7, 14)$-regular LDPC code gives the best (both LM1 and BEC) stopping-set thresholds and they are (respectively) 0.0364 and 0.0645. Finally, if the signal is non-negative, then FV is not possible during LM1 decoding and therefore 0.0364 is a lower bound on the true LM1 rate-$\frac{1}{2}$ threshold for uniform reconstruction. Fig. 6 shows the decoding/recovery thresholds for BEC stopping set analysis, LM1 stopping set analysis and LM1 DE analysis. Note that if the signal coefficients are non-negative, the threshold of LM1 given by stopping set analysis is comparable to the strong bound given in [48, Fig. 1(a)], and the threshold of LM1 given by DE analysis is comparable to the weak bound given in [48, Fig.

1(b)].

Since the scaling law analysis becomes somewhat tedious when complicated algorithms are applied, we consider only the (j, k)-regular code ensemble and the relatively simple algorithms LM1 and LM2-MB. The rather surprising result is that even with regular codes and simple decoding algorithms, the scaling law implies that LDPC codes with verification decoding perform very well for noiseless CS systems with strictly-sparse signals.

4. Signal Model

There are some significant differences between coding theory and CS. One of them is the signal model. The first difference is that coding theory typically uses discrete alphabets (see [49] for one exception to this) while CS deals with signals over the real numbers. Fortunately, some codes designed for large discrete alphabets (e.g., the q-ary symmetric channel) can be adapted to the real numbers. By exploring the connection and the analogy between real field and finite field with large q, the CS system can be seen as an essentially a syndrome-based source coding system [36]. Using the parity-check matrix of a non-binary LDPC code as the measurement matrix, the MP decoding algorithm can be used as the reconstruction algorithm.

The second difference in the signal model is that CS usually models the sparse signal $x \in \mathbb{R}^N$ as a random vector in an N-dimensional unit weak ℓ_r ball which is denoted as $U(\ell^N_{w,r})$. Note that the weak ℓ_r ball of radius M consists of vectors x whose decreasing rearrangement, denoted as x^*, satisfies $|x^*_1| \geq |x^*_2| \geq \cdots \geq |x^*_N|$ with $|x^*_i| \leq Mi^{-1/r}$. The constraint $x \in U(\ell^N_{w,r})$ defines the approximate sparsity property of the signal. As r approaches zero, the decreasing rearrangement of the x coefficients must decay faster. For decoding, the constants r and M provide a partial ordering of the set $\{x | x \in U(\ell^N_{w,0})\}$ and allow one to regularize the ill-conditioned

decoding problem.

In information theory and coding theory, the most commonly used signal model is a probabilistic model; the signal is treated as a random variable and the pdf is used to describe the signal. The sparsity of the signal can also be captured in the probabilistic model. For example, we can use the weighted sum of a Dirac delta function at zero and a very wide uniform distribution to model a strictly sparse signal.

5. Interesting Rate Regime

In coding theory, the code rate depends on the application and the interesting rate regime varies from close to zero to almost one. In CS systems, the signal is sparse in some domain and becomes increasingly sparse as the dimension increases. Intuitively, this means we can use codes with very little redundancy or very high code rate to represent the signal. So the interesting rate regime for CS systems is the high-rate regime. We consider the relationship between the system parameters and how they scale as the rate goes to one. The answer lies in the scaling law analysis developed in this chapter.

C. Main Results

The main results of this chapter are listed as follows. The details follow in section D and section E. Note that all results hold for randomly-chosen regular LDPC codes with variable-degree j and check-degree k. For a given j, we can increase k and observe how the decoding threshold scales. This provides a scaling law for the threshold and leads to conditions for successful reconstruction and the converse. One consequence of this is that randomized reconstruction can be achieved, for CS of strictly-sparse signals, when the number of measurements scales linearly with the sparsity of the

signal.

(i) [DE-BEC] For the BEC, there is a $K < \infty$ such that: a check-regular LDPC codes with average variable node degree $j \geq 2$ and check-degree k can recover a $\delta < \bar{\alpha}_j j/(k-1)$ fraction of erasures (w.h.p. as $n \to \infty$) when $k \geq K$. The constant $\bar{\alpha}_j$ (independent of k) is essentially the fraction of the optimal $\delta^* = j/k$ achieved as $n \to \infty$ and the rate goes to one. Conversely, if the erasure probability $\delta > \bar{\alpha}_j j/(k-1)$, then decoding fails (w.h.p. as $n \to \infty$) for all k.

(ii) [SS-BEC] For any $0 \leq \theta < 1$, there is a $K < \infty$ such that: for all $k \geq K$, a (j,k)-regular LDPC code with $j \geq 3$ can recover all erasures (w.h.p. as $n \to \infty$) of size $\theta ne\,(k-1)^{-j/(j-2)}$.

(iii) [DE-q-SC-LM1] For the q-SC, when one chooses a code randomly from the (j,k) regular ensemble with $j \geq 2$ and uses LM1 as decoding algorithm, then there is a $K_1 < \infty$ such that one can recover almost all error patterns of size $n\delta$ for $\delta < \bar{\alpha}_j (k-1)^{-j/(j-1)}$ (w.h.p. as $n \to \infty$) for all $k \geq K_1$. Conversely, when $\delta > \bar{\alpha}_j (k-1)^{-j/(j-1)}$, there is a $K_2 < \infty$ such that the decoder fails (w.h.p. as $n \to \infty$) for all $k \geq K_2$.

(iv) [DE-q-SC-LM2-MB] For the q-SC, when one chooses a code randomly from the (j,k) regular ensemble with $j \geq 3$ and uses LM2-MB as decoding algorithm, then there is a $K_1 < \infty$ such that one can recover almost all error patterns of size $n\delta$ for $\delta < \bar{\alpha}_j j/k$ (w.h.p. as $n \to \infty$). The constant $\bar{\alpha}_j$ (independent of k) is essentially the fraction of the optimal $\delta^* = j/k$ achieved as the rate goes to one. Conversely, there is a $K_2 < \infty$ such that the decoder fails (w.h.p. as $n \to \infty$) when $\delta > \bar{\alpha}_j j/k$ for all $k \geq K_2$.

(v) [SS-q-SC-LM1] For any $0 \leq \theta < 1$, there is a $K < \infty$ such that: For all $k \geq K$, a (j,k)-regular LDPC code with $j \geq 3$ using LM1 decoding can recover (w.h.p as $n \to \infty$) all q-SC error patterns of size $\theta n \bar{\beta}_j (k-1)^{-j/(j-2)}$ if no false

verifications occur.

Note that the constants K, K_1, K_2 and $\bar{\alpha}_j$ in (i), (ii), (iii), (iv) and (v) are different, but for the simplicity of expression, we use the same notation.

The rest of the chapter is organized as follows. In Section D, we derive the high-rate scaling based on DE analysis. We first show the high-rate scaling analysis for the BEC under MP decoding. Then, the analysis of high-rate scaling for the q-SC with LM1 and LM2-MB decoding algorithms is shown. In Section E, we derive the stopping set analysis for the q-SC with LM1 decoding algorithm and the high-rate scaling. The simulation results are shown in Section D.

D. High-Rate Scaling Via Density Evolution

1. DE Scaling Law Analysis for the BEC

DE analysis provides an explicit recursion which connects the distributions of messages passed from variable nodes to check nodes at two consecutive iterations of MP algorithms. In the case of BEC, this task has been accomplished in [50] and [22]. It has been shown that the expected fraction of erasure messages which are passed in the i-th iteration, called x_i, evolves as $x_i = \delta\lambda(1 - \rho(1 - x_{i-1}))$ where δ is the erasure probability of the channel. For general channels, the recursion may be much more complicated because one has to track the general distributions which cannot be represented by a single parameter [17].

To illustrate the scaling law, we start by analyzing the BEC case using DE. Although this is not applicable to CS, it motivates the scaling law analysis for the q-SC which is related to CS.

The scaling law of LDPC codes of check-regular ensemble over the BEC is shown by the following theorem.

Theorem 3. *Consider a sequence of check-regular LDPC codes with fixed variable node degree distribution $\lambda(x)$ and increasing check degree k. Let $j = 1/\int_0^1 \lambda(x)dx$ be the average symbol degree and $\overline{\alpha}_j$, which is called α-threshold, be the largest α such that $\lambda\left(1-e^{-\alpha j x}\right) \leq x$ for $x \in (0,1]$. For the erasure probability $\delta = \alpha j/(k-1)$, the iterative decoding of a randomly chosen length-n code from this ensemble fails (w.h.p as $n \to \infty$) for all k if $\alpha > \overline{\alpha}_j$. Conversely, if $\alpha < \overline{\alpha}_j$, then there exists a $K < \infty$ such that iterative decoding succeeds (w.h.p as $n \to \infty$) for all $k \geq K$.*

Lemma 3. *For all $s \geq 0$ and $k^{1+s} > |x|$, the sequence $a_k = \left(1 - \frac{x}{k^{1+s}}\right)^k$ is strictly increasing in k and*

$$1 - xk^{-s} \leq a_k \leq e^{-xk^{-s}}. \tag{3.1}$$

Proof of Lemma 3. We restrict our attention to $x \geq 0$ because the proof is simplified in this case and the continuation does not require $x < 0$. We show that a_k is strictly increasing with k by considering the power series expansion of $\ln a_k$, which converges if $k^{1+s} > |x|$. This gives

$$\ln a_k = k \ln\left(1 - \frac{x}{k^{1+s}}\right) = -xk^{-s} - \sum_{i=2}^{\infty} \frac{x^i}{i\, k^{(1+s)i-1}}, \tag{3.2}$$

and keeping only the first term shows that $\ln a_k \leq -xk^{-s}$. Since all the terms are negative and decreasing with k, we see that a_k is strictly increasing with k. Since a_k is convex in x for $k^{1+s} > |x|$, the lower bound $a_k \geq 1 - xk^{-s}$ follows from tangent lower bound at $x = 0$. □

Proof of Theorem 3. Using the change of variable, $x_i = \frac{\overline{\alpha}_j j}{k-1} y_i$, the DE recursion can be scaled to get

$$y_{i+1} = f_k\left(y_i\right) \triangleq \frac{\alpha}{\overline{\alpha}_j} \lambda\left(1 - \left(1 - \frac{\overline{\alpha}_j j y_i}{k-1}\right)^{k-1}\right). \tag{3.3}$$

By Lemma 3, $(1 - \frac{x}{k-1})^{k-1}$ increases monotonically (for $x \leq k-1$) to e^{-x}, and we see

that $f_k(y)$ decreases monotonically to $f_*(y) = \frac{\alpha}{\overline{\alpha}_j}\lambda\left(1 - e^{-\overline{\alpha}_j j y}\right)$. If $\alpha > \overline{\alpha}_j$, then (by the definition of $\overline{\alpha}_j$) $f_*(y) > y$ for some $y \in (0, 1]$. Since $f_k(y) \geq f_*(y)$, the recursion $y_{i+1} = f_k\left(y_i\right)$ will not converge to zero (from $y_0 = 1$) and iterative decoding will fail for all k w.h.p. as $n \to \infty$.

If $\alpha < \overline{\alpha}_j$, then $f_*(y) < y$ for $y \in (0, 1]$. Since $f_k(y) \searrow f_*(y)$, we can choose $K < \infty$ to be the first k such that $f_k(y) < y$ for $y \in (0, 1]$. In this case, the recursion $y_{i+1} = f_k\left(y_i\right)$ will converge to zero (from $y_0 = 1$) for all $k \geq K$ and iterative decoding will succeed w.h.p. as $n \to \infty$. □

The following proposition determines a few α-thresholds explicitly.

Proposition 1. *For (j, k) regular LDPC codes, the α-threshold is given by $\overline{\alpha}_j$ with $\overline{\alpha}_2 = 0.5$, $\overline{\alpha}_3 \approx 0.8184$, and $\overline{\alpha}_4 \approx 0.7722$.*

Proof. See Appendix E. □

Remark 5. *For example, if $j = 3$ and $\alpha = 0.75 < 0.8184 = \overline{\alpha}_3$, then numerical results show that $K = 9$ suffices so that DE converges for all $k \geq 9$ when $\delta < 3(0.75)/(k-1)$.*

2. Concentration Theorem for the BEC

Note that in the proof of Theorem 3, we make use of the standard concentration theorem [17] for MP decoding. For example, the "w.h.p." statements rely on the concentration theorem. However, the concentration theorem for the BEC DE analysis proved in [17] holds only for fixed k (i.e., k is independent of n). In many CS applications, the desired number of measurements is sublinear in the block length (i.e., $k = n^{\omega}$ with $\omega < 1$). In this section, we discuss how the proof of the concentration theorem can be modified to handle this case.

The concentration theorem [17] shows the performance of a randomly chosen

code concentrates around its expected value so that

$$\Pr\left\{\left|\frac{Z^{(\ell)}}{n_e} - p^{(\ell)}\right| \geq \epsilon\right\} \leq 2e^{-\beta\epsilon^2 n},$$

where $Z^{(\ell)}$ is the r.v. which is equal to the number of variable-to-check erasure messages in the ℓ-th iteration, $p^{(\ell)} = E\left[Z^{(\ell)}\right]/n_e$ is the expected erasure rate, n_e is the number of edges in the graph, and β is a constant which depends on the degree distribution. Let $T_{\ell,i}$ be the number of edges in the depth-2ℓ directed neighborhood of the i-th edge, then the constant β can be chosen to be $a/\max_i T_{\ell,i}$, where a is a constant independent of n and ℓ. For regular ensembles, the number $T_{\ell,i}$ is independent of i and this implies $\beta = \Theta\left(1/(j^{2\ell}k^{2\ell})\right)$. Concentration occurs as $n \to \infty$ for fixed j, k and ℓ.

In this section, we consider the case where the check-node degree increases with n. In this case, the concentration theorem is not informative if k grows faster than $n^{1/(2\ell)}$. But, the following theorem shows that the concentration theorem can be modified to handle the scaling law.

Theorem 4. *For check-degree $k = n^{\omega}$ and erasure rate $\delta = \frac{\alpha j}{k}$, the performance (in terms of the fraction of erasure messages) of a code randomly chosen from the (j,k)-regular ensemble over BEC concentrates around its expected value exponentially in $n^{\frac{1-\omega}{1+2\ell}}$ where ℓ is the number of decoding iterations. Mathematically, there exist constants $\beta'(j,\omega,\ell)$ and $N(j,\omega,\ell)$ such that*

$$\Pr\left\{\left|\frac{Z^{(\ell)}}{nj} - p^{(\ell)}\right| \geq \epsilon\right\} \leq 2e^{-\beta' n^{\frac{1-\omega}{1+2\ell}}\epsilon^2} \tag{3.4}$$

for any $\epsilon > 0$ and $n > N$.

Proof. See the Appendix F. □

3. DE Scaling Law Analysis for the q-SC

a. DE Scaling Law Analysis for LM1

For the simplicity of our analysis, we only consider (j,k)-regular code ensemble and the LM1 decoding algorithm [68] for the q-SC with error probability δ. The DE recursion for LM1 is (from [68])

$$x_{i+1}=\delta\left(1-\left[1-(1-\delta)\left(1-(1-x_i)^{k-1}\right)^{j-1}-x_i\right]^{k-1}\right)^{j-1}, \tag{3.5}$$

where x_i is the fraction of unverified messages in the i-th iteration. Our analysis of the scaling law relies on the following lemma.

Lemma 4. *Let the functions $g_{k+1}(x)$ and $\overline{g}_{k+1}(x)$ be defined by*

$$g_{k+1}(x) \triangleq \frac{\alpha}{\overline{\alpha}_j}\left(1-\left[1-\left(1-\frac{\alpha}{k^{j/(j-1)}}\right)\left(1-\left(1-\frac{\overline{\alpha}_j x}{k^{j/(j-1)}}\right)^k\right)^{j-1}-\frac{\overline{\alpha}_j x}{k^{j/(j-1)}}\right]^k\right)^{j-1}$$

and

$$\overline{g}_{k+1}(x) \triangleq \frac{\alpha}{\overline{\alpha}_j}\left(1-\left[1-\frac{\overline{\alpha}_j^{j-1}x^{j-1}}{k}-\frac{\overline{\alpha}_j x}{k^{j/(j-1)}}\right]^k\right)^{j-1},$$

where $\overline{\alpha}_j \geq 1$, $\alpha \in (0,\overline{\alpha}_j]$, and $j \geq 2$. For $x \in (0,1]$ and $k > \overline{\alpha}_j^{j-1}$, these functions satisfy (i) $g_k(x) \leq \overline{g}_k(x)$, (ii) $\overline{g}_k(x)$ is monotonically decreasing with k for $k > \overline{\alpha}_j^{j-1}$, and (iii) $\lim_{k\to\infty} g_k(x) = \lim_{k\to\infty} \overline{g}_k(x) = \frac{\alpha}{\overline{\alpha}_j}\left(1-e^{-\overline{\alpha}_j^{j-1}x^{j-1}}\right)^{j-1}$.

Proof. See the Appendix G. □

Theorem 5. *Consider a sequence of (j,k)-regular LDPC codes with fixed symbol degree $j \geq 2$ and increasing check degree k. Let $\bar{\alpha}_j$ be the largest α such that $(1-$*

$e^{-\alpha^{j-1}x^{j-1}})^{j-1} \leq x$ *for* $x \in (0,1]$. *If the sparsity of the signal is* $n\delta$ *for* $\delta = \alpha(k-1)^{-j/(j-1)}$ *and* $\alpha < \bar{\alpha}_j$, *then there exists a* K_1 *such that by randomly choosing a length-*n *code from the* (j,k) *regular LDPC code ensemble, LM1 reconstruction succeeds (w.h.p as* $n \to \infty$*) for all* $k \geq K_1$. *Conversely, if* $\alpha > \bar{\alpha}_j$ *then there exists a* K_2 *such that LM1 reconstruction fails (w.h.p as* $n \to \infty$*) for all* $k \geq K_2$.

Proof. Scaling (3.5) using the change of variables $\delta = \alpha(k-1)^{-j/(j-1)}$ and $x_i = \overline{\alpha}_j y_i (k-1)^{-j/(j-1)}$ gives $y_{i+1} = g_k(y_i)$. The function $\overline{g}_k(x)$ also allows us to define the upper bound $z_{i+1} = \overline{g}_k(z_i)$ where $z_i \leq y_i$ implies $z_{i+1} \leq y_{i+1}$.

Since $\left(1 - \frac{x}{k}\right)^k$ increases monotonically to e^{-x}, we see that $\overline{g}_k(y)$ decreases monotonically to $g_*(y)$. If $\alpha < \overline{\alpha}_j$, then $g_*(y) < y$ for all $y \in (0,1]$. Since $g_k(y) \leq \overline{g}_k(y) \searrow g_*(y)$, we can choose $K_1 < \infty$ to be the first k such that $\overline{g}_k(y) < y$ for all $y \in (0,1]$. In this case, the recursion $y_{i+1} = g_k(y_i)$ will converge to zero (from $y_0 = 1$) for all $k \geq K_1$ and iterative decoding will succeed w.h.p. as $n \to \infty$.

If $\alpha > \overline{\alpha}_j$, then (by the definition of $\overline{\alpha}_j$) $g_*(y) > y$ for some $y \in (0,1]$. Since $\lim_{k\to\infty} g_k(y) = g_*(y)$, there exists a K_2 such that, for all $k \geq K_2$, the recursion $y_{i+1} = g_k(y_i)$ will not converge to zero (from $y_0 = 1$) and iterative decoding will fail w.h.p. as $n \to \infty$. □

Remark 6. *If a randomly chosen code from the* (j,k) *regular ensemble is applied to a CS system with LM1 reconstruction, then randomized reconstruction succeeds (w.h.p as* $n \to \infty$*) when the sparsity is* $n\delta$ *with* $\delta < \bar{\alpha}_j(k-1)^{-j/(j-1)}$. *This requires* $m = \gamma n\delta$ *measurements with an oversampling ratio of* $\gamma > \gamma_0 = \bar{\alpha}_j^{-(j-1)/j}\delta^{-1/j}j$.

The following lemma shows how to calculate the constants in front of the scaling laws.

Corollary 2. $\bar{\alpha}_2 = 1$, $\bar{\alpha}_3 \approx 1.87321$, $\bar{\alpha}_4 \approx 1.66455$ *and* $\bar{\alpha}_5 \approx 1.52073$.

Proof. See Appendix H. □

Corollary 3. *For regular LDPC codes and LM1 reconstruction, choosing* $j = \lceil \ln \frac{1}{\delta} \rceil$ *gives a uniform lower bound on the oversampling ratio (as* $\delta \to 0$*) of* $\lceil \ln \frac{1}{\delta} \rceil$ *e.*

Proof. The minimum oversampling ratio $\gamma_0 = \bar{\alpha}_j^{-\frac{j-1}{j}} j\delta^{-1/j} \leq j\delta^{-1/j}$ and we choose $j = \lceil \ln \frac{1}{\delta} \rceil$, taking the logarithm of both sides shows that

$$\ln \gamma_0 \leq \ln \left\lceil \ln \frac{1}{\delta} \right\rceil + \frac{1}{\lceil \ln \frac{1}{\delta} \rceil} \ln \frac{1}{\delta} \leq \ln \left\lceil \ln \frac{1}{\delta} \right\rceil + 1. \tag{3.6}$$

□

b. Scaling Law Analysis Based on DE for LM2-MB

For the second algorithm in [68], the DE recursion for the fraction x_i of unverified messages in the i-th iteration is

$$x_{i+1} = \delta\Bigg(\lambda\left(1-\rho\left(1-x_i\right)\right) + \lambda^{'}\left(1-\rho\left(1-x_i\right)\right) \left(\rho\left(1-x_i\right) - \rho\Big(1-\left(1-\delta\right)\lambda\left(1-\rho\left(1-x_i\right)\right) - x_i\Big)\right)\Bigg). \tag{3.7}$$

Like the analysis of LM1, we first introduce a lemma to bound the scaled DE equation.

Lemma 5. *The functions* $g_k(x)$ *and* $\bar{g}_k(x)$ *are defined as*

$$g_k(x) \triangleq \frac{\alpha}{\bar{\alpha}_j}\Bigg(\Big(s(x)\Big)^{j-1} + (j-1)\Big(s(x)\Big)^{j-2}\left(1-\frac{\alpha j x}{k}\right)^{k-1} \left(1-\left(1-\frac{1-\frac{\alpha j}{k}}{1-\frac{\alpha j x}{k}}\Big(s(x)\Big)^{j-1}\right)^{k-1}\right)\Bigg),$$

where $s(x) = 1 - \left(1 - \frac{\alpha j x}{k}\right)^{k-1}$*, i.e.,* $1 - \rho(1-y)$*, and*

$$\bar{g}_k(x) \triangleq \frac{\alpha}{\bar{\alpha}_j}\left(\left(1-\left(1-\frac{\alpha j x}{k}\right)^k\right)^{j-1}+(j-1)\left(1-\left(1-\frac{\alpha j x}{k}\right)^k\right)^{j-2}\left(1-\left(1-\frac{\alpha j x}{k}\right)^k\right)\right).$$

For $x \in (0,1]$ and $k > \alpha$, these functions satisfy (i) $\bar{g}_k(x) > g_k(x)$, (ii) $\lim_{k\to\infty} g_k(x) = \lim_{k\to\infty} \overline{g}_k(x) = g_*(x)$ where

$$g_*(x) \triangleq \frac{\alpha}{\bar{\alpha}_j}\left(1-e^{-\alpha j x}\right)^{j-2}\left(1+(j-2)e^{-\alpha j x}\right), \tag{3.8}$$

and (iii) $\bar{g}_k(x)$ is a monotonically decreasing function of k.

Proof. See the Appendix I. □

Theorem 6. *Consider a sequence of (j,k)-regular LDPC codes with variable node degree $j \geq 3$. Let $\overline{\alpha}_j$ be the largest α such that $(1-e^{-\alpha j x})^{j-2}(1+(j-2)e^{-\alpha j x}) \leq x$ for $x \in (0,1]$. If the sparsity of the signal is $n\delta$ with $\delta = \alpha j/k$ and $\alpha < \overline{\alpha}_j$, then there exists a K_1 such that LM2-MB reconstruction succeeds (w.h.p as $n \to \infty$) for all $k \geq K_1$. Conversely, if $\alpha > \overline{\alpha}_j$ then there exists a K_2 such that LM2-MB decoding fails (w.h.p as $n \to \infty$) for all $k \geq K_2$.*

Proof. The LM2-MB DE recursion is given by (3.7). Using the change of variables $x_i = \frac{\overline{\alpha}_j j}{k} y_i$ and $\delta = \frac{\alpha j}{k}$, the scaled DE equation can be written as $y_{i+1} = g_k(y_i)$. Taking the limit as $k \to \infty$ gives $y_{i+1} = g_*(y_i)$.

If $\alpha < \overline{\alpha}_j$, then the definition of $\bar{\alpha}_j$ implies that $g_*(y) < y$ for $y \in (0,1]$. Since $g_k(y) \leq \overline{g}_k(y) \searrow g_*(y)$ (by Lemma 5), we can choose $K_1 < \infty$ to be the first k such that $\overline{g}_k(y) < y$ for $y \in (0,1]$. In this case, the recursion $y_{i+1} = g_k(y_i)$ will converge to zero (from $y_0 = 1$) for all $k \geq K_1$ and iterative decoding will succeed w.h.p. as

$n \to \infty$.

If $\alpha > \overline{\alpha}_j$, then (by the definition of $\overline{\alpha}_j$) $g_*(y) > y$ for some $y \in (0, 1]$. In this case, there is a K_2 and y such that $g_k(y) > y$ for all $k \geq K_2$, and the recursion $y_{i+1} = g_k(y_i)$ does not converge to zero (from $y_0 = 1$) and iterative decoding will fail w.h.p. as $n \to \infty$.

For $j = 2$, the quantity $\overline{\alpha}_2$ is undefined because $(1 - e^{-\alpha j x})^{j-2}(1 + (j-2)e^{-\alpha j x}) = 1$. This implies that $(2, k)$ regular LDPC codes do not obey this scaling law of LM2-MB decoding. □

Remark 7. *If a randomly chosen code from the (j, k) regular ensemble is applied to a CS system with LM2-MB reconstruction, then randomized reconstruction succeeds (w.h.p as $n \to \infty$) when the sparsity is $n\delta$ with $\delta < \bar{\alpha}_j j/k$. This requires $m \geq \gamma n \delta$ measurements and an oversampling ratio of $\gamma > \gamma_0 = 1/\bar{\alpha}_j$.*

Remark 8. *For (j, k) regular LDPC codes, the α-threshold of LM2-MB is given by $\overline{\alpha}_j$ and can be calculated numerically to get $\overline{\alpha}_3 = \frac{1}{6}$, $\overline{\alpha}_4 \approx 0.34$ and $\overline{\alpha}_5 \approx 0.37$.*

The interesting part of this result is that the number of measurements needed for randomized *reconstruction with LM2-MB scales linearly with the sparsity* of the signal. All previous reconstruction methods with reasonable complexity require a super-linear number of measurements.

4. Concentration Theorem for the q-SC

Note that the proof of Theorem 5 also depends on the standard concentration theorem in [17]. Like the BEC case, the result becomes non-informative if k grows faster than $n^{1/(2\ell)}$. Therefore, we conjecture (i.e., give without proof) the following concentration result for LM1 decoding for a randomly chosen code and q-SC error pattern. For $k = n^{\omega}$ and $\delta = \alpha(k-1)^{-j/(j-1)}$, the fraction of unverified messages after ℓ iterations

concentrates around its expected value $p^{(\ell)}$. Mathematically, there exist constants $\beta'(j,\omega,,\ell)$ and $N(j,\omega,\ell)$ such that

$$\Pr\left\{\left|\frac{Z^{(\ell)}}{nj}-p^{(\ell)}\right|\geq\epsilon\right\}\leq 2e^{-\beta' n^{\frac{1-\omega}{1+2\ell}}\epsilon^2}$$

for any $\epsilon>0$ and $n>N$.

To show the concentration theorem, the idea is similar with the one used in the BEC case. We first reduce the graph to a smaller one by removing the verified edges in the first iteration, then prove the concentration for the residual graph. The probability that an edge is not removed in the first iteration is

$$\delta+(1-\delta)\left((1-(1-\delta)^{k-1}\right)^j,$$

so the average number of edges which are not removed in the first iteration is

$$nj\left(\delta+(1-\delta)\left((1-(1-\delta)^{k-1}\right)^j\right).$$

Note that the average number of check nodes which are not removed is $\frac{nj}{k}(1-(1-\delta)^k)$. Therefore, the normalized (by the number edges connected to the remaining check nodes) probability of an edge to remain is

$$\begin{aligned}\epsilon &\triangleq \frac{nj\left(\delta+(1-\delta)\left((1-(1-\delta)^{k-1}\right)^j\right)}{k\frac{nj}{k}\left(1-(1-\delta)^k\right)}\\ &= \frac{\delta+(1-\delta)\left(1-(1-\delta)^{k-1}\right)^j}{1-(1-\delta)^k}.\end{aligned}\tag{3.9}$$

If we substitute $\alpha(k-1)^{-j/(j-1)}$ for δ in (3.9) and use a Taylor expansion around $k=\infty$, we have $\epsilon=\frac{1}{k}+o\left(\frac{1}{k}\right)$. So the probability that a check node has degree t after removal is $\binom{k}{t}\epsilon^t(1-\epsilon)^{k-t}$ which converges to Poisson distribution with mean 1 as $k\to\infty$. Following the approach used for the BEC case shows that the concentration

theorem also holds for LM1 on the q-SC.

E. Scaling Laws Based on Stopping Set Analysis

DE analysis provides the threshold below which the *randomized* (or *non-uniform*) recovery is guaranteed, in the following sense: the signal and the measurement matrix are both chosen randomly, and w.h.p. the reconstruction algorithm gives the correct answer. If the reconstruction algorithm is guaranteed to succeed for all signals of sufficient sparsity, this is called *uniform* recovery. On the other hand, if reconstruction algorithm is uniform over all support sets of sufficient sparsity, but succeeds w.h.p. over the amplitudes of the non-zero elements (i.e., has a small but non-zero failure probability based on amplitudes), then the reconstruction is called *uniform-in-probability* recovery.

According to the analysis in section D, we know that the number of measurements needed for randomized recovery by using LM2-MB scales linearly with the sparsity of the signal. Still, the reconstruction algorithm may fail due to the support set (e.g., it reaches a stopping set) or due to the non-zero amplitudes of the signal (e.g., a false verification occurs).

In this section, we will analyze the performance of MP decoding algorithms with uniform-in-probability recovery in the high-rate regime. This follows from a stopping set analysis of the decoding algorithms. A stopping set is defined as an erasure pattern (or internal decoder state) from which the decoding algorithm makes no further progress. Following the definition in [41], we let $G = (V \cup C, E)$ be the Tanner graph of a code where V is the set of variable nodes, C is the set of check nodes and E is the set of edges between V and C. A subset $U \subseteq V$ is a BEC stopping set if no check node is connected to U via a single edge. The scaling law below uses

the average stopping-set enumerator for LDPC codes as a starting point.

1. Scaling Law Analysis for Stopping Sets on the BEC

The *average stopping set distribution* $E_{n,j,k}(s)$ is defined as the average (over the ensemble) number of stopping sets with size s in a randomly chosen (j,k) regular code with n variable nodes. The *normalized stopping set distribution* $\gamma_{j,k}(\alpha)$ is defined as $\gamma_{j,k}(\alpha) \triangleq \lim_{n\to\infty} \frac{1}{n} \ln E_{n,j,k}(n\alpha)$. The *critical stopping ratio* $\alpha^*_{j,k}$ is defined as $\alpha^*_{j,k} \triangleq \inf\{\alpha > 0 : \gamma_{j,k}(\alpha) \geq 0\}$. Intuitively, when the normalized size of the stopping set is greater than or equal to $\alpha^*_{j,k}$, the average number of stopping sets grows exponentially with n. When the normalized size of the stopping set is less than $\alpha^*_{j,k}$, the average number of stopping sets decays exponentially with n. In fact, there exist codes with no stopping sets of normalized size less than $\alpha^*_{j,k}$. Therefore, the quantity $\alpha^*_{j,k}$ can also be thought of as a *deterministic decoding threshold.*

The normalized average stopping set distribution $\gamma_{j,k}(\alpha)$ for (j,k) regular ensembles on the BEC is given by [42]

$$\gamma_{j,k}(\alpha) \leq \gamma_{j,k}(\alpha; x) \triangleq \frac{j}{k} \ln \left(\frac{(1+x)^k - kx}{x^{k\alpha}} \right) - (j-1)h(\alpha),$$

where $h(\cdot)$ is the entropy of a binary distribution and the bound holds for any $0 \leq x \leq 1$. The optimal value x_0 is the unique positive solution of

$$\frac{x((1+x)^{k-1} - 1)}{(1+x)^k - kx} = \alpha. \tag{3.10}$$

This gives the following theorem.

Theorem 7. *For any $0 \leq \theta < 1$, there is a $K < \infty$ such that, for all $k \geq K$, a randomly chosen (j,k) regular LDPC code ($j \geq 3$) will (w.h.p. as $n \to \infty$) correct all erasure patterns of size less than $\theta ne(k-1)^{-j/(j-2)}$.*

Sketch of Proof. Since there is no explicit solution for x_0, we use a 2nd order expansion of the LHS of (3.10) around $x = 0$ and solve for x. This gives $x_0 = \sqrt{\frac{\alpha}{k-1}} + o(\alpha)$. Since $\gamma_{j,k}(\alpha) \leq \gamma_{j,k}(\alpha, x)$ holds for all $x \geq 0$, we have

$$\gamma_{j,k}(\alpha) \leq \frac{j}{k} \ln \left(\frac{\left(1+\sqrt{\frac{\alpha}{k-1}}\right)^k - k\sqrt{\frac{\alpha}{k-1}}}{\frac{\alpha}{k-1}^{\frac{k\alpha}{2}}} \right) - (j-1)h(\alpha). \tag{3.11}$$

Next we expand the RHS of (3.11) around $\alpha = 0$ and neglect the high order terms; solving for α gives an upper bound on the critical stopping ratio

$$\alpha^*_{j,k} \leq \exp\left(\frac{j - 2 - j\ln(k-1)}{j-2} \right).$$

It can be shown that this bound on $\alpha^*_{j,k}$ is tight as $k \to \infty$. This means that, for any $0 \leq \theta < 1$, there is a K such that $\theta e(k-1)^{-j/(j-2)} \leq \alpha^*_{j,k} \leq e(k-1)^{-j/(j-2)}$ for all $k > K$. Therefore, the critical stopping ratio $\alpha^*_{j,k}$ scales like $e(k-1)^{-j/(j-2)}$ as $k \to \infty$. □

Remark 9. *Although the threshold is strictly increasing with j, this ignores the fact that the code rate is decreasing with j. However, if one optimizes the oversampling ratio instead, then the choice of $j^* = 2 + 2\ln(k-1)$ is nearly optimal. Moreover, it leads to the simple formula $\alpha^*_{j^*,k} = \frac{1}{k-1}$ and an oversampling ratio which grows logarithmically with k.*

2. Stopping Set Analysis for the q-SC with LM1

A stopping set for LM1 is defined by considering a decoder where S, T, U are disjoint subsets of V corresponding to verified, correct, and incorrect variable nodes. Decoding progresses if and only if (i) a check node has all but one edge attached to S or (ii) a check node has all edges attached to $S \cup T$. Otherwise, the pattern is a stopping set. In the stopping set analysis for q-SC, we can define $E_{n,j,k}(\alpha, \beta)$ as the *average*

number of stopping sets with $|T| = n\alpha$ correctly received variable nodes and $|U| = n\beta$ incorrectly received variable nodes where n is the code length.

The average number of stopping sets $E_{n,j,k}(\alpha, \beta)$ can be computed by counting the number of ways, $S_{n,j,k}(a, b)$, that a correct variable nodes, b incorrect variables nodes, and $n - a - b$ verified variable nodes can be connected to $\frac{nj}{k}$ check nodes to form a stopping set. The number $S_{n,j,k}(a, b)$ can be computed using the generating function for one check,

$$g_k(x, y) \triangleq (1 + x + y)^k - ky - \left((1 + x)^k - 1\right),$$

which enumerates the number of edge connection patterns ("1" counts verified edges, "x" counts correct edges, and "y" counts incorrect edges) that prevent decoder progress. Generalizing the approach of [42] gives

$$E_{n,j,k}(\alpha, \beta) = \frac{\binom{n}{n\alpha, n\beta, n(1-\alpha-\beta)} S_{n,j,k}(\alpha n, \beta n)}{\binom{nj}{nj\alpha, nj\beta, nj(1-\alpha-\beta)}} \tag{3.12}$$

where

$$S_{n,j,k}(a, b) \triangleq \text{coeff}\left(g_k(x, y)^{nj/k}, x^{ja} y^{jb}\right).$$

For this work, we are mainly interested in largest β for which $E_{n,j,k}(\alpha, \beta)$ goes to zero as $n \to \infty$. Since the growth (or decay) rate of $E_{n,j,k}(\alpha, \beta)$ is exponential in n, this leads us to consider the *normalized average stopping set distribution* $\gamma_{j,k}(\alpha, \beta)$ which is defined as

$$\gamma_{j,k}(\alpha, \beta) = \lim_{n \to \infty} \frac{1}{n} \ln E_{n,j,k}(\alpha, \beta). \tag{3.13}$$

Likewise, the *critical stopping ratio* $\beta^*_{j,k}$ is defined as

$$\beta^*_{j,k} = \inf\{\beta \in [0,1] : w_{j,k}(\beta) > 0\} \tag{3.14}$$

where

$$w_{j,k}(\beta) \triangleq \sup_{\alpha \in [0,1-\beta]} \gamma_{j,k}(\alpha, \beta).$$

Note that $w_{j,k}(\beta)$ describes the asymptotic growth rate of the number of stopping sets with number of incorrectly received nodes $n\beta$. The number of stopping sets with size less than $n\beta^*_{j,k}$ decays exponentially with n and the ones with size larger than $n\beta^*_{j,k}$ grows exponentially with n.

Theorem 8. *The normalized average stopping set distribution $\gamma_{j,k}(\alpha, \beta)$ for LM1 can be bounded by*

$$\begin{gathered}\gamma_{j,k}(\alpha,\beta) \le \gamma_{j,k}(\alpha,\beta;x,y) \triangleq \\ \frac{j}{k}\ln\frac{\left(1+(1+x+y)^k - ky - (1+x)^k\right)}{x^{k\alpha}y^{k\beta}} \\ + (1-j)h(\alpha,\beta,1-\alpha-\beta)\end{gathered} \tag{3.15}$$

where the tightest bound is given by choosing (x, y) to be the unique positive solution of

$$\frac{x\left((1+x+y)^{k-1} - (1+x)^{k-1}\right)}{1+(1+x+y)^k - ky - (1+x)^k} = \alpha \tag{3.16}$$

and

$$\frac{y\left((1+x+y)^{k-1} - 1\right)}{1+(1+x+y)^k - ky - (1+x)^k} = \beta. \tag{3.17}$$

Proof. Starting from (3.12) and using Stirling's formula, it can be verified easily that

$$\lim_{n\to\infty}\frac{1}{n}\ln\frac{\binom{n}{n\alpha,n\beta,n(1-\alpha-\beta)}}{\binom{nj}{nj\alpha,nj\beta,nj(1-\alpha-\beta)}}=(1-j)h(\alpha,\beta,1-\alpha-\beta),$$

where $h(\cdot)$ is the entropy of a ternary distribution. Using a Chernoff-type bound for $S_{n,j,k}(a,b)$ (i.e., coeff$\,(f(x,y),x^iy^j)\le\frac{f(x,y)}{x^iy^j}$ for all $x,y>0$), we define

$$\psi_{j,k}(\alpha,\beta;x,y)\triangleq\frac{nj}{k}\ln\frac{\left(1+(1+x+y)^k-ky-(1+x)^k\right)}{x^{k\alpha}y^{k\beta}}.$$

Minimizing the bound over x,y gives

$$\gamma_{j,k}(\alpha,\beta)\le\gamma_{j,k}(\alpha,\beta;x,y)=$$
$$\psi_{j,k}(\alpha,\beta;x,y)+(1-j)h(\alpha,\beta,1-\alpha-\beta),$$

where (x,y) is the unique positive solution of (3.16) and (3.17). One can also show that the bound is exponentially tight in n. □

3. Scaling Law Analysis for LM1 Stopping Sets

In CS literature, we are only interested in the scenario that β is small. It means we need to perform stopping set analysis in the high-rate regime or to the signal vectors with sparse support. For the convenience of analysis, we only derive the analysis for (j,k) regular codes though it can be generalized to irregular codes [42]. In our analysis, the variable node degree j is fixed and the check node degree k is increasing. By calculating the scaling law of $w_{j,k}(\beta)$, we find the uniform-in-probability recovery decoding threshold $\beta^*_{j,k}$ which tells us the relationship between the minimum number of measurements needed for uniform-in-probability recovery and the sparsity of the signal.

The following theorem shows the scaling law of LM1 for the q-SC.

Theorem 9. *There is a code from (j,k) regular LDPC code ensemble and a constant K such that for the q-SC, all error patterns of size $n\delta$ for $\delta < \bar{\beta}_j(k-1)^{-j/(j-2)}$ can be recovered by LM1 (w.h.p. as $n \to \infty$) for $k \geq K$ where $\bar{\beta}_j$ is the unique positive root on c of the following implicit function*

$$v(d) = \frac{d}{2}\big((c-1)j\ln(1-c) - 2c\ln(c) + (1+c)(j-2)(-1+\ln d)\big) \tag{3.18}$$

where $d = (1-c)^{-j/(j-2)}c^{2/(j-2)}$.

Lemma 6. *Consider sequences of (x_k, y_k) given by (3.16) and (3.17) which satisfy $\beta_k = \Theta\left((k-1)^{-j/(j-2)}\right)$ as k goes to infinity. In this case, the implied x_k, y_k, and α_k all tend to zero.*

Proof. See the Appendix J. □

Lemma 7. *For the q-SC with LM1 decoding and $j \geq 3$, the average number of stopping sets with size sublinear in n goes to zero as $n \to \infty$. More precisely, for each $3 \leq j < k$ there exists a $\delta_{j,k} > 0$ such that*

$$\lim_{n\to\infty} \sum_{b=1}^{\delta_{j,k}n} \sum_{a=0}^{n-b} E_{n,j,k}\left(\frac{a}{n}, \frac{b}{n}\right) = 0.$$

Proof. See the Appendix K. □

Proof of Theorem 9. The main idea of the proof is to start from (3.15) and find a scaling law for $w_{j,k}(\beta)$ as k grows. Since $w_{j,k}(\beta)$ is the exponent of the average number of stopping sets and the resulting scaling function $v(d)$ is negative in the range $\big(0, \bar{\beta}_j\big)$, almost all codes have no stopping sets of size $n\delta$ with $0 < \delta < \bar{\beta}_j(k-1)^{-j/(j-2)}$. Because finding the limiting function of the scaled $w_{j,k}(\beta)$ is mathematically difficult,

we first find an upper bound on $w_{j,k}(\beta)$ and then analyze the limiting function of this upper bound.

Before we make any assumptions on the structure of x and y, we note that picking any x and y gives an upper bound of $\gamma_{j,k}(\alpha, \beta)$. To make the bound tight, we should pick good values for x and y. For example, the (x, y) which leads to the tightest bound is the positive solution of (3.16) and (3.17). Since we are free to choose the variables x and y as we like, we assume x and y decay with the same rate and are both equal to $o\left(\frac{1}{k-1}\right)$ so that the Taylor expansions of (3.16) and (3.17) converge.

Applying Taylor expansion for small x, y to (3.16) and (3.17), we have

$$\begin{aligned} xy(k-1) &\approx \alpha \\ (xy + y^2) &\approx \beta. \end{aligned}$$

Solving these equations for x and y gives the approximations

$$x_0 \approx \frac{\alpha}{\sqrt{(\beta-\alpha)(k-1)}} \qquad y_0 \approx \sqrt{\frac{\beta-\alpha}{k-1}}.$$

Next, we choose $\alpha = c\beta$ for $0 < c < 1$, which requires that $0 < \alpha < \beta$. [2] Applying these substitutions to (3.15) gives

$$\gamma_{j,k}\left(c\beta, \beta; \frac{c\beta}{\sqrt{\beta(1-c)(k-1)}}, \sqrt{\frac{\beta(1-c)}{k-1}}\right)$$

which equals

[2]The scaling regime we consider is $\beta = o(k^{-1})$ and this leads to the scaling of x, y. The scaling of x, y also implies that $0 < \alpha < \beta$. So we see that, although there exist stopping sets with $\alpha \geq \beta$, they do not occur in the scaling regime we consider.

$$\frac{\beta}{2}\Big((1+c)\,(2-j)\,(1-\ln(\beta)) - (1-c)\;j\;\ln(1-c) \\ -2\,c\,\ln(c) + (1+c)\;j\;\ln(-1+k)\Big) + O\left(\beta^{3/2}\right). \tag{3.19}$$

Plugging $\beta = d(k-1)^{-j/(j-2)}$ into this equation for $d \geq 0$ gives

$$\gamma_{j,k}(\alpha,\beta) \leq \frac{d}{2}(k-1)^{-j/(j-2)}\big((c-1)j\ln(1-c) - 2c\ln(c) \\ +(1+c)(2-j)(1-\ln d)\big) + O\left((k-1)^{-2j/(j-2)}\right). \tag{3.20}$$

Scaling the RHS of (3.20) by $(k-1)^{j/(j-2)}$ gives the limiting function

$$v(c,d) \triangleq \frac{d}{2}\big((c-1)j\ln(1-c) \\ -2c\ln(c) + (1+c)(2-j)(1-\ln d)\big). \tag{3.21}$$

Next, we maximize the scaled upper bound of $\gamma_{j,k}(\alpha,\beta)$ over α by maximizing $v(c,d)$ over c. The resulting function $v(d) \triangleq \max_{c\in(0,1)} v(c,d)$ is a scaled upper bound on $w_{j,k}(\beta)$ as k goes to infinity. Taking the derivative w.r.t. c, setting it to zero, and solving for d gives the unique solution

$$d = (1-c)^{-j/(j-2)}c^{2/(j-2)}. \tag{3.22}$$

Since the second derivative

$$\frac{d}{2}\left(-\frac{2}{c} - \frac{j}{1-c}\right)(k-1)^{-j/(j-2)}$$

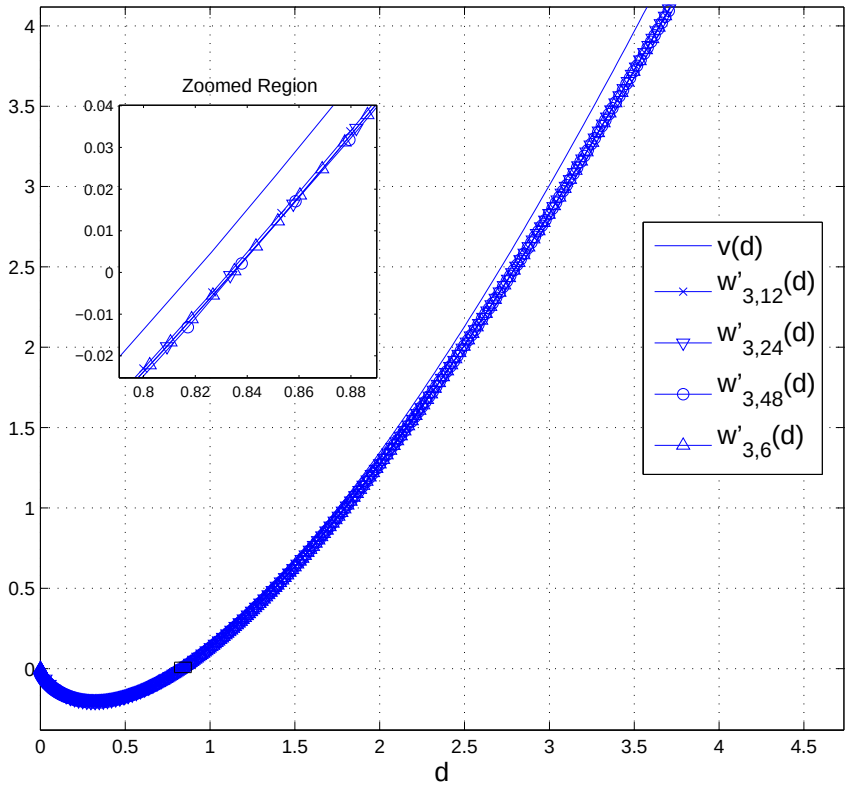

Fig. 7. Numerical evaluation $w'_{j,k}(d)$ and theoretical bound $v(d)$

is negative, so we have found a maximum. Moreover, $v(d)$ is given implicitly by (3.21) and (3.22). The only positive root of $v(d)$ is denoted $\bar{\beta}_j$ and is a constant independent of k. Fig. 7 shows the curves given by numerical evaluation of the scaled $w_{j,k}(\beta)$, which is given by

$$w'_{j,k}(d) = (k-1)^{j/(j-2)} w_{j,k}\left(d/(k-1)^{j/(j-2)}\right),$$

and the limiting function $v(d)$. The proof is not yet complete, however, because we have not yet considered stopping sets whose sizes are sublinear in n. To handle these, we use Lemma 7, which shows that the average number of stopping sets with size

sublinear in n also goes to zero. □

Remark 10. *In a CS system with strictly sparse signals and LM1 reconstruction, we have uniform-in-probability reconstruction (w.h.p. as $n \to \infty$) of all signals with sparsity at most $n\delta$ where $\delta < \bar{\beta}_j (k-1)^{-j/(j-2)}$. This requires $m = \gamma n \delta$ measurements and an oversampling rate of $\gamma > \gamma_0 = \bar{\beta}_j^{-(j-2)/j} j \delta^{-2/j}$.*

Remark 11. *If the signal has all non-negative components [48], then the verification based algorithm will have no FV because the neighbors of a check node will sum to zero only if these neighbors are exactly zero. Therefore, the above analysis implies uniform recovery of non-negative signals that are sufficiently sparse.*

F. Information Theory and Sparse CS

At first glance, the results in this chapter seem to be at odds with existing lower bounds on the number of measurements required for CS. In this section, we explore the fundamental conditions for linear scaling using sparse measurements from an information theoretic point of view.

Let k and j be check and symbol degrees; let n be the number of symbol nodes and $m = n^\omega$ be the number of check symbol nodes. The random signal vector X_1^n has i.i.d. components drawn from $f_X(x)$ and the random measurement vector is Y_1^m. The number of non-zero elements in the signal is controlled by assuming that the average number of non-zero symbol nodes attached to a check node is given by λ. This allows us to write $f_X(x) = \frac{k-\lambda}{k}\delta(x) + \frac{\lambda}{k} f_Z(x)$, where Z is the random variable associated with a non-zero signal element. Since $nj = mk$, the condition $\omega < 1$ implies $k \to \infty$ and that the number of non-zero symbol nodes attached to a check node becomes Poisson with mean λ. Therefore, the amount of information provided by the measurements is given by

$$
\begin{aligned}
H\left(Y_1^m\right) &\leq \sum_{i=1}^{m} H\left(Y_i\right) \\
&= \frac{nj}{k} \sum_{i=0}^{\infty} e^{-\lambda} \frac{\lambda^i}{i!} H(\underbrace{Z * Z * \cdots * Z}_{i \text{ times}}) \\
&\leq jn^{1-\omega} \sum_{i=0}^{\infty} e^{-\lambda} \frac{\lambda^i}{i!} (iH(Z)) \\
&= jn^{1-\omega} \lambda H(Z).
\end{aligned}
$$

Since λ/k is the average fraction of non-zero symbol nodes, the entropy of the signal vector can be written as

$$
\begin{aligned}
H\left(X_1^n\right) &= -nh\left(\frac{\lambda}{k}\right) + n\frac{\lambda}{k} H(Z) \\
&= \lambda n^{1-\omega} \ln \frac{1}{\lambda n^{-\omega}} + \lambda n^{1-\omega} H(Z) + O\left(n^{1-2\omega}\right).
\end{aligned}
$$

This implies that

$$
H(Y_1^m) - H(X_1^n) \leq \lambda n^{1-\omega} \left((j-1)H(Z) - \ n\frac{1}{\lambda n^{-\omega}} \right).
$$

Since a necessary condition for reconstruction is $H\left(Y_1^m\right) - H\left(X_1^n\right) \geq 0$, we therefore find that

$$
n \leq \exp\left(\frac{H(Z)(j-1) + \ln \lambda}{\omega} \right)
$$

is required for reconstruction. This implies, that for any CS algorithm to work, either $H(Z)$ has to be infinite or j has to grow at least logarithmically with n. This does not conflict with the analysis of LM2-MB in randomized reconstruction because, for signals over real numbers or unbounded alphabets, the entropy $H(Z)$ can be infinite.

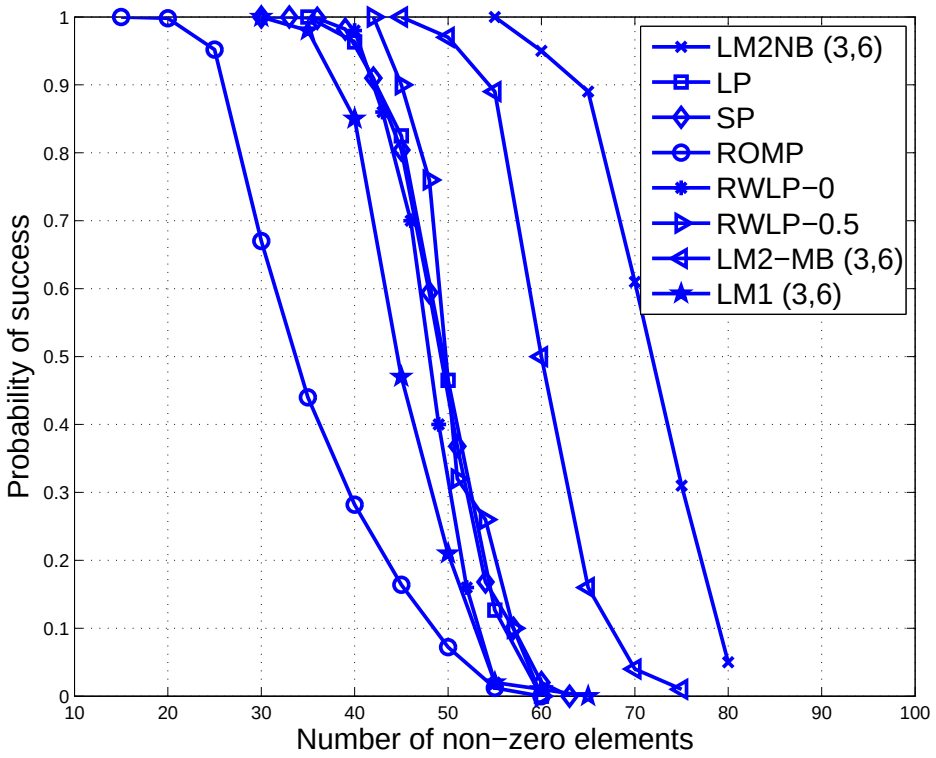

Fig. 8. Simulation results for zero-one sparse signals of length 256 with 128 measurements.

G. Simulation Results

In this section, we provide the simulation results of LM1, LM2-MB and LM2-NB reconstruction algorithms and compare these results with other reconstruction algorithms. We consider two types of strictly sparse signals. The first type is the zero-one sparse signal where the entries of the signal vector are either 0 or ± 1. The second type is the Gaussian sparse case where the entries of the signal are either 0 or a Gaussian random variable with zero mean and unit variance. We choose the signal length $n = 256$ and number of measurements $m = 128$.

We compare different recovery algorithms such as linear-programming (LP) [45],

subspace pursuit (SP) [51], regularized orthogonal matching pursuit (ROMP) [52], reweighted ℓ_p minimization (RWLP-p) [53], LM1, LM2-MB and LM2-NB. The measurement matrices for LM1, LM2-MB and LM2-NB are generated randomly from the $(3, 6)$, (4,8) and (5,10) ensembles without double edges and 4-cycles. We also pick the non-zero entries in the measurement matrices to be i.i.d. Gaussian random variables. In all other algorithms, the measurement matrices are i.i.d. Gaussian random matrices with zero mean and unit variance[3]. Each point is obtained by simulating 100 blocks. Fig. 8 shows the simulation results for the zero-one sparse signal and Fig. 9 shows the results for Gaussian sparse signal. From the results we can see LM2-MB and LM2-NB perform favorably when compared to other algorithms.

Another interesting observation is that LM1, LM2-MB and LM2-NB are not sensitive to the magnitudes of the non-zero coefficients. They perform almost the same for zero-one sparse signal and Gaussian sparse signal. This is due to the verification-based nature of the decoding algorithm. The other advantage of LM1, LM2-MB and LM2-NB is the computational complexity is linear for both the measuring process (i.e., encoding) and the reconstruction process (i.e., decoding).

We also find the maximum sparsity K^* for perfect reconstruction when we use parity-check matrices from $(3, k)$ and $(5, k)$ ensembles (with different k) as the measurement matrices when n is large and try to see how K^* scales with the code rate. In the simulation, we fix $n = 10000$, try different k's (or m's) and use LM2-MB as the decoding algorithm. Fig. 10 shows the how K^* scales with m in high rate regime. We also show the theoretical scaling in Fig. 10 which is $\bar{\alpha}_j nj/k$ with $\bar{\alpha}_3 = 1/6$ and $\bar{\alpha}_5 \approx 0.37$. Notice that the simulation and the theoretical results match very well.

[3]We also tried the other algorithms with our sparse measurement matrices (for the sake of fairness), but the performance was worse than the dense Gaussian random matrices.

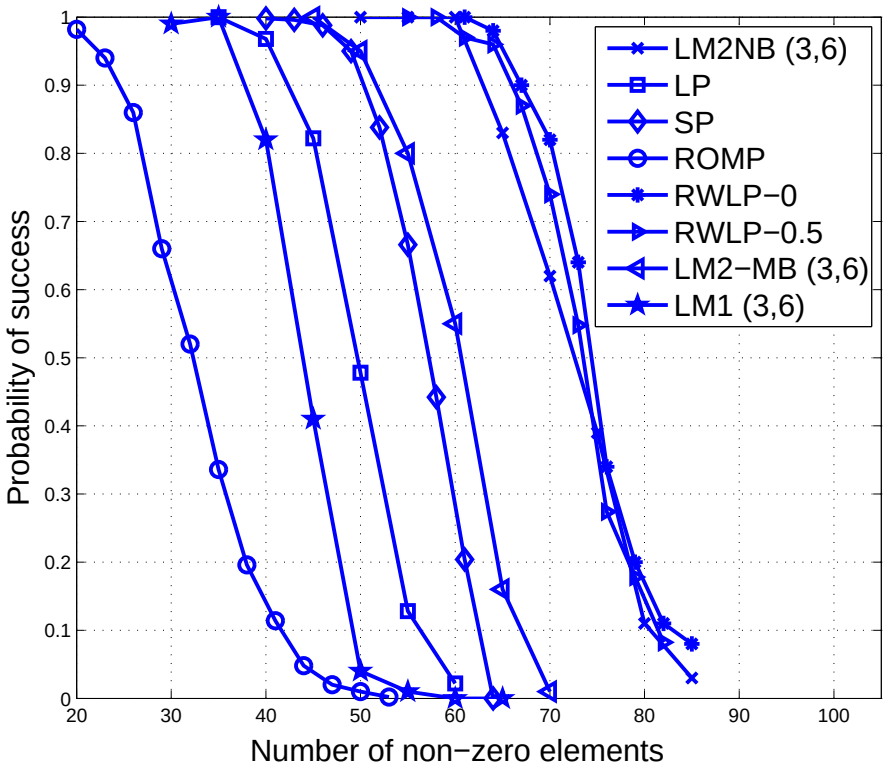

Fig. 9. Simulation results for Gaussian sparse signals of length 256 with 128 measurements.

The simulation results for $(3, 6)$, $(4, 8)$ and $(5, 10)$ ensembles are shown in Fig. 11 and Fig. 12. The results show that for short block length and rate a half, using measurement matrix from ensemble with higher VN/CN degree leads to worse performance. This seems to conflict the results shown in Fig. 10. Actually, in the scaling law analysis, we consider rates close to 1 and large block-length which is not satisfied in the simulation of Fig. 11 and Fig. 12.

In this chapter, we see how LDPC codes over large alphabet sets together with VB decoding algorithms can be applied to a compressed sensing system. In next chapter, we will discuss the modulatoin codes design for flash memories.

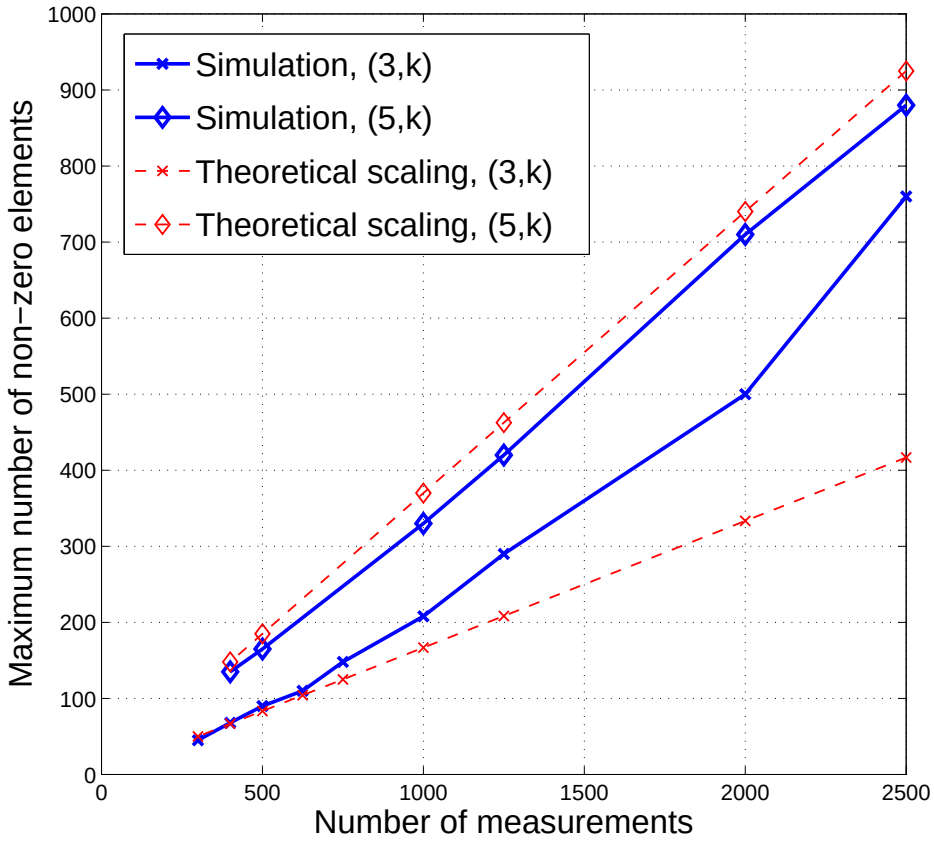

Fig. 10. Simulation of high rate scaling of $(3, k)$ and $(5, k)$ ensembles for block length $n = 10,000$.

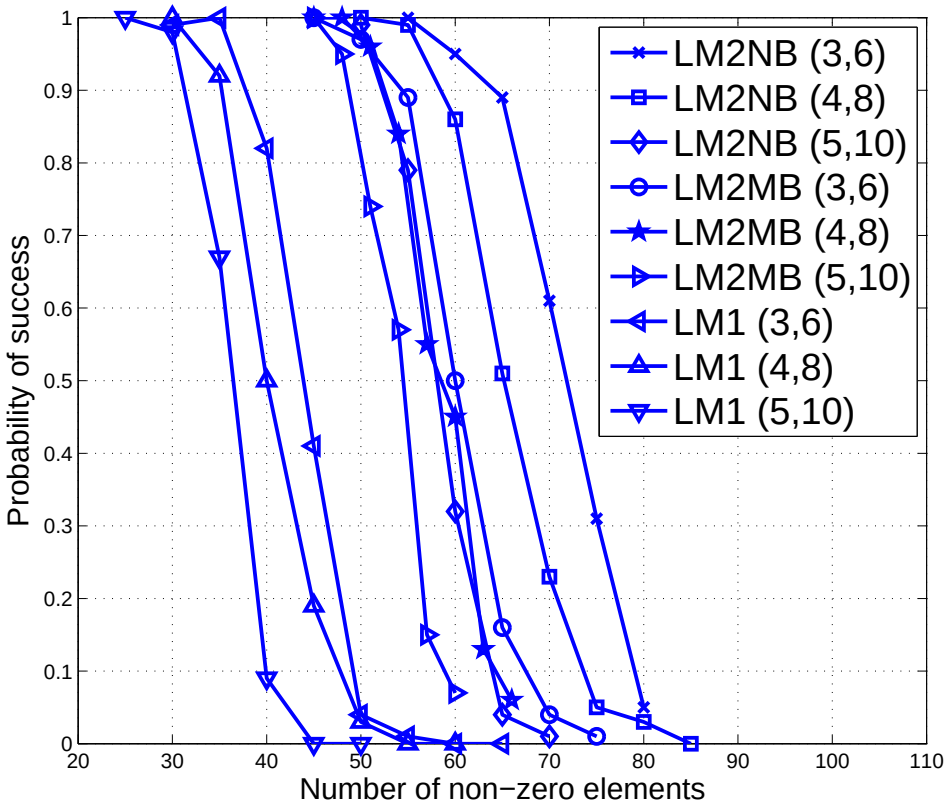

Fig. 11. Simulation results for zero-one spikes of length 256 with 128 measurements by using $(3, 6)$, $(4, 8)$ and $(5, 10)$ ensembles.

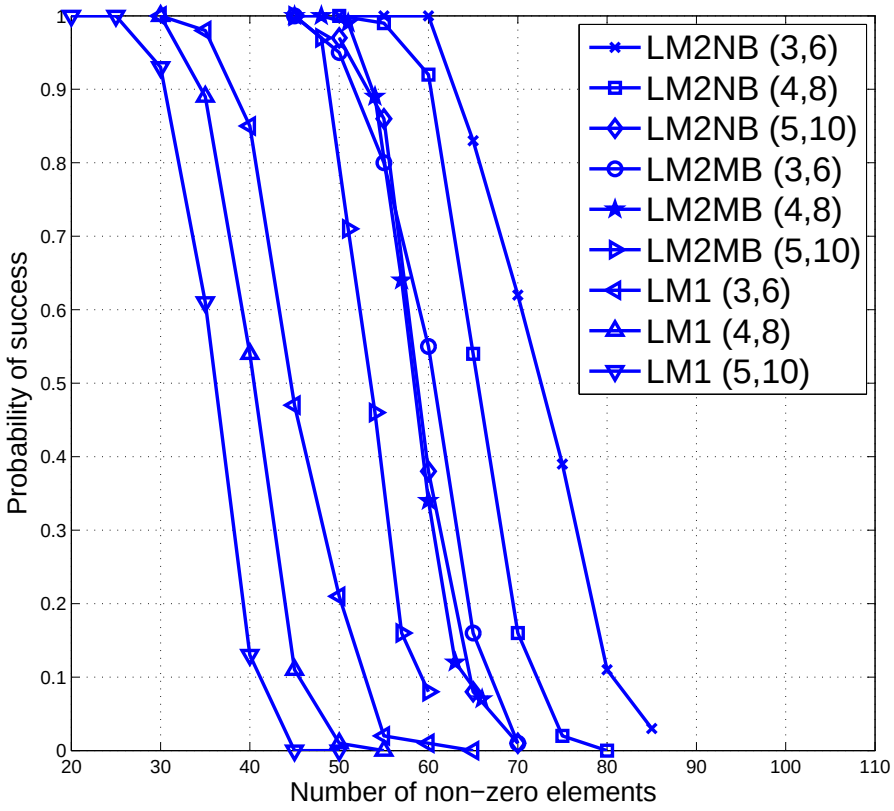

Fig. 12. Simulation results for Gaussian spikes of length 256 with 128 measurements by using $(3, 6)$, $(4, 8)$ and $(5, 10)$ ensembles.

CHAPTER IV

MODULATION CODES FOR FLASH MEMORIES BASED ON LOAD-BALANCING THEORY

A. Introduction

Information-theoretic research on capacity and coding for write-limited memory originates in [57], [56], [58] and [59]. In [57], the authors consider a model of write-once memory (WOM). In particular, each memory cell can be in state either 0 or 1. The state of a cell can go from 0 to 1, but not from 1 back to 0 later. These write-once bits are called *wits*. It is shown that, the efficiency of storing information in a WOM can be improved if one allows multiple rewrites and designs the storage/rewrite scheme carefully.

Multilevel flash memory is a storage technology where the charge level of any cell can be easily increased, but is difficult to decrease. Recent multilevel cell technology allows many charge levels to be stored in a cell. Cells are organized into blocks that contain roughly 10^5 cells. The only way to decrease the charge level of a cell is to erase the whole block (i.e., set the charge on all cells to zero) and reprogram each cell. This takes time, consumes energy, and reduces the lifetime of the memory. Therefore, it is important to design efficient rewriting schemes that maximize the number of rewrites between two erasures [74], [75], [60], [78], [73]. The rewriting schemes increase some cell charge levels based on the current cell state and message to be stored. In this chapter, we call a rewriting scheme a *modulation code*.

Two different objective functions for modulation codes are primarily considered in previous work: (i) maximizing the number of rewrites for the worst case [74, 75, 78] and (ii) maximizing for the average case [73]. As Finucane et al. [73] mentioned, the

reason for considering average performance is the averaging effect caused by the large number of erasures during the lifetime of a flash memory device. Our analysis shows that the worst-case objective and the average case objective are two extreme cases of our optimization objective. We also discuss under what conditions each optimality measure makes sense.

In previous work (e.g., [73, 75, 60, 78]), many modulation codes are shown to be asymptotically optimal as the number of cell-levels q goes to infinity. But the condition that $q \to \infty$ can not be satisfied in practical systems. Therefore, we also analyze asymptotically optimal modulation codes when q is only moderately large using the results from load-balancing theory [72, 76, 61]. This suggests an enhanced algorithm that improves the performance of practical system significantly. Theoretical analysis and simulation results show that this algorithm performs better than other asymptotically optimal algorithms when q is moderately large.

The structure of the chapter is as follows. The system model and performance evaluation metrics are discussed in Section B. An asymptotically optimal modulation code, which is universal over arbitrary i.i.d. input distributions, is proposed in Section C. The storage efficiency of this asymptotically optimal modulation code is analyzed in Section D. An enhanced modulation code is also presented in Section D. The storage efficiency of the enhanced algorithm is also analyzed in Section D. Simulation results and comparisons are presented in Section D.

B. System Model

1. System Description

Flash memory devices usually rely on error detecting/correcting codes to ensure a low error rate. So far, practical systems tend to use Bose-Chaudhuri-Hocquenghem

(BCH) and Reed-Solomon (RS) codes. The error-correcting codes (ECC's) are used as the outer codes while the modulation codes are the inner codes. In this chapter, we focus on the modulation codes and ignore the noise and the design of ECC for now.

Let us assume that a block contains $n \times N$ q-level cells and that n cells (called an n-cell) are used together to store k l-ary variables (called a k-variable). A block contains N n-cells and the N k-variables are assumed to be i.i.d. random variables. We assume that all the k-variables are updated together randomly at the same time and the new values are stored in the corresponding n-cells. This is a reasonable assumption in a system with an outer ECC. We use the subscript t to denote the time index and each rewrite increases t by 1. When we discuss a modulation code, we focus on a single n-cell. (The encoder of the modulation code increases some of the cell-levels based on the current cell-levels and the new value of the k-variable.) Remember that cell-levels can only be increased during a rewrite. So, when any cell-level must be increased beyond the maximum value $q-1$, the whole block is erased and all the cell levels are reset to zero. We let the maximal allowable number of block-erasures be M and assume that after M block erasures, the device becomes unreliable.

Assume the k-variable written at time t is a random variable x_t sampled from the set $\{0, 1, \cdots, l^k - 1\}$ with distribution $p_X(x)$. For convenience, we also represent the k-variable at time t in the vector form as $\bar{x}_t \in \mathbb{Z}_l^k$ where $\mathbb{Z}_l$ denotes the set of integers modulo l. The cell-state vector at time t is denoted as $\bar{s}_t = (s_t(0), s_t(1), \ldots, s_t(n-1))$ and $s_t(i) \in \mathbb{Z}_q$ denotes the charge level of the i-th cell at time t. When we say $\bar{s}_i \succeq \bar{s}_j$, we mean $s_i(m) \geq s_j(m)$ for $m = 0, 1, , \ldots, n-1$. Since the charge level of a cell can only be increased, continuous use of the memory implies that an erasure of the whole block will be required at some point. Although writes, reads and erasures can all

introduce noise into the memory, we neglect this and assume that the writes, reads and erasures are noise-free.

Consider writing information to a flash memory when encoder knows the previous cell state $\bar{s}_{t-1}$, the current k-variable $\bar{x}_t$, and an encoding function $f : \mathbb{Z}_l^k \times \mathbb{Z}_q^n \to \mathbb{Z}_q^n$ that maps $\bar{x}_t$ and $\bar{s}_{t-1}$ to a new cell-state vector $\bar{s}_t$. The decoder only knows the current cell state $\bar{s}_t$ and the decoding function $g : \mathbb{Z}_q^n \to \mathbb{Z}_l^k$ that maps the cell state $\bar{s}_t$ back to the variable vector $\bar{\hat{x}}_t$. Of course, the encoding and decoding functions could change over time to improve performance, but we only consider time-invariant encoding/decoding functions for simplicity.

2. Performance Metrics

a. Lifetime v.s. Storage Efficiency

The idea of designing efficient modulation codes jointly to store multiple variables in multiple cells was introduced by Jiang [74]. In previous work on modulation codes design for flash memory (e.g. [74], [75], [78], [73]), the lifetime of the memory (either worst-case or average) is maximized given fixed amount of information per rewrite. Improving storage density and extending the lifetime of the device are two conflicting objectives. One can either fix one and optimize the other or optimize over these two jointly. Most previous work (e.g., [73, 75, 60, 78]) takes the first approach by fixing the amount of information for each rewrite and maximizing the number of rewrites between two erasures. In this chapter, we consider the latter approach and our objective is to maximize the total amount of information stored in the device until the device dies. This is equivalent to maximizing the average (over the k-variable

distribution $p_X(x)$) amount of information stored per cell-level,

$$\gamma \triangleq E\left(\frac{\sum_{i=1}^{R} I_i}{n(q-1)}\right),$$

where I_i is the amount of information stored at the i-th rewrite, R is the number of rewrites between two erasures, and the expectation is over the k-variable distribution. We also call γ as *storage efficiency*.

b. Worst Case v.s. Average Case

In previous work on modulation codes for flash memory, the number of rewrites of an n-cell has been maximized in two different ways. The authors in [74, 75, 78] consider the worst case number of rewrites and the authors in [73] consider the average number of rewrites. As mentioned in [73], the reason for considering the average case is due to the large number of erasures in the lifetime of a flash memory device. Interestingly, these two considerations can be seen as two extreme cases of the optimization objective in (4.4).

Let the k-variables be a sequence of i.i.d. random variables over time and all the n-cells. The objective of optimization is to maximize the amount of information stored until the device dies. The total amount of information stored in the device[1] can be upper-bounded by

$$W = \sum_{i=1}^{M} R_i \log_2(l^k) \tag{4.1}$$

where R_i is the number of rewrites between the $(i-1)$-th and the i-th erasures. Note that the upper bound in (4.1) is achievable by uniform input distribution, i.e., when

[1]There is a subtlety here. If the n-cell changes to the same value, should it count as stored information? Should this count as a rewrite? This formula assumes that it counts as a rewrite, so that l^k values (rather than $l^k - 1$) can be stored during each rewrite.

the input k-variable is uniformly distributed over $\mathbb{Z}_{l^k}$, each rewrite stores $\log_2(l^k) = k\log_2 l$ bits of information. Due to the i.i.d. property of the input variables over time, R_i's are i.i.d. random variables over time. Since R_i's are i.i.d. over time, we can drop the subscript i. Since M, which is the maximum number of erasures allowed, is approximately on the order of 10^7, by the law of large numbers (LLN), we have

$$W \approx ME[R]\, k\log_2(l).$$

Let the set of all valid encoder/decoder pairs be

$$\mathcal{Q} = \{f, g | \bar{s}_t = f(\bar{s}_{t-1}, \bar{x}_t), \bar{x}_t = g(\bar{s}_t), \bar{s}_t \succeq \bar{s}_{t-1}\},$$

where $\bar{s}_t \succeq \bar{s}_{t-1}$ implies the charge levels are element-wise non-decreasing. This allows us to treat the problem

$$\max_{f,g\in\mathcal{Q}} W,$$

as the following equivalent problem

$$\max_{f,g\in\mathcal{Q}} E[R]\, k\log_2(l). \tag{4.2}$$

Denote the maximal charge level of the i-th n-cell at time t as $d_i(t)$. Note that time index t is reset to zero when a block erasure occurs and increased by one at each rewrite otherwise. Denote the maximal charge level in a block at time t as $d(t)$, which can be calculated as $d(t) = \max_i d_i(t)$. Define t_i as the time when the i-th n-cell reaches its maximal allowed value, i.e., $t_i \triangleq \min\{t | d_i(t) = q\}$. We assume, perhaps naively, that a block-erasure is required when any cell within a block reaches its maximum allowed value. The time when a block erasure is required is defined as $T \triangleq \min_i t_i$. It is easy to see that $E[R] = NE[T]$, where the expectations are over the k-variable distribution. So maximizing $E[T]$ is equivalent to maximizing W. So

the optimization problem (4.2) can be written as the following optimization problem

$$\max_{f,g\in\mathcal{Q}} E\left[\min_{i\in\{1,2,\cdots,N\}} t_i\right]. \tag{4.3}$$

Under the assumption that the input is i.i.d. over all the n-cells and time indices, one finds that the t_i's are i.i.d. random variables. Let their common probability density function (pdf) be $f_t(x)$. It is easy to see that T is the minimum of N i.i.d. random variables with pdf $f_t(x)$. Therefore, we have $f_T(x) = Nf_t(x)\left(1-F_t(x)\right)^{N-1}$, where $F_t(x)$ is the cumulative distribution function (cdf) of t_i. So, the optimization problem (4.3) becomes

$$\max_{f,g\in\mathcal{Q}} E\left[T\right] = \max_{f,g\in\mathcal{Q}} \int Nf_t(x)\left(1-F_t(x)\right)^{N-1} x\mathrm{d}x. \tag{4.4}$$

Note that when $N=1$, the optimization problem in (4.4) simplifies to

$$\max_{f,g\in\mathcal{Q}} E\left[t_i\right]. \tag{4.5}$$

This is essentially the case that the authors in [73] consider. When the whole block is used as one n-cell and the number of erasures allowed is large, optimizing the average (over all input sequences) number of rewrites of an n-cell is equivalent to maximizing the total amount of information stored W. The analysis also shows that the reason we consider average performance is not only due to the averaging effect caused by the large number of erasures. One other important assumption is that there is only one n-cell per block.

The other extreme is when $N \gg 1$. In this case, the pdf $Nf_t(x)\left(1-F_t(x)\right)^{N-1}$ tends to a point mass at the minimum of t and the integral $\int Nf_t(x)\left(1-F_t(x)\right)^{N-1} t\mathrm{d}t$ approaches the minimum of t. This gives the worst case stopping time for the programming process of an n-cell. This case is considered by [74, 75, 78]. Our analysis shows that we should consider the worst case when $N \gg 1$ even though the device

experiences a large number of erasures. So the optimality measure is not determined only by M, but also by N. When N and M are large, it makes more sense to consider the worst case performance. When $N = 1$, it is better to consider the average performance. When N is moderately large, we should maximize the number of rewrites using (4.4) which balances the worst case and the average case.

When N is moderately large, one should probably focus on optimizing the function in (4.4), but it is not clear how to do this directly. So, this remains an open problem for future research. Instead, we will consider a load-balancing approach to improve practical systems where q is moderately large.

3. $N = 1$ v.s. $N \gg 1$

If we assume that there is only one variable changed each time, the average amount of information per cell-level can be bounded by $\log_2 kl$ because there are kl possible new values. Since the number of rewrites can be bounded by $n(q-1)$, we have

$$\gamma \leq \log_2 kl. \tag{4.6}$$

If we allow arbitrary change on the k-variables, there are totally l^k possible new values. It can be shown that

$$\gamma \leq k \log_2 l. \tag{4.7}$$

For fixed l and q, the bound in (4.7) suggests using a large k can improve the storage efficiency. This is also the reason jointly coding over multiple cells can improve the storage efficiency [74]. Since optimal rewriting schemes only allow a single cell-level to increase by one during each rewrite, decodability implies that $n \geq kl - 1$ for the first case and $n \geq l^k - 1$ for the second case. Therefore, the bounds in (4.6) and (4.7) also require large n to improve storage efficiency.

The upper bound in (4.7) grows linearly with k while the upper bound in (4.6) grows logarithmically with k. Therefore, in the remainder of this chapter, we assume an arbitrary change in the k-variable per rewrite and $N = 1$, i.e., the whole block is used as an n-cell, to improve the storage efficiency. This approach implicitly trades instantaneous capacity for future storage capacity because more cells are used to store the same number of bits, but the cells can also be reused many more times.

Note that the assumption of $N = 1$ might be difficult for real implementation, but its analysis gives an upper bound on the storage efficiency. From the analysis above with $N = 1$, we also know that maximizing γ is equivalent to maximize the average number of rewrites.

C. Self-randomized Modulation Codes

In [73], modulation codes are proposed that are asymptotically optimal (as q goes to infinity) in the average sense when $k = 2$. In this section, we introduce a modulation code that is asymptotically optimal for arbitrary input distributions and arbitrary k and l. This rewriting algorithm can be seen as an extension of the one in [73]. The goal is, to increase the cell-levels uniformly on average for an arbitrary input distribution. Of course, decodability must be maintained. The solution is to use common information, known to both the encoder (to encode the input value) and the decoder (to ensure the decodability), to randomize the cell index over time for each particular input value.

Let us assume the k-variable is an i.i.d. random variable over time with arbitrary distribution $p_X(x)$ and the k-variable at time t is denoted as $x_t \in \mathbb{Z}_{l^k}$. The output of the decoder is denoted as $\hat{x}_t \in \mathbb{Z}_{l^k}$. We choose $n = l^k$ and let the cell state vector at time t be $\bar{s}_t = (s_t(0), s_t(1), \cdots, s_t(n-1))$, where $s_t(i) \in \mathbb{Z}_q$ is the charge level of the

i-th cell at time t. At $t = 0$, the variables are initialized to $\overline{s}_0 = (0, \ldots, 0)$, $x_0 = 0$ and $r_0 = 0$.

The decoding algorithm $\hat{x}_t = g(\bar{s}_t)$ is described as follows.

- Step 1: Read cell state vector $\bar{s}_t$ and calculate the ℓ_1 norm $r_t = \|\bar{s}_t\|_1$.
- Step 2: Calculate $s_t = \sum_{i=1}^{n-1} i s_t(i)$ and $\hat{x}_t = s_t - \frac{r_t(r_t+1)}{2} \bmod l^k$.

The encoding algorithm $\overline{s}_t = f(\overline{s}_{t-1}, x_t)$ is described as follows.

- Step 1: Read cell state $\bar{s}_{t-1}$ and calculate r_{t-1} and $\hat{x}_{t-1}$ as above. If $\hat{x}_{t-1} = x_t$, then do nothing.
- Step 2: Calculate $\Delta x_t = x_t - \hat{x}_{t-1} \bmod l^k$ and $w_t = \Delta x_t + r_{t-1} + 1 \bmod l^k$
- Step 3: Increase the charge level of the w_t-th cell by 1.

For convenience, in the rest of the chapter, we refer the above rewriting algorithm as "self-randomized modulation code".

Theorem 10. *The self-randomized modulation code achieves at least $n(q - q^{2/3})$ rewrites with high probability, as $q \to \infty$, for arbitrary k, l, and i.i.d. input distribution $p_X(x)$. Therefore, it is asymptotically optimal for random inputs as $q \to \infty$.*

Sketch of Proof. The proof is similar to the proof in [73]. Since exactly one cell has its level increased by 1 during each rewrite, r_t is an integer sequence that increases by 1 at each rewrite. The cell index to be written w_t is randomized by adding the value $(r_t + 1) \bmod l^k$. This causes each consecutive sequence of l^k rewrites to have a uniform affect on all cell levels. As $q \to \infty$, an unbounded number of rewrites is possible and we can assume $t \to \infty$.

Consider the first $nq - nq^{2/3}$ steps, the value $a_{t,k,l} \triangleq (r_t + 1) \bmod l^k$ is as even as possible over $\{0, 1, \cdots, l^k - 1\}$. For convenience, we say there are $(q - q^{2/3})$ $a_{t,k,l}$'s at

each value, as the rounding difference by 1 is absorbed in the $o(q)$ term. Assuming the input distribution is $p_X = \{p_0, p_1, \cdots, p_{l^k-1}\}$. For the case that $a_{t,k,l} = i$, the probability that $w_t = j$ is $p_{(j-i) \bmod l^k}$ for $j \in \{0, 1, \cdots, l^k - 1\}$. Therefore, w_j has a uniform distribution over $\{0, 1, \cdots, l^k - 1\}$. Since inputs are independent over time, by applying the same Chernoff bound argument as [73], it follows that the number of times $w_t = j$ is at most $q - 3$ with high probability (larger than $1 - 1/\text{poly}(q)$) for all j. Summing over j, we finish the proof. □

Remark 12. *Notice that the randomizing term r_t a deterministic term which makes w_t look* random *over time in the sense that there are equally many terms for each value. Moreover, r_t is known to both the encoder and the decoder such that the encoder can generate "uniform" cell indices over time and the decoder knows the accumulated value of r_t, it can subtract it out and recover the data correctly. Although this algorithm is asymptotically optimal as $q \to \infty$, the maximum number of rewrites $n(q - o(q))$ cannot be achieved for moderate q. This motivates the analysis and the design of an enhanced version of this algorithm for practical systems in next section.*

Remark 13. *A self-randomized modulation code uses $n = l^k$ cells to store a k-variable. This is much larger than the $n = kl$ used by previous asymptotically optimal algorithms because we allow the k-variable to change arbitrarily. Although this seems to be a waste of cells, the average amount of information stored per cell-level is actually maximized (see (4.6) and (4.7)). In fact, the definition of asymptotic optimality requires $n \geq l^k - 1$ if we allow arbitrary changes to the k-variable.*

Remark 14. *We note that the optimality of the self-randomized modulation codes is similar to the weak robust codes presented in [62].*

Remark 15. *We use $n = l^k$ cells to store one of $l^k - 1$ possible messages. This is slightly worse than the simple method of using $n = l^k - 1$. Is it possible to have*

self-randomization using only $n = l^k - 1$ cells? A preliminary analysis of this question based on group theory indicates that it is not. Thus, the extra cell provides the possibility to randomize the mappings between message values and the cell indices over time.

D. Load-balancing Modulation Codes

While asymptotically optimal modulation codes (e.g., codes in [74], [75], [78], [73] and the self-randomized modulation codes described in Section C) require $q \to \infty$, practical systems use q values between 2 and 256. Compared to the number of cells n, the size of q is not quite large enough for asymptotic optimality to suffice. In other words, codes that are asymptotically optimal may have significantly suboptimal performance when the system parameters are not large enough. Moreover, different asymptotically optimal codes may perform differently when q is not large enough. Therefore, asymptotic optimality can be misleading in this case. In this section, we first analyze the storage efficiency of self-randomized modulation codes when q is not large enough and then propose an enhanced algorithm which improves the storage efficiency significantly.

1. Analysis for Moderately Large q

Before we analyze the storage efficiency of asymptotically optimal modulation codes for moderately large q, we first show the connection between rewriting process and the load-balancing problem (aka the balls-into-bins or balls-and-bins problem) which is well studied in mathematics and computer science [72, 76, 61]. Basically, the load-balancing problem considers how to distribute objects among a set of locations as evenly as possible. Specifically, the balls-and-bins model considers the following

problem. If m balls are thrown into n bins, with each ball being placed into a bin chosen independently and uniformly at random, define the *load* as the number of balls in a bin, what is the maximal load over all the bins? Based on the results in Theorem 1 in [61], we take a simpler and less accurate approach to the balls-into-bins problem and arrive at the following theorem.

Theorem 11. *Suppose that m balls are sequentially placed into n bins. Each time a bin is chosen independently and uniformly at random. The maximal load over all the bins is L and:*

(i) If $m = d_1 n$, the maximally loaded bin has $L \leq \frac{c_1 \ln n}{\ln \ln n}$ balls, $c_1 > 2$ and $d_1 \geq 1$, with high probability $(1 - 1/poly(n))$ as $n \to \infty$.

(ii) If $m = n \ln n$, the maximally loaded bin has $L \leq \frac{c_4 (\ln n)^2}{\ln \ln n}$ balls, $c_4 > 1$, with high probability $(1 - 1/poly(n))$ as $n \to \infty$.

(iii) If $m = c_3 n^{d_2}$, the maximally loaded bin has $L \leq e c_3 n^{d_2 - 1} + c_2 \ln n$, $c_2 > 1$, $c_3 \geq 1$ and $d_2 > 1$, with high probability $(1 - 1/poly(n))$ as $n \to \infty$.

Proof. Denote the event that there are at least k balls in a particular bin as E_k. Using the union bound over all subsets of size k, it is easy to show that the probability that E_k occurs is upper bounded by

$$\Pr\{E_k\} \leq \binom{m}{k} \left(\frac{1}{n}\right)^k.$$

Using Stirling's formula, we have $\binom{m}{k} \leq \left(\frac{me}{k}\right)^k$. Then $\Pr\{E_k\}$ can be further bounded by

$$\Pr\{E_k\} \leq \left(\frac{me}{nk}\right)^k. \tag{4.8}$$

If $m = d_1 n$, substitute $k = \frac{c_1 \ln n}{\ln \ln n}$ to the RHS of (4.8), we have

$$\begin{aligned}
\Pr\{E_k\} &\leq \left(\frac{d_1 e \ln \ln n}{c_1 \ln n}\right)^{\frac{c_1 \ln n}{\ln \ln n}} \\
&= e^{\left(\frac{c_1 \ln n}{\ln \ln n}(\ln(d_1 e \ln \ln n) - \ln(c_1 \ln n))\right)} \\
&< e^{\left(\frac{c_1 \ln n}{\ln \ln n}(\ln(d_1 e \ln \ln n) - \ln \ln n)\right)} \\
&\leq e^{(-(c_1-1)\ln n)} = \frac{1}{n^{c_1-1}}.
\end{aligned}$$

Denote the event that all bins have at most k balls as $\tilde{E}_k$. By applying the union bound, it is shown that

$$\Pr\{\tilde{E}_k\} \geq 1 - \frac{n}{n^{c_1-1}} = 1 - \frac{1}{n^{c_1-2}}.$$

Since $c_1 > 2$, we finish the proof for the case of $m = d_1 n$.

If $m = n \ln n$, substitute $k = \frac{c_4 (\ln n)^2}{\ln \ln n}$ to the RHS of (4.8), we have

$$\begin{aligned}
\Pr\{E_k\} &\leq \left(\frac{e \ln \ln n}{c_4 \ln n}\right)^{\frac{c_4 (\ln n)^2}{\ln \ln n}} \\
&= e^{\left(\frac{c_4 (\ln n)^2}{\ln \ln n}(\ln e \ln \ln n - \ln c_4 \ln n)\right)} \\
&\leq e^{\left(\frac{c_4 (\ln n)^2}{\ln \ln n}(\ln e \ln \ln n - \ln \ln n)\right)} \\
&\leq e^{\left(-(c_4-1)(\ln n)^2\right)} = o\left(\frac{1}{n^2}\right).
\end{aligned}$$

By applying the union bound, we finish the proof for the case of $m = n \ln n$.

If $m = c_3 n^{d_2}$, substitute $k = e c_3 n^{d_2 - 1} + c_2 \ln n = e c_3 n^{d_2 - 1} + c_2 \ln n$ to the RHS of

(4.8), we have

$$\begin{aligned}\Pr\{E_k\} &\leq \left(\frac{ec_3n^{d_2-1}}{ec_3n^{d_2-1}+c_2\ln n}\right)^{c_3en^{d_2-1}+c_2\ln n}\\ &= e^{\left((c_5n^{d_2-1}+c_2\ln n)\left(\ln c_5n^{d_2-1}-\ln(c_5n^{d_2-1}+c_2\ln n)\right)\right)}\\ &\leq e^{\left((c_5n^{d_2-1}+c_2\ln n)\left(-\frac{c_2\ln n}{c_5n^{d_2-1}}\right)\right)}\\ &\leq e^{(-c_2\ln n)} = \frac{1}{n^{c_2}}\end{aligned}$$

where $c_5 = ec_3$. By applying the union bound, it is shown that

$$\Pr\{\tilde{E}_k\} \leq 1-\frac{n}{n^{c_2}} = 1-\frac{1}{n^{c_2-1}}.$$

Since $c_2 > 1$, we finish the proof for the case of $m = c_3n^{d_2}$. □

Remark 16. *Note that Theorem 11 only shows an upper bound on the maximum load L with a simple proof. More precise results can be found in Theorem 1 of [61], where the exact order of L is given for different cases. It is worth mentioning that the results in Theorem 1 of [61] are different from Theorem 11 because Theorem 1 of [61] holds with probability $1-o(1)$ while Theorem 11 holds with probability $(1-1/poly(n))$.*

Remark 17. *The asymptotic optimality in the rewriting process implies that each rewrite only increases the cell-level of a cell by 1 and all the cell-levels are fully used when an erasure occurs. This actually implies $\lim_{m\to\infty}\frac{L}{m/n}=1$. Since n is usually a large number and q is not large enough in practice, the theorem shows that, when q is not large enough, asymptotic optimality is not achievable. For example, in practical systems, the number of cell-levels q does not depend on the number of cells in a block. Therefore, rather than $n(q-1)$, only roughly $n(q-1)\frac{\ln\ln n}{\ln n}$ charge levels can be used as $n\to\infty$ if q is a small constant which is independent of n. In practice, this loss could be mitigated by using writes that increase the charge level in multiple cells*

simultaneously (instead of erasing the block).

Theorem 12. *The self-randomized modulation code has storage efficiency* $\gamma = c \ln \frac{k \ln l}{c}$ *when* $q - 1 = c$ *and* $\gamma = \frac{c}{\theta} k \ln l$ *when* $q - 1 = c \ln n$ *as* n *goes to infinity with high probability (i.e.,* $1 - o(1)$*).*

Proof. Consider the problem of throwing m balls into n bins and let the r.v. M be the number of balls thrown into n bins until some bin has more than $q-1$ balls in it. While we would like to calculate $E[M]$ exactly, we still settle for an approximation based on the following result. If $m = cn \ln n$, then there is a constant $d(c)$ such that maximum number of balls L in any bin satisfies

$$(d(c) - 1) \ln n \leq L \leq d(c) \ln n$$

with probability $1 - o(1)$ as $n \to \infty$ [61] . The constant $d(c)$ is given by the largest x-root of

$$x(\ln c - \ln x + 1) + 1 - c = 0,$$

and solving this equation for c gives the implicit expression $c = -d(c)W\left(-e^{-1-\frac{1}{d(c)}}\right)$. Since the lower bound matches the expected maximum value better, we define $\theta \triangleq d(c) - 1$ and apply it to our problem using the equation $\theta \ln n = q - 1$ or $\theta = \frac{q-1}{\ln n}$. Therefore, the storage efficiency is $\gamma = \frac{m \ln n}{n(q-1)} = \frac{c}{\theta} k \ln l$

If $m = cn$, the maximum load is approximately $\frac{\ln n}{\ln \frac{n \ln n}{m}}$ with probability $1 - o(1)$ for large n [61]. By definition, $q - 1 = \frac{\ln n}{\ln \frac{n \ln n}{m}}$. Therefore, the storage efficiency is $\gamma = \frac{m \ln n}{n(q-1)} = c \ln \frac{\ln n}{c} = c \ln \frac{k \ln l}{c}$. □

Remark 18. *The results in Theorem 12 show that when* q *is on the order of* $O(\ln n)$*, the storage efficiency is on the order of* $\Theta(k \ln l)$*. Taking the limit as* $q, n \to \infty$ *with* $q = O(\ln n)$*, we have* $\lim \frac{\gamma}{k \ln l} = \frac{\theta}{c} > 0$*. When* q *is a constant independent of* n*, the storage efficiency is on the order of* $\Theta(\ln k \ln l)$*. Taking the limit as* $n \to \infty$ *with*

$q-1=c$, *we have* $\lim \frac{\gamma}{k \ln l}=0$. *In this regime, the self-randomized modulation codes actually perform very poorly even though they are asymptotically optimal as* $q \to \infty$.

2. Load-balancing Modulation Codes

Considering the bins-and-balls problem, can we distribute balls more evenly when m/n is on the order of $o(n)$? Fortunately, when $m = n$, the maximal load can be reduced by a factor of roughly $\frac{\ln n}{(\ln \ln n)^2}$ by using *the power of two random choices* [76]. In detail, the strategy is, every time we pick two bins independently and uniformly at random and throw a ball into the less loaded bin. By doing this, the maximally loaded bin has roughly $\frac{\ln \ln n}{\ln 2} + O(1)$ balls with high probability. Theorem 1 in [72] gives the answer in a general form when we consider d random choices. The theorem shows there is a large gain when the number of random choice is increased from 1 to 2. Beyond that, the gain is on the same order and only the constant can be improved.

Based on the idea of 2 random choices, we define the following load-balanced modulation code.

Again, we let the cell state vector at time t be $\bar{s}_t = (s_t(0), s_t(1), \cdots, s_t(n-1))$, where $s_t(i) \in \mathbb{Z}_q$ is the charge level of the i-th cell at time t. This time, we use $n = l^{k+1}$ cells to store a k-variable $x_t \in \mathbb{Z}_{l^k}$ (i.e., we write $(k+1)\log_2 l$ bits to store $k \log_2 l$ bits of information). The information loss provides l ways to write the same value. This flexibility allows us to avoid sequences of writes that increase one cell level too much. We are primarily interested in binary variables with 2 random choices or $l = 2$. For the power of l choices to be effective, we must try to randomize (over time), the l possible choices over the set of all $\binom{n}{l}$ possibilities. The value $r_t = \|\bar{s}_t\|_1$ is used to do this. Let H be the Galois field with l^{k+1} elements and $h : \mathbb{Z}_{l^{k+1}} \to H$ be a bijection that satisfies $h(0) = 0$ (i.e., the Galois field element 0 is associated with the integer 0).

The decoding algorithm calculates $\hat{x}_t$ from $\bar{s}_t$ and operates as follows:

- Step 1: Read cell state vector $\bar{s}_t$ and calculate the ℓ_1 norm $r_t = \|\bar{s}_t\|_1$.
- Step 2: Calculate $s_t = \sum_{i=1}^{n} i s_t(i)$ and $\hat{x}'_t = s_t \bmod l^{k+1}$.
- Step 3: Calculate $a_t = h\left(\left(r_t \bmod l^k - 1\right) + 1\right)$ and $b_t = h\left(r_t \bmod l^k\right)$
- Step 4: Calculate $\hat{x}_t = h^{-1}\left(a_t^{-1}\left(h(\hat{x}'_t) - b_t\right)\right) \bmod l^k$.

The encoding algorithm stores x_t and operates as follows.

- Step 1: Read cell state $\bar{s}_{t-1}$ and decode to $\hat{x}'_{t-1}$ and $\hat{x}_{t-1}$. If $\hat{x}_{t-1} = x_t$, then do nothing.
- Step 2: Calculate $r_t = \|\bar{s}_{t-1}\|_1 + 1$, $a_t = h\left(\left(r_t \bmod l^k - 1\right) + 1\right)$, and $b_t = h\left(r_t \bmod l^k\right)$
- Step 3: Calculate $x_t^{(i)} = h^{-1}\left(a_t h(x_t + i l^k) + b_t\right)$ and $\Delta x_t^{(i)} = x_t^{(i)} - \hat{x}'_{t-1} \bmod l^{k+1}$ for $i = 0, 1, \ldots l-1$.
- Step 4: Calculate[2] $w_t = \arg\min_{j \in \mathbb{Z}_l}\{s_{t-1}(\Delta x_t^{(j)})\}$. Increase the charge level by 1 of cell $\Delta x_t^{(w_t)}$.

Note that the state vector at $t = 0$ is initialized to $s_0 = (0, \ldots, 0)$ and therefore $x_0 = 0$. The first arbitrary value that can be stored is x_1.

The following conjecture suggests that the ball-loading performance of the above algorithm is identical to the random loading algorithm with $l = 2$ random choices.

Conjecture 2. *If $l = 2$ and $q - 1 = c \ln n$, then the load-balancing modulation code has storage efficiency $\gamma = k$ with probability 1-$o(1)$ as $n \to \infty$. If $q - 1 = c$, the storage efficiency $\gamma = \frac{c \ln 2}{\ln \ln n} k$ with probability 1-$o(1)$.*

[2]Ties can be broken arbitrarily.

Sketch of Proof. Consider the affine permutation $\pi_x^{(a,b)} = h^{-1}(ah(x)+b)$ for $a \in H\backslash 0$ and $b \in H$. As a, b vary, this permutation maps the two elements x_t and $x_t + l^k$ uniformly over all pairs of cell indices. After $m = n(n-1)$ steps, we see that all pairs of a, b occur equally often. Therefore, by picking the less charged cell, the modulation code is almost identical to the random loading algorithm with two random choices. Unfortunately, we are interested in the case where $m \ll n^2$ so the analysis is somewhat more delicate. If $m = cn\ln n$, the highest charge level is $c\ln n - 1 + \frac{\ln\ln n}{\ln 2} \approx c\ln n$ with probability $1 - o(1)$ [72]. Since $q - 1 = c\ln n$ in this case, the storage efficiency is $\gamma = \frac{cn\ln n \log_2 2^k}{nc\ln n} = k$. If $m = cn$, then $q - 1 = c$ and the maximum load is $c - 1 + \ln\ln n/\ln 2 \approx \frac{\ln\ln n}{\ln 2}$. By definition, we have $\frac{\ln\ln n}{\ln 2} = q - 1$. Therefore, we have $\gamma = \frac{cn\log_2 2^k}{n(q-1)} = \frac{c\ln 2}{\ln\ln n}k$. □

Remark 19. *If $l = 2$ and q is on the order of $O(\ln n)$, Conjecture 2 shows that the bound (4.7) is achievable by load-balancing modulation codes as n goes to infinity. In this regime, the load-balancing modulation codes provide a better constant than self-randomized modulation codes by using twice many cells.*

Remark 20. *If $l = 2$ and q is a constant independent of n, the storage efficiency is $\gamma_1 = c\ln\frac{k}{c}$ for the self-randomized modulation code and $\gamma_2 = \frac{c\ln 2}{\ln\ln n}\log_2\frac{n}{2}$ for the load-balancing modulation code. But, the self-randomized modulation code uses $n = 2^k$ cells and the load-balancing modulation code uses $n = 2^{k+1}$ cells. To make fair comparison on the storage efficiency between them, we let $n = 2^{k+1}$ for both codes. Then we have $\gamma_1 = c\ln\frac{\log_2 n}{c}$ and $\gamma_2 = \frac{c\ln 2}{\ln\ln n}\log_2\frac{n}{2}$. So, as $n \to \infty$, we see that $\frac{\gamma_1}{\gamma_2} \to 0$. Therefore, the load-balancing modulation code outperforms the self-randomized code when n is sufficiently large.*

E. Simulation Results

In this section, we present the simulation results for the modulation codes described in Sections C and 2. In the figures, the first modulation code is called the "self-randomized modulation code" while the second is called the "load-balancing modulation code". Let the "loss factor" η be the fraction of cell-levels which are not used when a block erasure is required: $\eta \triangleq 1 - \frac{E[R]}{n(q-1)}$. We show the loss factor for random loading with 1 and 2 random choices as comparison. Note that η does not take the amount of information per cell-level into account. Results in Fig. 13 show that the self-randomized modulation code has the same η with random loading with 1 random choice and the load-balancing modulation code has the same η with random loading with 2 random choices. This shows the optimality of these two modulation codes in terms of ball loading.

We also provide the simulation results for random loading with 1 random choice and the codes designed in [73], which we denote as FLM-$(k = 2, l = 2, n = 2)$ algorithm, in Fig. 14. From results shown in Fig. 14, we see that the FLM-$(k = 2, l = 2, n = 2)$ algorithm has the same loss factor as random loading with 1 random choice. This can be actually seen from the proof of asymptotic optimality in [73] as the algorithm transforms an arbitrary input distribution into an uniform distribution on the cell-level increment. Note that FLM algorithm is only proved to be optimal when 1 bit of information is stored. So we just compare the FLM algorithm with random loading algorithm in this case. Fig. 15 and Fig. 16 show the storage efficiency γ for these two modulation codes. Fig. 15 and Fig. 16 show that the load-balancing modulation code performs better than self-randomized modulation code when n is large. This is also shown by the theoretical analysis in Remark 20.

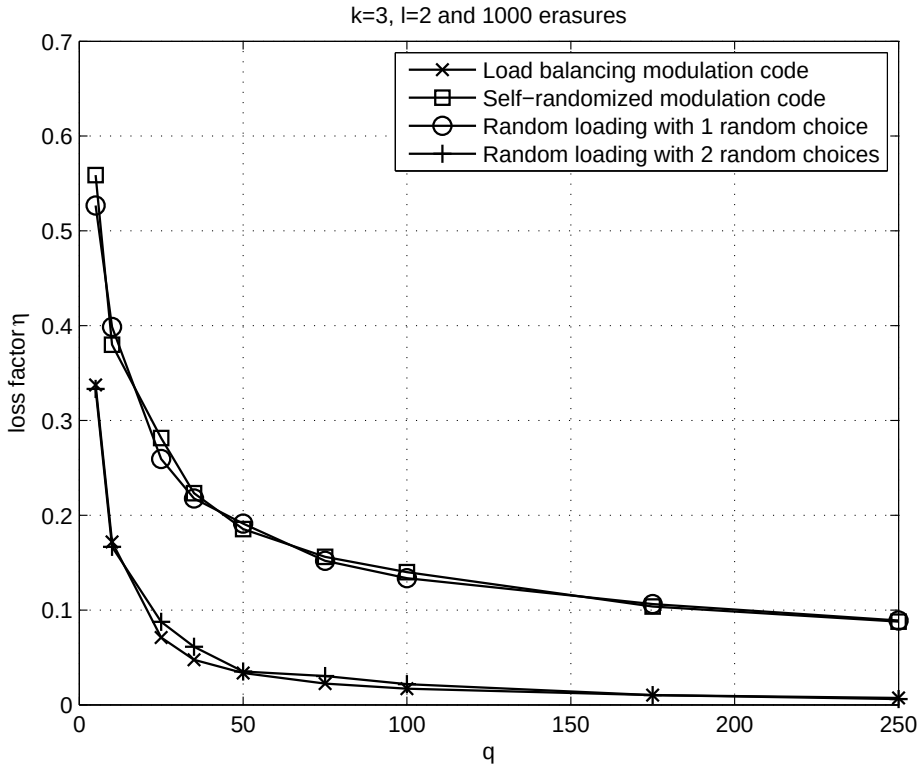

Fig. 13. Simulation results for random loading and algorithms we proposed with $k = 3$, $l = 2$ and 1000 block erasures.

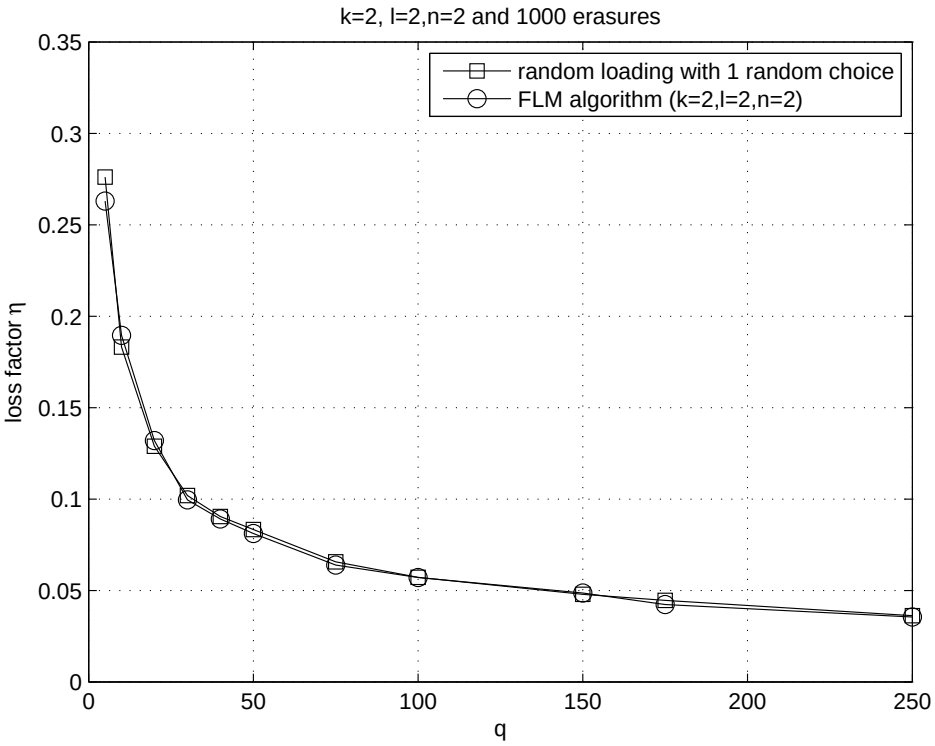

Fig. 14. Simulation results for random loading and codes in [4] with $k = 2$, $l = 2$, $n = 2$ and 1000 block erasures.

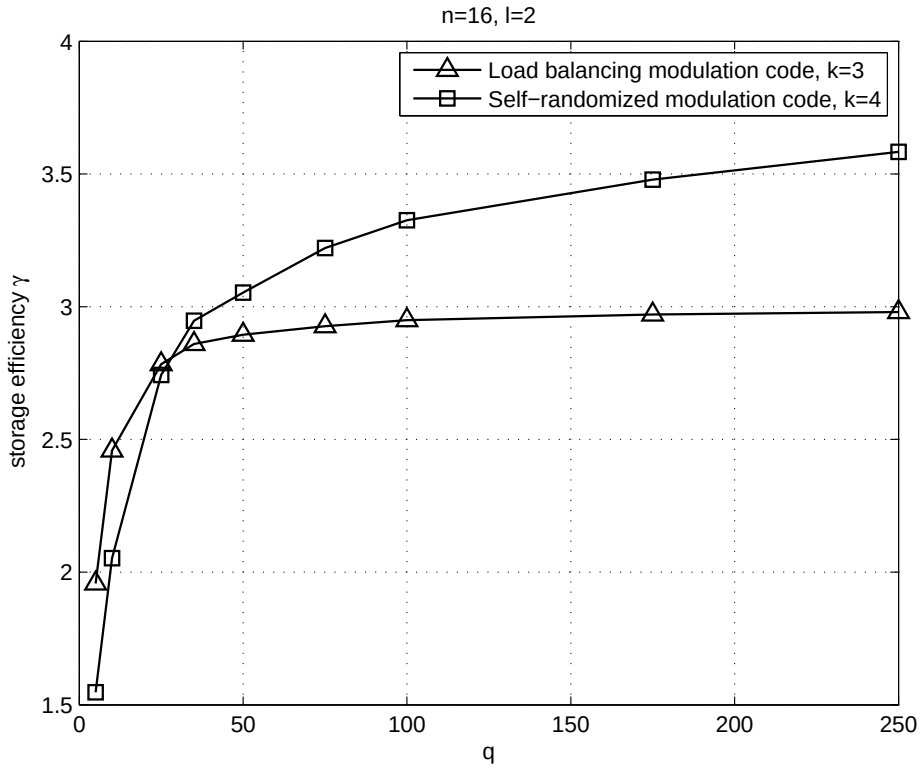

Fig. 15. Storage efficiency of self-randomized modulation code and load-balancing modulation code with $n = 16$.

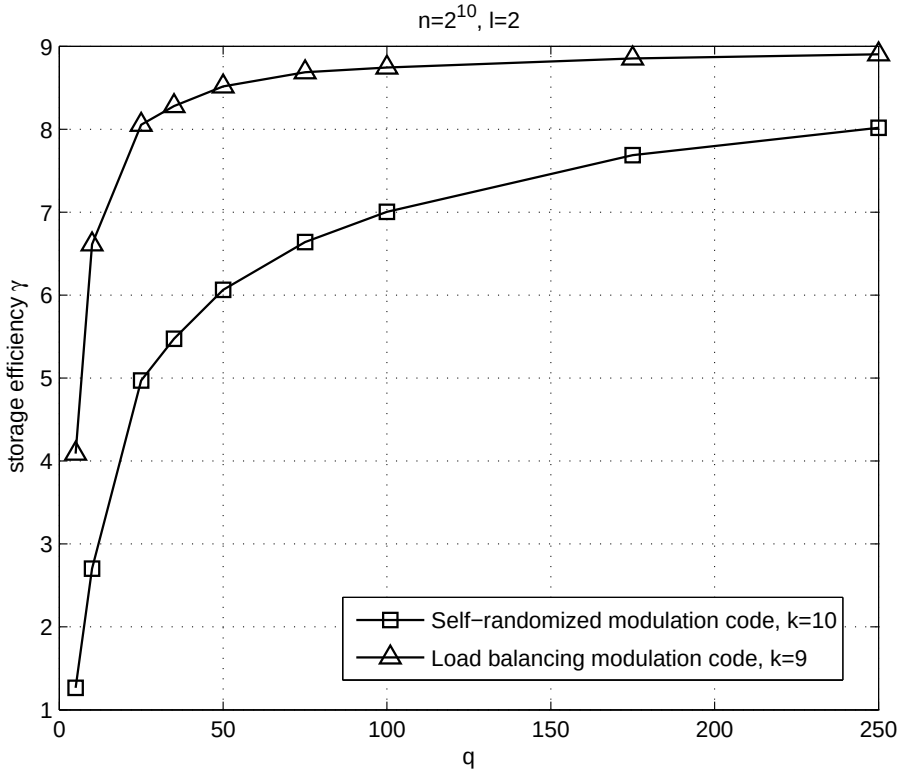

Fig. 16. Storage efficiency of self-randomized modulation code and load-balancing modulation code with $n = 2^{10}$.

F. A Group-Theoretic Approach to Modulation Code Design for Flash Memory

In this section, we propose a group-theoretic approach to modulation code design for flash memory. This approach provides a general mathematical framework for the modulation codes design problem. We show that many previously proposed modulation codes can be described in this framework. The framework also provides the mathematical tools for analyzing and comparing different codes. Some interesting connections between different codes are revealed. Furthermore, by using the idea from load-balancing theory, this framework can be used to find good modulation codes.

Consider a modulation code which represents one of M messages using n cells with q levels, which range from 0 to $q-1$. Suppose also that there are only certain (e.g., one of N_A) allowable changes in the message. For example, think of a message containing k l-ary variables (i.e., $M = l^k$) where only one variable can be changed at a time (i.e., $N_A = k(l-1)$). It will be useful to model the message set as a group and the allowable changes as a subset of that group. The set of modulation codewords also has a natural mapping to the group $\mathbb{Z}_q^n$, where $\mathbb{Z}_q$ is the set of integers modulo q. The following gives the general definition of modulation codes in a group-theoretic framework.

1. General Definition of Modulation Codes

Let $(G, +)$ be an M-element group representing the message set and $A \subseteq G$ be a subset with N_A elements. The set $A = \{a_1, \ldots, a_{N_A}\}$ represents the allowable single-step changes in G. We will assume that the subset A generates G because otherwise, the subset of possible messages will depend on the initial message. Let $(H, +)$ be a group representing the cell state and $B \subseteq H$ be a subset with N_B elements. The set B

represents the allowable single-step changes in H. The encoder $E : H \times A \rightarrow H$ maps the current state and the message change to the new state. The decoder $D : H \rightarrow G$ maps the cell state to the message value. A modulation code is defined by the tuple$\mathcal{C} \triangleq \{G, H, A, B, E, D\}$.

2. A Simple Modulation Code

For the reason of simplicity, we first consider the decoder which is a group homomorphism: $D(s_1 + s_2) = D(s_1) + D(s_2)$ for all s_1,$s_2 \in H$. Since the decoder mapping is a group homomorphism, one can write, for every allowable change b,

$$x_t = D\left(s_t\right) = D\left(s_{t-1} + b\right) = D\left(s_{t-1}\right) + D\left(b\right) = x_{t-1} + D(b).$$

Combining this equation and the set of allowable state changes, we discover that decoding will succeed (i.e., return x_t) iff

$$b \in D^{\dagger}\left(\Delta x_t\right) \triangleq D^{-1}\left(x_t - x_{t-1}\right) \cap B,$$

where $D^{-1}(g) = \{h \in H \mid D(h) = g\}$. If $N_B = N_A$, every valid message change must be mapped to a valid cell state change and $D^{\dagger} : A \rightarrow B$ must be a bijection. Under the assumption of group homomorphism, there is only one encoder/decoder pair up to isomorphism which is uniquely defined. The implied encoder is given by

$$E\left(s_{t-1}, x_t\right) = \begin{cases} s_{t-1} & \text{if } x_t = D\left(s_{t-1}\right) \\ s_{t-1} + D^{\dagger}\left(x_t - D(s_{t-1})\right) & \text{otherwise.} \end{cases}$$

For efficiency, one would like to design modulation codes that maximize the number of rewrites before any cell-level reaches q (or returns to 0 in the $\mathbb{Z}_q$ group). One difficulty is that the input distribution may not be uniform over all allowable

message changes. In this case, one cell level may reach q while other cell levels are still quite small. The main problem with the setting above (i.e., $N_B = N_A$ and the homomorphism property) is that the encoder/decoder pair must have a one-to-one mapping between the allowable message change set A and the allowable cell state change set B. This prevents us from building randomization into the system to mitigate the effect of non-uniform input distributions.

Lemma F.1. *If $N_B = N_A$ and the decoder is a group homomorphism, all messages can be encoded by increasing the charge level of a single cell by one. All valid encoder/decoder pairs are isomorphic to the encoder/decoder pair uniquely defined by*

$$D(s) = \sum_{i=1}^{N_A} s(i)a_i$$

where the multiplication of an integer $s(i)$ by a group element a_i is defined as

$$s(i)a_i \triangleq \underbrace{a_i + a_i + \cdots + a_i}_{s(i) \text{ times}}.$$

We say a modulation code is robust *if the average number of rewrites is $n(q-1) - o(nq)$. This modulation code is not robust (for non-uniform i.i.d. input distributions).*

Proof. Since exactly one cell's charge level can be increased by 1, each element in B is uniquely represented by a cell index. Notice that the valid encoder/decoder must have a one-to-one mapping between B and A. So all the valid encoders are the same up to isomorphism. Since encoding the same change repeatedly will keep increasing the cell-level of a particular cell, this encoder is not robust against general non-uniform i.i.d. input distributions and none of these encoders are robust. □

Remark 21. *It would be interesting to know if there exists an encoder/decoder which can handle arbitrarily i.i.d. input distributions if we are allowed to design the decoder*

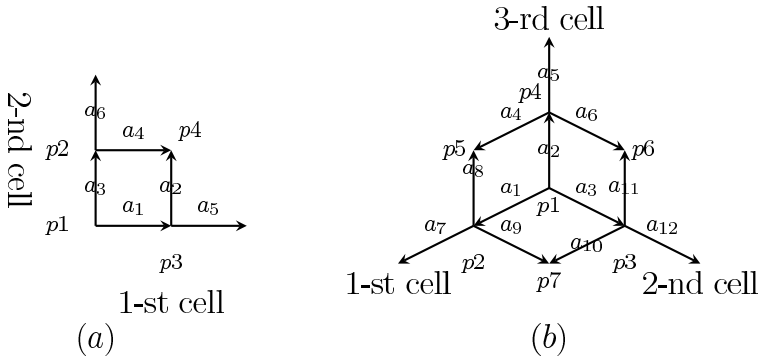

Fig. 17. Encoding grids for $n = 2$ and $n = 3$

which is not a group homomorphism, (e.g., by using different mappings between A and B over time) The following subsection shows that if $N_A = N_B$ and $A = G\backslash\{0\}$, i.e., the input variable is allowed to change arbitrarily, there is no robust encoder/decoder when $n = 2$ and $n = 3$. However, if $A \subset G\backslash\{0\}$ and $N_A = N_B$, there exists a robust encoder/decoder by using different mappings over time when $n = 2$ [73].

3. Modulation Codes via Periodic Tiling

In this subsection, we consider robust modulation code design when $N_A = N_B$ and the decoder is not a group homomorphism. Without loss of generality, we consider the case that we use n cells to store an $(n + 1)$-ary variable, i.e., $H = \mathbb{Z}_q^n$ and $G = \{0, 1, \cdots, n\}$. We only allow one cell's charge level to be increased by 1 at each rewrite, i.e., $B = \{1, 2, \cdots, n\}$. We allow arbitrary change of the message, i.e., the set $A = \{+1, +2, \cdots, +n\}$. Then there is only 1 mapping between A and B up to isomorphism. Actually, there are $n!$ different mappings, but they are the same by relabeling the indices of the cells and allowable message changes.

Before we discuss the encoder/decoder, we note that there may be different Abelian groups given the size of the group. For example, the Klein four-group and the cyclic group of order 4, i.e., $\mathbb{Z}_4$, are two different Abelian groups of size 4. Different

group structures may affect the existence of the robust codes.

Lemma F.2. *When $n = N_A = N_B = 2$, there is only one encoder/decoder pair up to isomorphism and it is not robust.*

Proof. When $n = 2$, there is only one group of size 3 and the addition operation is modulo 3 and we can write $A = \{+1, +2\}$ and $B = \{1, 2\}$. There are two possible mappings between A and B. The encoder can be represented by a 2-dimensional directed grid shown in Figure 17-(a). The two axes represent the cell levels of two cells and the numbers labeled on the arrows represent the change of the input message $a \in A$. The decoder can be defined naturally by the encoding grid. Let's look at an arbitrary point p_1 and its adjacent points p_2, p_3 and p_4 shown in Figure 17-(a). Encodability requires that each point (i.e., cell state) can encode all the elements in A. Since p_4 can be reached in two different ways, the decodability of p_4 requires $a_1 + a_2 = a_3 + a_4$ where the addition is modulo 3 addition. By simply enumerating all the cases, it is easy to verify that the encodability and decodability can only be satisfied by using the same mapping between A and B on p_1, p_2 and p_3. Treating p_1 as the origin and using induction shows that all the points in the encoding grid must use the same mapping to satisfy the encodability and decodability. Therefore, there are only two encoder/decoder pairs and they are the same up to relabeling. □

Lemma F.3. *When $n = N_A = N_B = 3$, there are more than 1 encoder/decoder pairs. But there is no robust code.*

Proof. When $n = 3$, there are two groups of size 4, i.e., the Klein four-group and $\mathbb{Z}_4$. Since we consider arbitrary change of the input variable and one cell-level increased per rewrite, $A = \{+1, +2, +3\}$ and $B = \{1, 2, 3\}$. the encoder can be represented by a 3-dimensional directed grid shown in Figure 17-(b). The three axes represent the cell levels of three cells. Note that there are 6 possible mappings between A and B.

Similarly, the encodability of $p_i, i = 1, \cdots, 4$ requires that each point can encode all the messages in A. The decodability of p_5, p_6 and p_7 requires different encoding paths starting from p_1 to p_5, p_6 and p_7 to have the same sum of the message changes, i.e., $a_1 + a_8 = a_2 + a_4$, $a_2 + a_6 = a_3 + a_{11}$ and $a_1 + a_9 = a_3 + a_{10}$ where the addition is defined by the group addition operator. It is easy to see that the encodability and decodability can be satisfied by using the same mapping on $p_i,\ i = 1 \cdots, 4$.

Interestingly, the encodability and decodability can also be satisfied by using different mappings for $p_i,\ i = 1 \cdots, 4$. For example, an encoding grid based on the Klein four-group is shown in Figure 17-(c) and an encoding grid based on $\mathbb{Z}_4$ is shown in Figure 17-(d). We note that the whole encoding grid can be generated by using the mappings on $p_i,\ i = 1 \cdots, 4$ recursively. Therefore, we also call the encoding grid of $p_i, i = 1, \cdots 4$ as the fundamental encoding grid. In detail, in the first step, we treat p_1 as the origin and determine the mappings used on p_2, p_3 and p_4. Next step, we treat p_2, p_3 and p_4 as new origins and determine the mappings on the points which are one-edge away from the new origins by finding the correspondence between the mappings used on p_1 and that used on p_2, p_3 and p_4. One can generate the whole encoding grid by doing this recursively. More details about the recursive generation of the encoding gird will be discussed soon. To handle arbitrarily i.i.d. input distribution, the encoding grid should have roughly equal number of +1's, +2's and +3's on all three axes. By enumerating all the possibilities, it can be shown that this cannot be satisfied by designing the fundamental encoding grid or using different groups. Therefore, there is no robust codes when $n = 3$. □

Tiling is one of the most commonly known concepts in combinatorics. Intuitively, an n-dimensional shape $\mathcal{S}$ tiles the n-dimensional space $\mathbb{Z}^n$ if disjoint copies of $\mathcal{S}$ cover $\mathbb{Z}^n$. The construction of our encoding grid for general n is related to the problem of

periodically tiling the encoding grid by copies of different fundamental encoding grids. But this interesting connection does not make the problem easier. In this case, each tile defines the message change for each edge in the encoding grid. The constraint between adjacent tiles is the decodability of the newly added grid points.

In general, when $N_B = N_A = n$, the construction of an n-dimensional encoding grid can be done as follows. First, we need to construct the fundamental encoding grid which is defined at the origin, all its one-step encoding neighbors and the edges starting from these points where each edge is associated with an element in A. Without loss of generality, we assume a_i is associated with the edge along the i-th dimension on the origin and the encoding function on the origin is denoted as $E_0 \triangleq \{a_1, a_2, \ldots, a_n\}$.

The construction of the fundamental encoding grid needs to satisfy two constraints. The first constraint is the encodability which means each element in A occurs once and only once on every grid point in the fundamental encoding grid. Let $\{\pi_j : \mathbb{Z}_n \to \mathbb{Z}_n, j = 1, 2, \ldots, n!\}$ be the set of permutations of $\mathbb{Z}_n$. The encoding functions of the one-step encoding neighbors of the origin are determined by the fundamental encoding grid and denoted as $E_i, i = 1, \ldots, n$. We can find j_i such that $E_i = \pi_{j_i}(E_0) = \{\pi_{j_i}(a_1), \pi_{j_i}(a_2), \ldots, \pi_{j_i}(a_n)\}$. The second constraint is the decodability which means the sum of the edge values on different paths between the origin and any of its two-step encoding grid points should be the same. Note that there are $n!$ different ways to associate the elements in A to the edges of a grid point such that the encodability is satisfied. One trivial construction of the fundamental encoding grid is to use the same encoding function on all the grid points in the fundamental encoding grid. However, the construction of the fundamental encoding grid may not be unique.

After we construct the fundamental encoding grid, we can tile the whole encoding

grid in the following way. For the i-th (indexed by the dimension) one-step encoding neighbor of the origin, its encoding function is already constructed by the fundamental encoding grid of the origin. We can construct the encoding function of its k-th one-step encoding neighbor as $\pi_{j_i}(E_k)$. By doing this recursively, different permuted copies of the fundamental encoding grid can be used to tile the whole encoding grid and the encodability and decodability are both satisfied. This reduces the construction of the encoding grid to the construction of the fundamental encoding grid. A robust code requires roughly the same number of each element in A along each dimension. The existence of the robust codes for general n remains an open problem. The main difficulty to make the general statement is that the construction of the fundamental encoding grid when $n > 3$ is complicated and group of size $n + 1$ is not unique.

In the following discussion, we will see how to design robust modulation codes to combat non-uniform i.i.d. input distribution using a different approach.

4. Self-randomized Modulation Codes (SRMC)

If $N_B = N_A + 1$ and we only increase one cell-level by 1 at each rewrite, then we can define a self-randomized modulation code (SRMC) [79] by using an integer dither sequence r_t to modify the encoded sequence x_t. Let $r_t = \phi(s_t)$ be a sequence which is uniquely defined by s_t. For example, if $H = \mathbb{Z}_q^n$ and $B = \{e_0, e_1, \ldots, e_{n-1}\}$ is the set of unit vectors, then we can choose $\phi(s_t) = \|s_t\|_1$ (i.e., the ℓ_1 norm) and this implies that $r_{t+1} = r_t + 1$. Let $\pi_i : G \to G$ be a sequence of permutations on G for $i = 0, 1, \ldots, (q-1)^n - 1$. Then, we can define the dithered decoder to be

$$\widetilde{D}\left(s_t\right) = \pi_{\phi(s_t)}\left(D(s_t)\right)$$

and the dithered encoder to be

$$\widetilde{E}\left(s_{t-1}, x_t\right) = \begin{cases} s_{t-1}, & \text{if } x_t = \widetilde{D}\left(s_{t-1}\right) \\ s_{t-1} + D^{\dagger}\left(\pi^{-1}_{\phi(s_{t-1})}\left(x_t - \widetilde{D}(s_{t-1})\right)\right), & \\ & \text{otherwise} \end{cases}.$$

Lemma F.4. *Let elements of G be $g_0, g_1, \ldots g_{M-1}$ in some arbitrary order and let π_i be a left circular shift by $\frac{i(i+1)}{2} \bmod M$ places in this order. In other words, let $\pi_i(g_j) = g_k$ with $k = \left(j + \frac{i(i+1)}{2}\right) \bmod M$. Then, the expected cell-level increment is the same for every cell after every integer multiple of M steps.*

Proof. The difference of the dither term $\frac{i(i+1)}{2}$ between π_i and π_{i-1} is $\frac{i(i+1)}{2} - \frac{i(i-1)}{2} = i$. Each time, the input distribution is circularly (due to the modulo M operation) shifted by 1. Therefore any integer multiple of M steps of permutations transfers any input distribution into a uniform distribution, i.e., any integer multiple of M steps of permutations map the elements in G to each element in A equally number (more than 1) of times. □

Lemma F.5. *If $N_B = N_A + 1$, SRMC allows only one cell-level increment at each rewrite and the cell index is uniform over time for arbitrary non-uniform i.i.d. input distributions.*

Proof. If we have one extra cell, the valid encoder/decoder can choose N_A cells and have the one-to-one mapping between the those cells and the message changes. Different choice of these N_A cells gives different encoder up to isomorphism. Therefore, there are $N_A + 1$ different encoders. This allows us to randomize over all the encoders to make the encoder robust against non-uniform but i.i.d. input distributions. □

5. Self-randomized Modulation Codes with Input Decomposition (SRMC-ID)

Although the SRMC is shown to be asymptotically optimum as $q \to \infty$ for arbitrary l, k, and n [79], when q is small, the number of rewrites can be improved significantly by allowing input decomposition and multiple cell-level increments when the desired cell is full. When the cell given by the SRMC encoder is full, instead of performing block erasure, we can write the input as the sum (modulo l^k) of two numbers. Then the input can be stored by encoding these two numbers separately which causes two cell-levels to be increased per input. Note that the analysis on the average number of rewrites is difficult for this scheme. But the worst case performance can be analyzed easily.

Theorem 13. *SRMC-ID guarantees* $\frac{1}{2}\left(\left\lfloor \frac{n}{2} \right\rfloor - 1\right)(q-1)$ *rewrites before any cell-level reaches* q*.*

Proof. Note that there are $\left\lfloor \frac{n+1}{2} \right\rfloor$ ways to decompose any input number. So this scheme fails only when more than half of the cells are full. Since the failure occurs only if more than half of the cells are full, the worst case input will keep writing the same cell until it is full, then the encoder starts to decompose the input. For any input value, it takes $\frac{1}{2}\left(\left\lfloor \frac{n}{2} \right\rfloor - 1\right)(q-1)$ steps to fill up the other $\left(\left\lfloor \frac{n}{2} \right\rfloor - 1\right)$ cells. Therefore, the total number of rewrites for the worst case is $\left(1 + \frac{1}{2}\left(\left\lfloor \frac{n}{2} \right\rfloor - 1\right)\right)(q-1)$. □

Remark 22. *Comparing to the upper bound shown in [75], which is* $[n - k(l-1) + 1](q-1)$*, this scheme achieves roughly 1/4 number of rewrites of this upper bound. To be fair, we must point out that in the SRMC, each rewrite stores* $\log_2(l^k - 1)$ *bits of information since the SRMC allows arbitrary change of the message. But the the bound given in [75] considers only one variable changed each rewrite, i.e.,* $\log_2 k(l-1)$ *bits of information is stored per rewrite.*

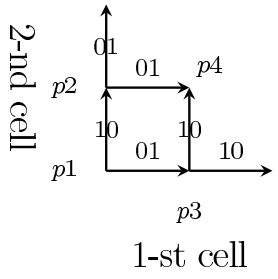

Fig. 18. Encoding grid of the FLM(2,2,2) code

Example 1. *The floating code with parameter $k = l = n = 2$, which is proposed in [73], which is denoted as the FLM(2,2,2) code here, can be described as a group theoretic framework as follows.*

Let $G = \{00, 01, 10, 11\}$, $A = \{10, 01\}$, $B = \{10, 01\}$ and $H = \mathbb{Z}_q^2$. The encoder is defined as

$$\widetilde{E}\left(s_{t-1}, x_t\right) = s_{t-1} + b_{i^*}$$

where $i^* = \|s_{t-1}\|_1 + j$ and $x_t - x_{t-1} = a_j$, $j \in \{0, 1\}$. Note that all the arithmetic is modulo 2. There are also two possible mappings between A and B and the ℓ_1 norm of s_{t-1} is serving as the dither sequence to randomize the choices of mappings. The difference between the SRMC and the FLM (2,2,2) code is, the FLM (2,2,2) code only allows two changes instead of all 4 possible changes as SRMC does, (i.e., the FLM code stores 1 bit of information per rewrite by using 2 cells while SRMC store 2 bits of information per rewrite by using 2 cells) This constraint on A provides the extra degrees of freedom to randomize the encoders. It is also interesting to look at the encoding grid of the FLM(2,2,2). Figure 18 shows the encoding grid of the FLM(2,2,2) code. It can be seen that the mappings used on the grid points are alternating between two different mappings. For large q, there are roughly the same number of 10's and 01's on both axes. Therefore the FLM(2,2,2) code is asymptotically robust against arbitrarily i.i.d. input distributions even with $N_A = N_B$.

We note that this does not conflict with the conclusion of $n = 2$ in subsection 3 where we allow arbitrary change on the input and $|G| = 2$. In the FLM(2,2,2) code, $N_A < |G| - 1$, i.e., the input variable is not allowed to change arbitrarily and $|G| = 4$. Since $|G| = 4$, there are two groups, i.e., the Klein four-group and $\mathbb{Z}_4$. The encoding grid of FLM(2,2,2) code is based on the Klein four-group. It is easy to see, if $\mathbb{Z}_4$ is used, all the grid points must use the same mapping to satisfy the encodability and decodability. We can say, the FLM(2,2,2) code sacrifices information per rewrite to obtain the robustness against arbitrarily i.i.d. input distribution. The SRMC can be thought of as another way to trade-off between the storage efficiency, which is defined as the amount of information stored per cell-level between two block erasures, and the robustness. To compare the storage efficiency of FLM(2,2,2) and SRMC, one can either use 2 cells to storage 1 binary variable by using SRMC or use 2 cells to storage 2 binary variables but only allow 1 variable to change each time. It is obvious that the FLM(2,2,2) and SRMC are equivalent in terms of storage efficiency. Unlike the SRMC, the FLM code cannot be generalized to general parameters [73].

In the following subsection, we will see how to design codes which have better storage efficiency than the SRMC when q is finite by combining ideas from load-balancing theory with the group-theoretic approach.

6. Load-balancing Modulation Codes: The Power of Two Random Choices

An analogy between random-loading and random input cell-charging enables one to use the ideas from load-balancing theory to help make the cell-levels more uniform than SRMC does [79] when q is finite. There are many papers on load-balancing algorithms based on the power of d random choices, depending on how we choose the d random bins. For the m-balls-into-n-bins problem, the analysis [77, 76] shows that

when we choose d (not necessarily distinct) bins uniformly and independently, then the maximum load is $O\left(\frac{\ln\ln n}{\ln d}\right)$ with high probability (w.h.p) if the number of balls $m = cn$. If we first divide the bins into d parts and randomly (i.i.d.) choose one from each group each time, and a tie can be broken randomly, then the maximum load is $O\left(\frac{\ln\ln n}{d}\right)$. If the tie is broken by picking the lowest part number, then the maximum load is $O\left(\frac{\ln\ln n}{d\phi(d)}\right)$ where $\phi(d)$ is a constant less than 2 and dependent on d. Depending on different load-balancing algorithms, we can have different LBMC's. As the result, the average number of rewrites of the LBMC's may be analyzed by using the results in the load-balancing literature.

a. Type-A LBMC (LBMC-A)

Let $N_B = 2(N_A + 1)$ [3]. Let $r_1(t)$ and $r_2(t)$ be two dither sequences, namely, $r_1(t) = \phi_1(s_t)$ and $r_2(t) = \phi_2(s_t)$. For any message-change $a \in A$, we have two mappings $b_1 = \pi_{r_1(t)}(a)$ and $b_2 = \pi_{r_2(t)}(a)$ such that the following properties hold.

- $\pi_{r_1(t)}(A) \cup \pi_{r_2(t)}(A) = B$ and $\pi_{r_1(t)}(A) \cap \pi_{r_2(t)}(A) = \emptyset$ holds for all t.
- $\pi_{r_1(t)}(a)$ and $\pi_{r_2(t)}(a)$ for all $a \in A$ are pairwise uniform over $B \times B$.

The encoder of LBMC-A is as follows

$$E\left(s_{t-1}, x_t\right) = \begin{cases} s_{t-1}, & \text{if } x_t = D\left(s_{t-1}\right) \\ s_{t-1} + D^{\dagger}\left(x_t - D(s_{t-1})\right), & \\ & \text{otherwise} \end{cases}$$

[3]To use the power of two random choices, we privide two candidates $b_1, b_2 \in B$ for each input value $a \in A$. For the same reason, we need one extra cell for each N_A cells to allow randomization such that the cell-level increases uniformly for arbitarily i.i.d. input.

where

$$D^{\dagger}(a) = \pi_{r_{i^*(t)}}(a)$$

and

$$i^* = \arg\min_{i\in\{1,2\}} s_{t-1}\left(\pi_{r_i(t)}(a)\right).$$

Note that tie in the arg min can be broken arbitrarily. Note that the group-theoretic definition of LBMC-A is very general and we can have different designs for $\pi_{r_1(t)}(a)$, $\pi_{r_2(t)}(a)$ and the corresponding decoders. The following is an example of a particular LBMC-A.

Example 2. *[LBMC-A] The LBMC described in [79] is a particular design of an LBMC-A. In particular, Let* $N_B = 2(N_A+1) = 2^{k+1}$. *We choose* $r_1(t) = r_2(t) = \|s_{t-1}\|_1 + 1$ *where the addition is the integer addition. Let* H *be the Galois field* $GF(2^{k+1})$ *and* $h : \mathbb{Z}_{2^{k+1}} \to H$ *be a bijection that satisfies* $h(0) = 0$ *(i.e., the Galois field element 0 is associated with the integer 0). The two choices* $\pi_{r_1(t)}(a)$ *and* $\pi_{r_2(t)}(a)$ *are calculated as* $\pi_{r_1(t)}(a) = h^{-1}\left(c_t h(x_t) + d_t\right)$ *and* $\pi_{r_2(t)}(a) = h^{-1}\left(c_t h(x_t + 2^k) + d_t\right)$, *where* $c_t = h\left(\left(r_t \bmod 2^k - 1\right) + 1\right)$, *and* $d_t = h\left(r_t \bmod 2^k\right)$. *The encoder increases the lower cell by 1.*

b. Type-B LBMC (LBMC-B)

Based on the results in [77], the load-balancing performance can be further improved if we divide all the cells into d parts and we randomly pick one from each part each time. We can design a different LBMC as follows. Let $N_B = 2(N_A+1)$. Let $r_1(t) = \phi_1(s_t)$ and $r_2(t) = \phi_2(s_t)$ be two pairwise uniform dither sequences. The idea of LBMC-B is, we first divide all the cells into two parts, say, $s_t^{(1)} = \{s_t(0), \cdots, s_t(N_A)\}$ and $s_t^{(2)} = \{s_t(N_A+1), \cdots, s_t(2N_A+1)\}$ and apply an SRMC on each part. Denote the encoders and decoders of the two SRMC's as $\widetilde{E}_i(\cdot)$ and $\widetilde{D}_i(\cdot)$, $i = 1, 2$. Then decoder

of LBMC-B $\widetilde{D}(s_t)$ is as

$$\begin{aligned}\widetilde{D}(s_t) &= \widetilde{D}_1(s_t^{(1)}) + \widetilde{D}_1(s_t^{(2)}) \\ &= \pi_{r_1(t)}\left(D_1(s_t^{(1)})\right) + \pi_{r_2(t)}\left(D_2(s_t^{(2)})\right)\end{aligned}$$

where $\pi_{r(t)}(\cdot)$ is a permutation from G to G which is indexed by $r(t)$.

The encoder $E\left(s_{t-1}, x_t\right) = s_t$ is defined as

$$\begin{cases} s_t = s_{t-1}, & \text{if } x_t = \widetilde{D}\left(s_{t-1}\right) \\ s_t^{(1)} = s_{t-1}^{(1)}; s_t^{(2)} = s_{t-1}^{(2)} + D^{\dagger(2)}\left(\pi_{r_2(t-1)}^{-1}\left(x_t - \widetilde{D}(s_{t-1})\right)\right), & \\ & \text{if } x_t \neq \widetilde{D}\left(s_{t-1}\right) \text{ and } i^* = 2 \\ s_t^{(2)} = s_{t-1}^{(2)}; s_t^{(1)} = s_{t-1}^{(1)} + D^{\dagger(1)}\left(\pi_{r_1(t-1)}^{-1}\left(x_t - \widetilde{D}(s_{t-1})\right)\right), & \\ & \text{otherwise} \end{cases}$$

where

$$i^* = \arg\min_{i\in\{1,2\}} s_{t-1}\left(D^{\dagger(i)}\left(\pi_{r_i(t-1)}^{-1}\left(x_t - \widetilde{D}(s_{t-1})\right)\right)\right)$$

and ties are broken by always picking the cell in the first group. Note that there may be different encoder/decoder pairs can be design following this principal. A particular example is as follows.

Example 3. *[LBMC-B] Let* $r_1(t) = c_1\|s_t\|_1 + \left\lfloor \frac{\|s_t\|_1}{n} \right\rfloor$ *and* $r_2(t) = c_2\|s_t\|_1$ *where* c_1, c_2 *and* $n = N_A + 1$ *are pairwise co-prime. For input value* $a \in A$*, the cell index candidate in* $s_t^{(1)}$ *is chosen as* $\tilde{E}_1(a) = (a + r_1(t)) \bmod n$ *and the cell index candidate in* $s_t^{(2)}$ *is chosen as* $\tilde{E}_2(a) = ((a + r_2(t)) \bmod n) + n$*. It is easy to see* $\tilde{E}_1(a)$ *and* $\tilde{E}_2(a)$ *are pairwise uniform over time, i.e., each possible pair appears with equal number of times at every multiple of* n^2 *rewrites.*

Remark 23. *We note that when the number of rewrites is less than* n^2*, this scheme*

provides slightly better load-balancing performance than "always-go-left" random loading algorithm since the pairs are chosen in a deterministic way to avoid duplicated choice with certainty. In the "always-go-left" load balancing algorithm [77], although two candidates are chosen independently and uniformly, there is still some non-zero probability that the same pair is chosen more than once.

7. Analysis of SRMC and LBMC

The SRMC and LBMC algorithms are motivated by the load-balancing algorithms. Various load-balancing algorithms have been analyzed thoroughly in previous work. But the difference between SRMC/LBMC and the load-balancing algorithms is SRMC/LBMC are pseudo-random so that the analyisis of load-balancing algorithms does not extend easily. During the simulation, we observe that the load-balancing modulation codes perform very similarly or slightly better than their load-balancing counterparts. So we leave it as the following conjecture.

Conjecture 3. *Suppose n q-level cells are used to store k distinct l-ary random variables and each rewrite only increases one cell by 1. The input is an i.i.d. random variable over $\mathbb{Z}_l^k$, where $n = 2l^k$, and arbitrary change is allowed at each rewrite. When any cell level reaches q, the total number of rewrites by applying LBMC-A is denoted as Q_{LBMC-A} and the total number of rewrites by applying LBMC-B is denoted as Q_{LBMC-B}. As the counterparts, let's consider the load-balancing problem of loading infinite number of balls into n bins sequentially. If any bin has $(q-1)$ balls, the loading process stops and the total number of balls in all the bins is denoted as Q_{RL2C} if the RL2C algorithm is applied. The total number of balls by applying the always-go-right algorithm with two partitions is denoted as Q_{AGR2}. Then $E[Q_{LBMC-A}] \geq E[Q_{RL2C}]$ and $E[Q_{LBMC-B}] \geq E[Q_{AGR2}]$.*

Remark 24. *A rigorous proof is technically non-trivial and the main difficulty in analyzing the average maximum load for SRMC/LMBC is that the choices at different times are not independent. So the techniques used in the previous work on the balls-into-bins problem, e.g. [72][77][76], can not be applied. Therefore, we leave it as a conjecture and instead use simulation results to support this conjecture.*

G. Simulation Results

We simulate three load-balancing algorithms and three corresponding modulation codes. They are random loading with 1 choice (RL1C), random loading with two choices (RL2C), always-go-left with 2 choices, the SRMC described in Section a, the modulation code shown in Example 2 (LBMC-A) and the modulation code shown in Example 3 (LBMC-B). Note that we choose $n = 16$, q from 1 to 16 and calculate the normalized (by q) maximum cell levels for different schemes in Figure 19. In RL2C, two independent choices of the bins can be the same and tie is broken arbitrarily. In always-go-left, the two bins are chosen independently and uniformly from two groups and tie is broken by always picking the left cell. In LBMC-A, two bins are chosen pairwise uniformly as described in [79] and tie is broken arbitrarily. In LBMC-B, two bins are chosen pairwise uniformly from two groups and tie is broken by always choosing the left cell.

From the simulation results, we can see that SRMC, LBMC-A and LBMC-B match the corresponding load-balancing algorithms very well. Note that when q is small, these modulation codes perform slightly better than the load-balancing algorithms due to the deterministic randomness of the modulation codes.

We also simulate the SRMC with input decomposition (SRMC-ID) algorithm and compare it to RL1C and RL2C algorithms. The simulation results are shown

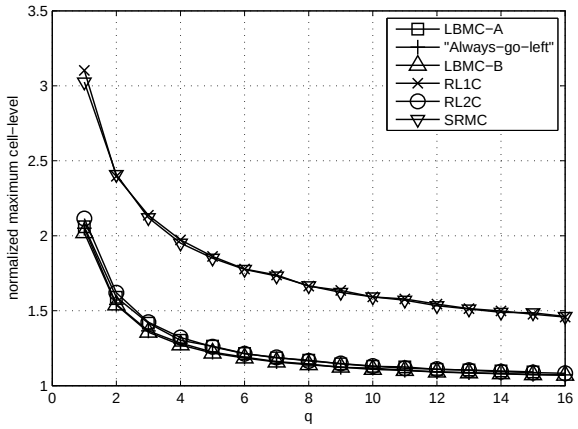

Fig. 19. Simulation results of load-balancing algorithms and the corresponding modulation codes with $n = 16$.

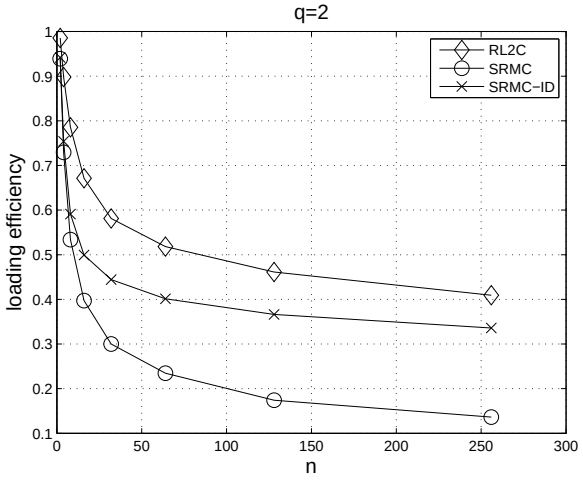

Fig. 20. Loading efficiency of SRMC-ID, RL1C and RL2C algorithms when $q = 2$

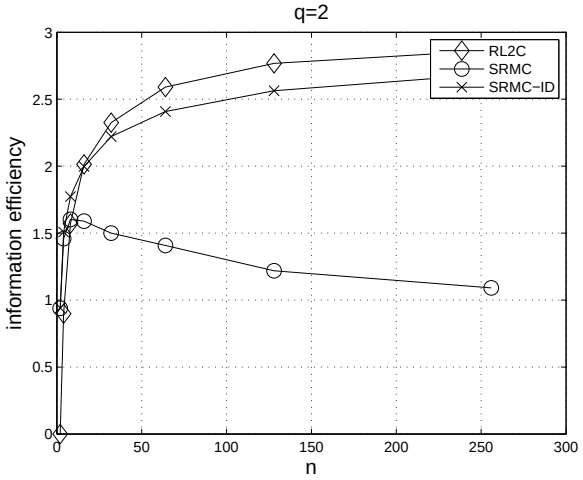

Fig. 21. Information efficiency of SRMC-ID, RL1C and RL2C algorithms when $q = 2$

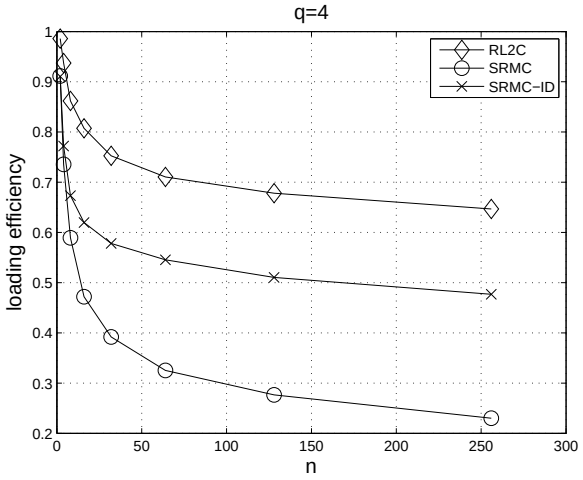

Fig. 22. Loading efficiency of SRMC-ID, RL1C and RL2C algorithms when $q = 4$

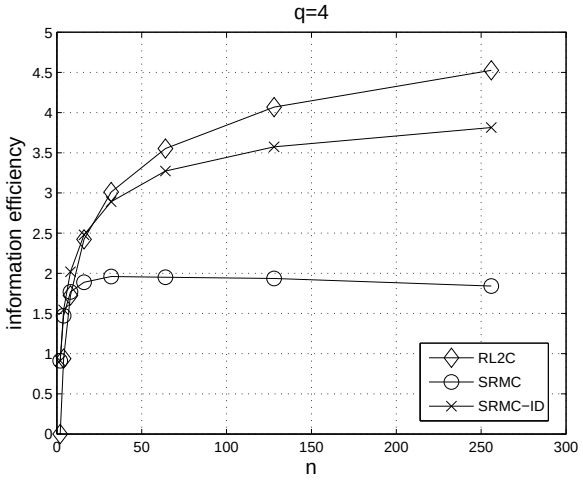

Fig. 23. Information efficiency of SRMC-ID, RL1C and RL2C algorithms when $q = 4$

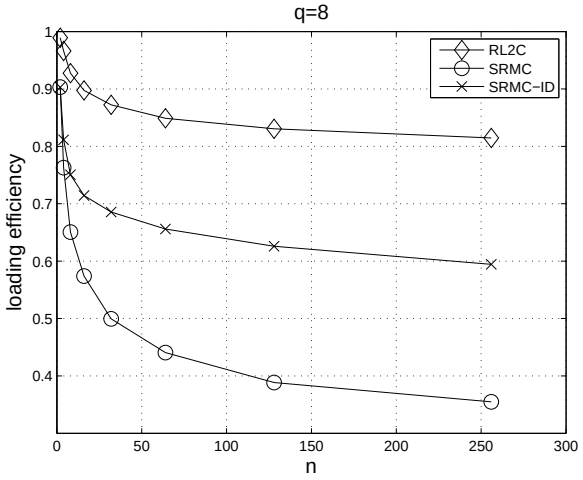

Fig. 24. Loading efficiency of SRMC-ID, RL1C and RL2C algorithms when $q = 8$

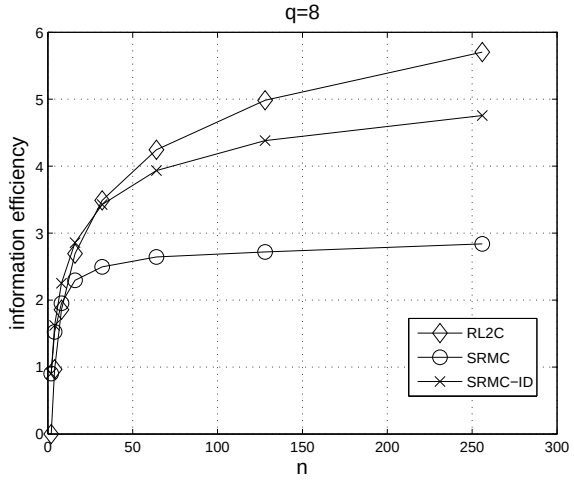

Fig. 25. Information efficiency of SRMC-ID, RL1C and RL2C algorithms when $q = 8$

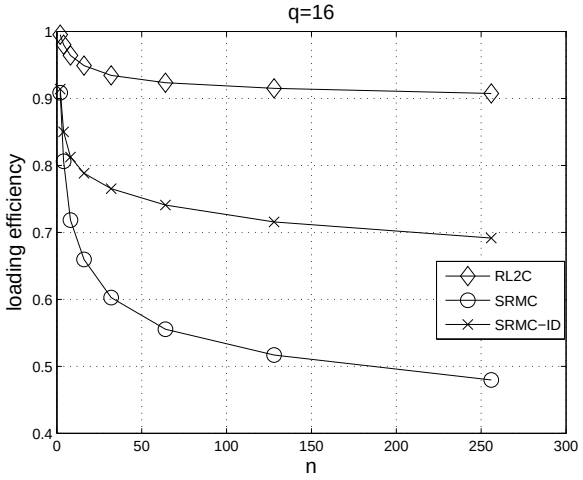

Fig. 26. Loading efficiency of SRMC-ID, RL1C and RL2C algorithms when $q = 16$

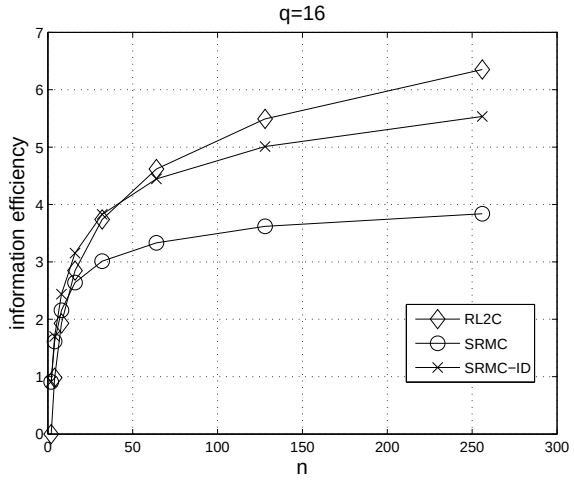

Fig. 27. Information efficiency of SRMC-ID, RL1C and RL2C algorithms when $q = 16$

in Figure 20 to 27. In the simulation, we choose $q = 2, 4, 8$ and 16, and $n = 2^i$, $i = 1, 2, \ldots, 8$. We define the loading efficiency as the number of rewrites normalized by $n(q-1)$ when any cell level reaches q. We also define the information efficiency as the amount of information stored (in terms of bits) normalized by $n(q-1)$ when any cell level reaches q. We compare the loading efficiency and information efficiency of RL2C, SRMC and SRMC-ID algorithms. In SRMC and RL2C algorithms, if any cell level reaches q, we stop the simulation and calculate the loading efficiency and information efficiency, which is the number of rewrites normalized by $n(q-1)$. In SRMC-ID, if the desired cell is full, the input is decomposed into two values and two non-full cells are increased by 1 each. This gives 1 rewrite, but 2 cell increases. It can be seen that SRMC-ID improved the average number of rewrites of SRMC significantly.

CHAPTER V

LDPC CODES FOR RANK MODULATION IN FLASH MEMORIES

A. Introduction

In the previous chapter, we discuss different ways to design efficient modulation codes which improve the reuse efficiency and extend the life-span of the device. In this chapter, we will discuss the error-correcting codes (ECC's) design for flash momories.

Research has already been done on error-correcting codes (ECCs), rewriting codes and capacity analysis for rank modulation. Based on the *Kendall tau distance*, error-correcting codes for a single rank modulation block were explored in [63, 66]. In [66], a family of single-error-correcting codes was constructed, whose number of codewords is provably at least half of optimal. In [63], the asymptotic rates of optimal ECCs were derived, and the existence of good t-error-correcting codes was proved. Rewriting codes that enable data to be rewritten without block erasures and capacity analysis were also presented [65, 67].

In this chapter, we study LDPC codes for rank modulation. Compared to codes designed for a single rank modulation block, where the cell levels are ordered into a permutation [63, 66], our LDPC code approach partitions the cells into many small groups and uses rank modulation separately for each group. In this work, we treat the rank modulation code as part of the channel, as shown in Fig. 28. We consider the physical channel together with the rank modulation encoder and decoder as the equivalent channel for the ECC. We analyze the equivalent channel, and study the design of good LDPC codes for the channel. A family of symbol-flipping verification-based (SFVB) decoding algorithms is proposed and analyzed.

The structure of the chapter is as follows. In Section B, we study the equivalent

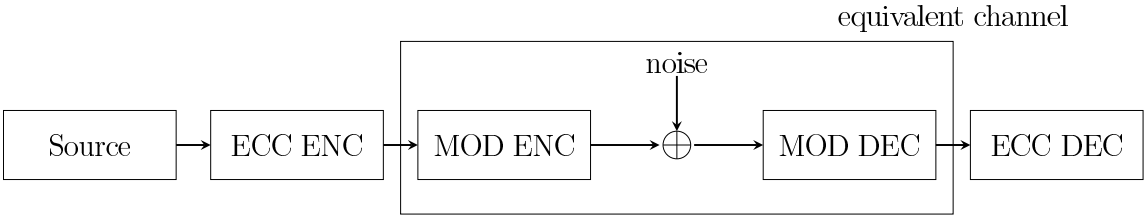

Fig. 28. Block diagram for ECC with modulation codes for flash memory.

channel model of rank modulation. Section C presents the LDPC codes design and the performance analysis for rank modulation. Section D presents the simulation results.

B. Equivalent Channel of Rank Modulation

In this section, we define rank modulation and its LDPC codes, and analyze the equivalent channel.

1. Rank Modulation

Consider a group of n cells, whose levels are $c_1, c_2, \ldots, c_n$. Here $c_i \in \mathbb{R}$ for $i = 1, \ldots, n$, and represents the amount of charge stored in the i-th cell. Let S_n be the set of $m = n!$ permutations of length n. Specifically, we have $S_n = \{s_1, s_2, \cdots, s_m\}$ where, for $i = 1, \ldots, m$,

$$s_i \triangleq (s_i(1), s_i(2), \cdots, s_i(n))$$

is a permutation of $(1, 2, \cdots, n)$. The rank modulation scheme [65] defines a mapping $\mathcal{R}$

$$\mathcal{R} : \{(c_1, \ldots, c_n) \in \mathbb{R}^n\} \rightarrow S_n$$

as follows. If $\mathcal{R}(c_1, c_2, \ldots, c_n) = (i_1, i_2, \ldots, i_n)$, then for any $j_1 \neq j_2 \in \{1, \ldots, n\}$, $i_{j_1} > i_{j_2}$ if and only if $c_{j_1} \geq c_{j_2}$. Namely, the function $\mathcal{R}$ ranks the n cells based on the relative order of their cell levels. The rank modulation scheme uses the permutation induced by the cell levels, namely $\mathcal{R}(c_1, \ldots, c_n)$, to represent data.

Let $\mathbb{Z}_m = \{1, 2, \ldots, m\}$ be the ring of integers modulo m. Let Π be a bijection:

$$\Pi : S_n \to \mathbb{Z}_m.$$

As a modulation code, the rank modulation scheme has an encoder and a decoder, as shown in Fig. 28. Given a variable $x \in \mathbb{Z}_m$ as input to the encoder, the flash memory programs the n cells such that their cell levels $(c_1, \ldots, c_n)$ satisfy

$$\Pi(\mathcal{R}(c_1, \ldots, c_n)) = x.$$

For the decoder, it takes the cell levels $(c_1, \ldots, c_n)$ as input, and outputs the variable $\Pi(\mathcal{R}(c_1, \ldots, c_n)) \in \mathbb{Z}_m$.

2. LDPC Codes over Integer Rings for Rank Modulation

One way to design LDPC codes for rank modulation is to match the alphabet size of the LDPC codes and rank modulation. We can define the LDPC code over the integer ring $\mathbb{Z}_m$. As Fig. 28 shows. the outputs of the LDPC code encoder are mapped to permutations in S_n by Π. Then the permutation is represented by cell levels by rank modulation. The rank modulation demodulator calculates the permutations implied by the cell levels. The permutations are mapped to elements in $\mathbb{Z}_m$ for the LDPC code decoder.

An element $x' \in \mathbb{Z}_m$ has a multiplicative inverse, and is called *invertible*, if and only if x' and m are co-prime (i.e., $\gcd(x', m) = 1$). Let N be the block length of the code, and let M be the number of parity check equations. Every symbol of the

code is realized by a group of n cells using rank modulation, so the code corresponds to nN cells in total. Let $\Omega \subseteq \mathbb{Z}_m$ be the subset of invertible elements which is also known as the multiplicative group of $\mathbb{Z}_m$.

An LDPC code over $\mathbb{Z}_m$ is defined by its parity-check matrix H, which is an M by N sparse matrix whose non-zero entries are chosen independently and uniformly from Ω. A valid codeword $\mathbf{X} \in \mathbb{Z}_m^N$ should satisfy all the parity check equations, i.e., $\mathbf{X}H = 0$. The number of invertible elements $|\Omega|$ is given by $\varphi(m)$, where $\varphi(\cdot)$ is Euler's totient function. It is known that large integer rings must have a reasonable number of invertible elements because $\varphi(m)$ can be bounded by $\varphi(m) \geq m^{1-\epsilon}$ for arbitrary $\epsilon > 0$ and large enough m.

We can obtain the generator matrix G by using Gaussian elimination to place H in the row reduced form. In particular, one needs to find an invertible element as the pivot, and use it to zero out all the elements below and above it. If the element on the i-th row and the i-th column is not an invertible element, then we will look for an invertible element from $H_{j,k}$, where $j \geq i$ and $k \geq i$, and exchange the i-th row with the j-th row and exchange the i-th column with the k-th column. Note that elementary row operations do not change the code, and exchanging two columns only changes the order of the two code symbols.

For some parity-check matrices, the Gaussian elimination may get stuck before one finds the row-reduced echelon form of H (e.g., there is no invertible element available for exchange). However, this seems to occur with very small probability when H is sparse and m is large. The probability of finding a G from H is easily seen as equivalent to the probability of H being full rank. The results in [64] show that a uniform random matrix over $\mathbb{Z}_m$ is full rank with high probability as $m \to \infty$. Restricting our attention to sparse matrices whose non-zero entries are invertible changes the setting of the problem, and we do not have analytical results in this case.

But numerical experiments show that one can almost always find a generator matrix G by trying Gaussian elimination on several randomly chosen H's.

During the analysis and numerical simulation, we find that the LDPC codes over integer rings can cause non-negligible false verification if a message-passing style decoder is applied. However, this can be fixed by designing LDPC codes over finite fields shown in the following subsection.

3. LDPC Codes over Finite Fields for Rank Modulation

Another way to design LDPC codes for rank modulation is to design the codes over a finite field $GF(q)$ which is embedded into $\mathbb{Z}_m$ such that $q \leq m$ [22, 69]. In detail, let $\mathcal{L}$ be a subset of $\mathbb{Z}_{n!}$ with q elements. To associate $\mathcal{L}$ with $GF(q)$, one can design an arbitrary one-to-one correspondence between $\mathcal{L}$ and $GF(q)$ which is denoted as $\wp : \mathcal{L} \rightarrow GF(q)$.

The parity-check matrix H is an M by N matrix whose non-zero elements are randomly chosen from $GF(q)$ and the codes are defined over $GF(q)$.

The input and output alphabet sets of the encoder are both $GF(q)$ and the coded symbols are mapped to the permutations in $\mathcal{L}$ symbol by symbol by $\wp$. Each permutation is then modulated by rank modulation and passes through the physical channel. The rank modulation demodulator calculates the permutation implied by the cell levels read back from the channel and the permutation is mapped back to $GF(q)$ as the input of the LDPC code decoder. Note that sometimes the output of the rank modulation demodulator is not in $\mathcal{L}$ due to the channel error. In this case, the outputs of the rank modulation demodulator need to be pre-processed before LDPC code decoding. We will discuss more details about this in next section.

Comparing to the codes over integer rings described in previous subsection, here we only use q symbols out of all m possible symbols implied by rank modulation

during the encoding and this causes a rate loss by a factor of $\zeta_n \triangleq \frac{\log q}{\log n!}$. For example, $\zeta_5 = 0.869$, $\zeta_6 = 0.948$, $\zeta_7 = 0.976$ and $\zeta_8 = 0.980$. This field-embedding modification leads to codes with lower false verification probability and better performance at the error floor regime [22, 69].

4. Equivalent Channel

In this subsection, we consider the equivalent channel for rank modulation. By treating the modulation encoder and decoder as part of the channel, the input and output alphabet sets of the equivalent channel are both $\mathbb{Z}_m$. Let the input variable be x, and the output variable be y, where $x, y \in \mathbb{Z}_m$. We first derive the transition probability of the equivalent channel, namely, the probability of y given x, which we denote by $W(y|x)$. Due to the difficulty of having a closed-form mathematical description, we derive an approximation of this channel when the noise is i.i.d. additive Gaussian with small variance.

Let us first consider the general case. Consider a group of n cells (which represents a code symbol), and let the physical noise in the cell levels have the joint probability density function (pdf) $f(w_1, w_2, \cdots, w_n)$, where w_i is the noise in the i-th cell's level. We need to find the channel transition probability $W(y|x)$. Say that the rank-modulation encoder maps the input integer x to $s_i = \Pi^{-1}(x)$, and programs the cell levels as $c = \{c_1, c_2, \cdots, c_n\}$ such that $\mathcal{R}(c) = s_i$. Then the cell levels are distorted by the additive noise $w = (w_1, w_2, \cdots, w_n)$, where w_i is the noise in the i-th cell level.

The rank-modulation decoder reads the noisy cell levels $c' = \{c_1 + w_1, c_2 + w_2, \cdots, c_n + w_n\}$, computes the permutation $s_j = \mathcal{R}(c')$, and outputs the integer $y = \Pi(s_j)$. (Note that the mapping $\Pi(\cdot)$ can be chosen arbitrarily.) Without loss of generality, we assume that $\Pi^{-1}(1) = (1, 2, \cdots, n)$, the identity permutation. Since

Π is a bijection, we also abuse notation and write $W(s_j|s_i) = W(y|x)$ for a channel whose inputs and outputs are permutations.

Let π be a permutation on $\{1, 2, \ldots, n\}$. Since permutations naturally operate on each other via composition, we have $\pi \circ s_i = (\pi(s_i(1)), \ldots, \pi(s_i(n)))$. For any fixed bijection Π, the equivalent channel is *symmetric* if and only if for all s_i and s_j, $W\left(\Pi(s_j)|\Pi(s_i)\right) = W\left(\Pi(\pi \circ s_j|\Pi(\pi \circ s_i)\right)$ holds for all π.

When the noise is i.i.d., it is easily seen that the equivalent channel is symmetric. Therefore, without loss of generality, we assume the input variable is 1, and analyze $W(y|1)$. For any input integer x, the channel transition probability is $W(y|x) = W(y'|1)$ where $y' = \Pi(\pi \circ \Pi^{-1}(y))$, and π is the unique permutation satisfying $\pi \circ \Pi^{-1}(x) = \Pi^{-1}(1)$.

Let $s_1 = \Pi^{-1}(1) = (1, 2, \cdots, n)$, $c = (1, 2, \ldots, n)$, and $c' = c + w$. Then, the hard decision receiver for rank modulation computes $y = \mathcal{R}(c') = \mathcal{R}((c_1 + w_1, \ldots, c_n + w_n))$ and the channel transition probability $W(s_j|s_1)$ is given by

$$W(s_j|s_1) = \int_{\mathcal{A}_j} f(w_1, w_2, \cdots, w_n) \mathrm{d}w_1 \mathrm{d}w_2 \cdots \mathrm{d}w_n$$

where $\mathcal{A}_j = \mathcal{R}^{-1}(s_j)$ is the decision region for s_j. The integration domain $\mathcal{A}_j$ can be simplified to the intersection of $n - 1$ half spaces given by

$$\mathcal{A}_j \triangleq \left\{ w \;\middle|\; \begin{array}{c} w_{s_j(1)} + c_{s_j(1)} < w_{s_j(2)} + c_{s_j(2)} \\ w_{s_j(2)} + c_{s_j(2)} < w_{s_j(3)} + c_{s_j(3)} \\ \vdots \\ w_{s_j(n-1)} + c_{s_j(n-1)} < w_{s_j(n)} + c_{s_j(n)} \end{array} \right\}.$$

This decision region remains valid for the general case where the cell levels are not equally separated and the noise has an arbitrary joint distribution on the n cells. Since the integral cannot be further simplified without making more assumptions, it

is computed numerically in practice using Monte Carlo integration.

In the following, we assume that the noise is i.i.d. Gaussian with zero mean and small variance σ^2. We also assume that all the adjacent cell levels are separated equally by a constant Δ. (Note that for rank modulation, the memory only needs to use comparison circuits to program and read cells, and the real cell levels are unknown to the decoder. Results obtained by assuming evenly-spaced cell levels can be extended later for more robust code design.)

To study the approximate channel transition probability $\tilde{W}(y|x)$, we first define two concepts.

Definition 1. *An* adjacent transposition *in a rank-modulation symbol is a transposition of adjacent cell levels (among the n cells). For example, an adjacent transposition in a permutation induced by the n cell levels, causes the two integers i and $i+1$ (for some $i \in \{1, 2, \ldots, n-1\}$) to switch their positions.*

Definition 2. *For $i = 0, 1, \ldots, \frac{n(n-1)}{2}$, for any permutation $s \in S_n$, we define $\mathcal{N}_i(s)$, called the i-th neighborhood of s, as follows: the minimum number of adjacent transpositions to change any permutation in $\mathcal{N}_i(s)$ to s (and vice versa) is i. Namely, $\mathcal{N}_i(s)$ is the set of permutations at Kendall tau distance i from s. We note that $|\mathcal{N}_1(s)| = n - 1$.*

Given the input symbol $s \in S_n$ to the channel, the output symbol will be s with probability $1 - p$. Let p_1 be the approximate probability that the output symbol is $s' \in \mathcal{N}_1(s)$. For small noise, it can be shown that $p \approx |\mathcal{N}_1(s)|p_1 = (n-1)p_1$, that is, the dominant errors are the errors causing one adjacent transposition. The following lemma makes this notion precise, Due to the limited space, its proof is omitted.

Lemma 8. *For additive i.i.d. Gaussian noise with small variance σ^2, the channel model can be approximated as follows. (We call it the* approximate channel *model.)*

Let $x, y \in S_n$ be the input and output permutations to the channel, respectively. Then the channel transition probability for the approximate channel is

$$\tilde{W}(y|x) = \begin{cases} 1-(n-1)p_1 + o(p_1) & \text{if } x = y \\ p_1 + o(p_1) & \text{if } y \in \mathcal{N}_1(x) \\ O(p_1^2) & \text{otherwise} \end{cases} \tag{5.1}$$

where $p_1 = \frac{\sigma}{\sqrt{\pi}\Delta} \exp\left(-\frac{\Delta^2}{4\sigma^2}\right)$.

Proof. The idea of the proof is that we first show the probability of any pair of adjacent cell-levels getting flipped is on the order of $o(\exp(-\Delta^2))$. Therefore only those pairs which are separated by Δ are likely to be flipped. The second step is to show the dominant error event is that only one adjacent pair gets flipped.

Consider two cell levels which are separated by Δ. Let the i.i.d. Gaussian noise be w_1 and w_2. The probability that they are flipped is

$$p_0 \triangleq \int_{w_2=-\infty}^{\infty} \int_{w_1 \geq w_2+\Delta}^{\infty} f(w_1) f(w_2) \mathrm{d}w_1 \mathrm{d}w_2$$

where $f(\cdot)$ is the pdf of the Gaussian noise. By symmetry,

$$p_0 = \int_{w=-\infty}^{-\frac{\sqrt{2}}{2}\Delta} f(w)\mathrm{d}w = Q\left(\frac{\Delta}{\sqrt{2}\sigma}\right).$$

It is well know that when σ is small, a good approximation of the Q function is $Q(x) = \frac{1}{\sqrt{2\pi}x} \exp\left(-\frac{x^2}{2}\right)\left(1 + O(x^{-1})\right)$. Therefore, we have

$$p_0 = \frac{\sigma}{\sqrt{\pi}\Delta} \exp\left(-\frac{\Delta^2}{4\sigma^2}\right)(1 + O(\sigma)) = p_1 + o(p_1).$$

The probability that two cell levels which are separated by $i\Delta$ are flipped is $p_i = O(p_1^{i^2})$.

Next, we will show that, as noise variance goes to zero, with high probability,

there is only one pair of adjacent cell-levels being flipped when an error occurs. Note that there are two possible locations for two pairs of cell-levels, i.e., they are either disjoint and independent or they share a cell. If two pairs share a cell and both pairs are flipped, the probability of this can be upper bounded by the probability that the lowest cell-level and the highest cell-level of these three cells are flipped, which is $O(p_1^4)$. On the other hand, if two pairs are independent, it is easy to see that the probability that both pairs are flipped is $O(p_1^2)$. Since the number of choices of two pairs are independent of p_1, the probability that there are more than 1 adjacent and disjoint pairs of cell levels being flipped is $O(p_1^2)$. Therefore, each error symbol in $\mathcal{N}_1(x)$ occurs with probability $p_1 + o(p_1)$. The probability that the error symbol is not in $\mathcal{N}_1(x)$ is $O(p_1^2)$. The symbol is correct with probability $1-(n-1)p_1+o(p_1)$. Note that we use the Landau notation by following the definition of [19]. All the Landau notations are used to described the behavior of some variables when σ is near 0. $\square$

Remark 25. *Note that the above analysis still holds even when the cell-level separation is not uniform. This is the case if we consider rewriting using rank modulation. The similar analysis shows that each error symbol in $\mathcal{N}_1(x)$ occurs with probability $p_1 + o(p_1)$. The error symbol $y \notin \mathcal{N}_1(x)$ occurs with probability $O(p_1^2)$ and the symbol is correct with probability $1 - |\mathcal{N}_1(x)|\, p_1 + o(p_1)$. The only difference is that there are only $m \leq n-1$ symbols in $\mathcal{N}_1(x)$ where there are only m adjacent pairs of cell-levels which are separated by Δ and the other $n-1-m$ pairs of adjacent cell-levels are separated by $c\Delta$ with $c > 1$.*

Note that we evaluate the channel capacity for the real channel and the approximate channel when $n = 5$. The results are shown in Fig. 29. The unit of the y-axis is a symbol per channel use.

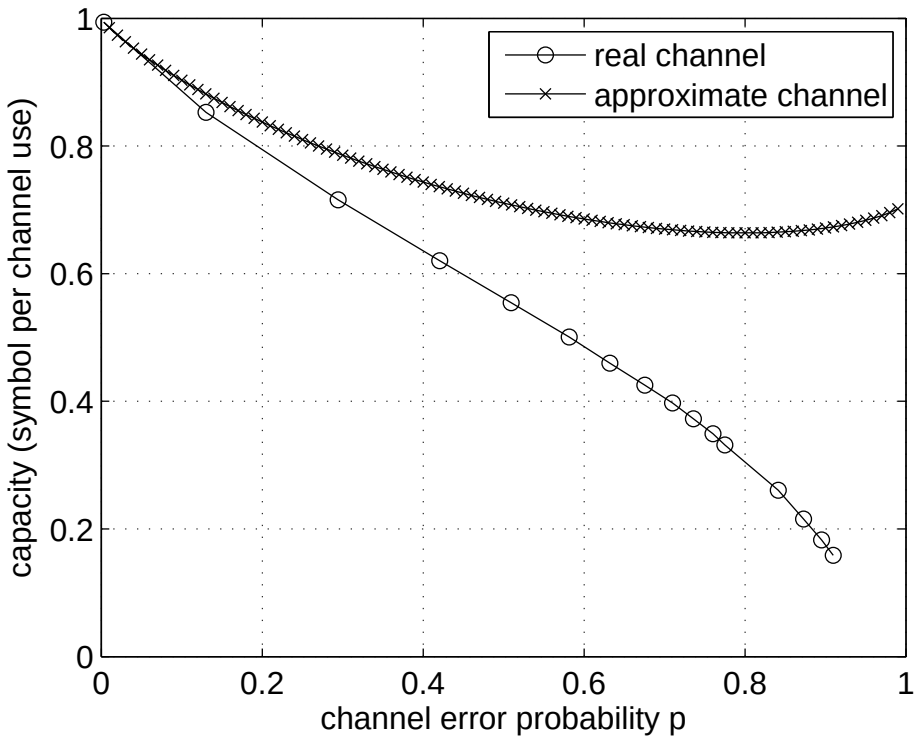

Fig. 29. Capacity of the real channel model and the approximate channel.

C. LDPC Codes Design for Rank Modulation

Before introducing the LDPC codes, we first analyze the maximum likelihood (ML) decoder for the approximate channel.

Lemma 9. *For $p_1 < \frac{1}{2(n-1)}$, the ML decoder for the approximate channel is the minimum Hamming distance decoder as follows*

$$\hat{X} = \arg \min_{X:XH=0} \|X - Y\|_H.$$

Theorem 14. *For any error correcting code, the maximum rate for zero error ML decoding is $\frac{\log(n-1)!}{\log n!}$.*

Proof. The condition for zero-error ML decoding is that there is at most one valid codeword in the $\mathcal{N}_1$ neighborhood of any vector $Y \in \Pi_n^N$. Therefore, the maximum number of codeword can be obtained by the sphere packing argument. Let Q be the maximum number of non-overlapping spheres in $\mathbb{Z}_m^N$. It is obvious to see $Q = \frac{(m)^N}{n^N}$. Since there is at most one codeword in each sphere, therefore the maximum number of codewords is Q and the maximum code rate for zero-error ML decoding is $R^* = \frac{\log M}{\log(m)^N} = \frac{\log(n-1)!}{\log m}$. □

By Stirling's approximation, we have $R^* \approx \frac{\log\left(\frac{(n-1)}{e}\right)^{(n-1)}}{\log\left(\frac{n}{e}\right)^n} = \frac{(n-1)(\log(n-1)-1)}{n(\log n-1)} = 1 - o(\frac{1}{n})$. This shows that there exist high rate codes such that ML decoding has a negligible error rate.

1. Iterative verification-based decoder

We are interested in message-passing decoders for this problem because they have complexity that is linear in N. In particular, we design a *symbol-flipping* algorithm based on verification. The idea is to jointly use the a priori information about the

channel and the assumption that two error messages match with low enough probability as n is large enough. We note that this general idea can be applied to the decoders of both codes over integer rings and finite fields and the density evolution analysis is exactly the same for both codes. This results in decoding algorithms that are similar to both symbol-flipping algorithms, which can be seen as a generalization of the Gallager-A and Gallager-B hard decision decoding algorithms [55], and verification-based decoding algorithm [68]. We also refer these algorithms as the symbol-flipping verification-based (SFVB) algorithms. The event that two incorrect symbols match is also called false verification (FV). Note that when we analyze the SFVB algorithms, we don't consider FV.

Note that the verification-based decoding algorithms can be further classified into node-based (NB) and message-based (MB) algorithms [69]. Therefore, the SFVB decoding algorithms also have two categories, namely, node-based SFVB (NBSFVB) decoding algorithms and message-based SFVB (MBSFVB) decoding algorithms. The messages from the variable nodes to the check nodes (resp. from the check nodes to the variable nodes) are multiplied by the (resp. inverse of the) random entries in the parity-check matrix. For simplicity, we will not describe this explicitly in the following.

a. NBSFVB Decoding Algorithm

In each iteration, we first calculate the check sums of the check nodes. Then we perform the symbol-flipping operation on the variable nodes simultaneously by the following rules.

- If all the check nodes connected to a variable node are satisfied (i.e., the check sum equals 0), then the variable node remains at its current value x.

- Otherwise, the variable node tries to change its value to one of the $\mathcal{N}_1(y)$ neighbors of its received value y.
  - If it finds a value $x' \in \mathcal{N}_1(y)$ which satisfies all the checks, then the variable node flips its value to x': $x \rightarrow x'$.
  - If it cannot find a value which satisfies all the checks, it keeps its value unchanged.

We note that this algorithm cannot be analyzed by density evolution since the two-way messages on the same edge are not independent. This algorithm is a node-based message-passing decoding [69] and one can analyze it using the differential equation approach described in [22] [69]. Also note that this algorithm can be further improved by relaxing the flipping condition. The idea is similar to Gallager-B hard decision decoding algorithm [55] for the BSC and we skip the details here.

b. MBSFVB Decoding Algorithm

By using the a priori channel information and the fact that the size of the channel output alphabet set is large, we propose the MBSFVB decoding algorithm. The decoding rules are described as follows.

- Check Node Operation: Let the variable-to-check messages be $m_0, m_1, \cdots, m_{r-1}$. For a check node of degree r, the check-to-variable message on the j-th edge is $m'_j = m_j - \sum_{k=0}^{r-1} m_k$.
- Variable Node Operation: For a variable node of degree l, let the channel output value be y and check-to-variable messages be $m'_0, m'_1, \cdots, m'_{l-1}$. The variable-

to-check message on the j-th edge is

$$m_j = \begin{cases} m, & \text{if C1} \\ y' & \text{if not(C1) and C2} \\ y & \text{otherwise} \end{cases}$$

where condition C1 represents the event that at least 2 messages from the message set $\{y\} \cup \{m'_0, m'_1, \cdots, m'_{l-1}\} \backslash \{m'_j\}$ match and equal m. Condition C2 represents the event that there exists a value $y' \in \mathcal{N}_1(y)$ such that y' matches at least 1 message in $\{m'_0, m'_1, \cdots, m'_{l-1}\} \backslash \{m'_j\}$.

Based on [69], we derive the DE analysis of MBSFVB under the assumption that the output message is correct whenever condition C1 or C2 holds (i.e., there is no false verification (FV)). Let p_i be the probability that a variable-to-check message is incorrect at iteration i. Let q_i be the probability that a check-to-variable message is incorrect at iteration i. Note that $p_0 = p$ is the channel error probability. The DE equations for the MBSFVB algorithm are

$$q_i = 1 - (1 - p_i)^{r-1}$$

and

$$p_{i+1} = p q_i^{l-1}.$$

Taking the irregularity into account, the DE equations can be written as

$$q_i = 1 - \rho(1 - p_i)$$

and

$$p_{i+1} = p\lambda(q_i),$$

where $\lambda(x)$ and $\rho(x)$ are the variable node and check node degree distributions in the edge perspective.

Remark 26. *The DE recursion of the MBSFVB algorithms is identical to the DE recursion of the BEC. The decoding threshold of the (3,6) ensemble with the MBSFVB algorithm is* $p^*_{MBSFVB} = 0.428$. *The decoding threshold of the (3,50) ensemble with the MBSFVB algorithm is* $p^*_{MBSFVB} = 0.047$.

In the real channel model, error symbols in the $\mathcal{N}_2$-neighborhood may also be corrected by the MBSFVB or NBSFVB algorithm. But the a priori channel information, i.e, the fact that the correct symbol can only lie in the $\mathcal{N}_1$-neighborhood of the received output symbol, helps improve the decoding threshold significantly. A simple modification of the MBSFVB algorithm that does not utilize the channel a priori information actually performs very poorly. For example, let us define the MBSFVB algorithm as follows.

- Check Node Operation: The check node operation is the same as with the MBSFVB algorithm.

- Variable Node Operation: For a variable node of degree l, let the channel output value be y and check-to-variable messages be m'_0, m'_1, $\cdots$, m'_{l-1}. The variable-to-check message on the j-th edge is

$$m_j = \begin{cases} m & \text{if } m'_k = m, \text{ for all } k \neq j; \\ y & \text{otherwise.} \end{cases}$$

We derived the DE analysis, and the result shows that the decoding threshold is 0.092 for the (3,6) ensemble. Due to the poor performance, we omit the details about the DE analysis here.

As mentioned in previous section, when the codes are defined over $GF(q)$ which is embedded into $\mathbb{Z}_m$, sometimes the output permutation y of the rank modulation demodulator is not in $\mathcal{L}$, hence y cannot be mapped back to $GF(q)$. In this case, we know that the correct permutation falls in $\mathcal{N}_1(y)$ with high probability. We assign an arbitrary element in $Z(y) \triangleq \{z | z \in GF(q), \wp^{-1}(z) \in \mathcal{N}_1(y), \wp^{-1}(z) \in \mathcal{L}\}$ to this coded symbol and we assign the rest elements in $Z(y)$ as its decoding neighborhood. Then we can still apply the SFVB decoding algorithms described above. We note that this results in slightly better error correcting performance than that predicted by the DE analysis because there is less ambiguity on those symbols on the boundary of $\mathcal{L}$ given the channel observation. Since the error correcting capability is better if the transmitted symbol is on the boundary of $\mathcal{L}$, the analysis of the average performance can not be simplified to the analysis of the all-1 codeword (where each permutation is the identity permutation). We only use the DE result as an upper bound of the decoding threshold.

In our DE anlaysis, we assume that there is no FV. During our simulations of codes over $\mathbb{Z}_m$, we discovered that the probability of FV is not negligible for moderate n (e.g., $n \leq 8$). The problem is not that the alphabet size is too small (e.g., $8! = 40320$ is large enough based on previous work). Instead, it appears that the issue of FV is more complicated for codes over integer rings. The reason is that multiplication with random edge weights maps some incorrect symbols (e.g., $(n-1)! \in \mathbb{Z}_m$) to sets whose size are significantly smaller than m or even $\phi(m)$. Therefore, the probability that two incorrect symbols match is too large to be ignored. Unfortunately, we do not yet have a good analysis of the probability of FV. Still, the simulation results show that, for $n \geq 8$, the probability of FV is low enough that the algorithm is of interest.

D. Simulation Results

We evaluate the capacity of the real channel and the approximate channel. The result is shown in Fig. 29. From the result we can see that the approximate channel and the real channel have approximately the same capacity when $p < 0.05$.

We simulate the MBSFVB decoder, or the verification-based (VB) decoder, with the $(3, 50)$ regular ensemble with rate 0.94 and threshold $p^*_{VB} = 0.047$. The reason we choose the $(3, 50)$ ensemble is that the approximate channel and the real channel model have approximately the same capacity around the decoding threshold. The channel used is the real channel model. Note that the threshold $p^*_{VB} = 0.047$ implies the threshold $\sigma^*_{VB} = 0.315$ in the real channel model. The codes are generated without 4 cycles. Each coded symbol is stored by 5 to 8 cells with rank modulation. We simulate both codes over integer rings and codes over finite fields with different alphabets. We also simulate the VB decoder without FV by artificially avoiding FV's. To see how the numerical simulation matches the DE analysis, we choose the block length to be 10^5 and the maximum number of decoding iterations to be 100. Each point is the average of up to 10^8 trials. And the results are shown in Fig. 30. It can be seen that the simulation matches the DE analysis very well.

To be fair, we also compare the VB decoder with the full belief propagation (BP) decoder, where each message is a probability mass function (pmf) over $\mathbb{Z}_m$. For the check node, the check-to-variable message is calculated by the convolution of all other variable-to-check pmf's. For the variable node, the variable-to-check message is calculated by normalizing the product of the channel pmf and all other check-to-variable pmf's. The complexity of this method in the probability domain is $O(Nm^2)$ per iteration without optimization. The log domain FFT-based decoder has complexity $O(Nm \log(m))$ per iteration [70, 71].

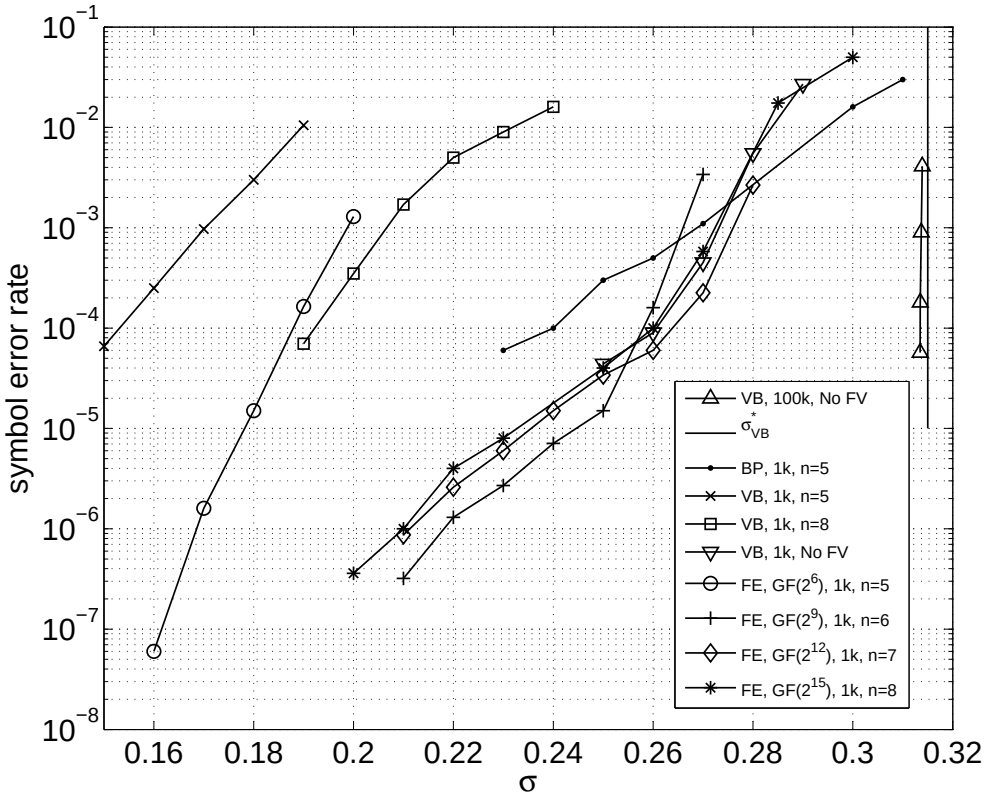

Fig. 30. Simulation results comparing the VB decoder with the full BP decoder using the real channel model.

We compare the VB decoder with the full BP decoder in Fig. 30. We simulate codes from the (3,50) ensemble by the VB decoder and the full BP decoder. We simulate codes over integer rings $\mathbb{Z}_m$ and finite fields which are embedded in $\mathbb{Z}_m$. We choose the block length to be 1000 and $n = 5$ to 8 and without FV for the VB decoder and $n = 5$ for the full decoder due to high complexity of the full BP decoder and the reason of fair comparison. The real channel model is used. The x-axis is the standard deviation σ of the i.i.d. Gaussian noise. From the results we can see the VB decoder without FV's has similar performance with the full decoder. The performance loss of the codes over $\mathbb{Z}_m$ with $n = 5$ and $n = 8$ is due to the probability of FV's. While this loss can be compensated by using the codes over $GF(q)$ which is embedded into the rings. This improves the performance significantly with a slight rate loss.

CHAPTER VI

CONCLUSIONS

In this dissertation, we study LDPC codes over large alphabets and their applications on compressed sensing and flash memory. The analysis of decoding threshold for different decoding algorithms are derived. During the analysis of the decoding threshold, we prove that LDPC codes are capacity-achieving over the q-ary symmetric channel. We also propose the list-message-passing decoding algorithm which provides a smooth trade-off between the computational complexity and the error correcting capability. The application of LDPC codes over large alphabets to compressed sensing systems shows significant improvement of the system performance. We analyze the performance and compare our LDPC codes based compressed sensing system to many other existing algorithms. We also discuss LDPC codes design for flash memory and analyze the decoding threshold.

REFERENCES

[1] C. E. Shannon, "A mathematical theory of communication," *Bell System Technical Journal*, vol. 27, pp. 379–423 and pp. 623–656, Jul. and Oct. 1948.

[2] C. Berrou and A. Glavieux and P. Thitimajshima, " Near shannon limit error-correcting coding and decoding: turbo-codes," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Geneva, Switzerland, May 1993, pp. 1064–1070.

[3] C. Berrou and A. Glavieux, "Near optimum error correcting coding and decoding: turbo-codes," *IEEE Trans. Commun.*, vol. 44, no. 10, pp. 1261–1271, Oct. 1996.

[4] R. G. Gallager, "Low-density parity-check codes," *IRE Trans. Inf. Theory*, vol. 18, no. 1, pp. 21–28, Jan. 1962.

[5] D. MacKay and R. Neal, "Good codes based on very sparse matrices," *Lecture Notes in Computer Science*, vol. 1025, pp. 100–111, 1995.

[6] T. Richardson, M. Shokrollahi, and R. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 619–637, Feb. 2001.

[7] A. Shokrollahi, "Capacity-approaching codes on the q-ary symmetric channel for large q," in *Proc. IEEE Inform. Theory Workshop*, San Antonio, TX, Oct. 2004, pp. 204–208.

[8] C. Weidmann, "Coding for the q-ary symmetric channel with moderate q," in *Proc. IEEE Int. Symp. Information Theory*, Toronto, Canada, Jul. 2008, pp. 2156–2159.

[9] G. Lechner and C. Weidmann, "Optimization of binary ldpc codes for the q-ary symmetric channel with moderate q," in *Proc. 5th International Symposium on Turbo Codes and Related Topics*, Lausanne, Switzerland, Sept., 2008, pp. 221–224.

[10] M. Luby and M. Mitzenmacher, "Verification-based decoding for packet-based low-density parity-check codes," *IEEE Trans. Inf. Theory*, vol. 51,no. 1, pp. 120–127, Jan. 2005.

[11] J. Metzner, "Majority-logic-like decoding of vector symbols," *IEEE Trans. Commun.*, vol. 44, no. 10, pp. 1227–1230, Oct. 1996.

[12] J. Metzner, "Majority-logic-like vector symbol decoding with alternative symbol value lists," *IEEE Trans. Commun.*, vol. 48, no. 12, pp. 2005–2013, Dec. 2000.

[13] D. Bleichenbacher, A. Kiyayias, and M. Yung, "Decoding of interleaved Reed-Solomon codes over noisy data," in *Proc. of ICALP*, pp. 97–108, 2003.

[14] A. Shokrollahi and W. Wang, "Low-density parity-check codes with rates very close to the capacity of the q-ary symmetric channel for large q," in *Proc. IEEE Int. Symp. Inf. Theory*, Chicago, IL, Jun. 2004, pp. 275.

[15] A. Shokrollahi and W. Wang, "Low-density parity-check codes with rates very close to the capacity of the q-ary symmetric channel for large q." Unpublished extended abstract, 2004.

[16] E. Arikan, "Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels," *IEEE Trans. Inf. Theory*, vol. 55, no. 7, pp. 3051–3073, 2009.

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

[18] V. Guruswami and P. Indyk, "Linear time encodable and list decodable codes," in *Proc. 35th Annual ACM Symp. on Theory of Comp.*, San Diego, CA, 2003, pp. 126–135.

[19] D. E. Knuth, "Big omicron and big omega and big theta," In *SICACT News*, vol. 8, no. 2, pp. 18–24, Apr. 1976.

[20] T. J. Richardson and R. L. Urbanke, *Modern Coding Theory*. Cambridge University Press., New York, USA, 2008.

[21] R. Storn and K. Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *J. Global Optim.*, vol. 11, no. 4, pp. 341–359, 1997.

[22] M. Luby, M. Mitzenmacher, M. Shokrollahi, and D. Spielman, "Efficient erasure correcting codes," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 569–584, Feb. 2001.

[23] N. Wormald, "Differential equations for random processes and random graphs," *Annals of Applied Probability*, vol. 5, no. 4, pp. 1217–1235, 1995.

[24] N. Wormald, "The differential equation method for random graph processes and greedy algorithms," *Lecture Notes in Summer School on Randomized Algorithms*, Antonin, Poland, 1997.

[25] F. Zhang and H. D. Pfister, "Verification decoding of high-rate ldpc codes with applications in compressed sensing." submitted to *IEEE Trans. on Inf. Theory*

also available in Arxiv preprint cs.IT/0903.2232v3, 2009.

[26] A. Kavcic, X. Ma, and M. Mitzenmacher, "Binary intersymbol interference channels: Gallager codes, density evolution, and code performance bounds," *IEEE Trans. Inf. Theory*, vol. 49, no. 7, pp. 1636–1652, Jul. 2003.

[27] S. Chen, D. Donoho, and M. Saunders, "Atomic decomposition by basis pursuit," *SIAM J. Sci. Comp.*, vol. 20, no. 1, pp. 33–61, 1998.

[28] D. L. Donoho, "Compressed sensing," *IEEE Trans. Inf. Theory*, vol. 52, no. 4, pp. 1289–1306, 2006.

[29] E. J. Candès, J. Romberg, and T. Tao, "Robust uncertainty principles: exact signal reconstruction from highly incomplete frequency information," *IEEE Trans. Inf. Theory*, vol. 52, no. 2, pp. 489–509, 2006.

[30] A. C. Gilbert, M. J. Strauss, J. A. Tropp, and R. Vershynin, "One sketch for all: fast algorithms for compressed sensing," In *in Proceedings of the ACM Symposium on the Theory of Computing (STOC 2007)*, 2007.

[31] A. Cohen, W. Dahmen, and R. DeVore, "Compressed sensing and best k-term approximation," *IGPM Report, RWTH-Aachen*, Jul. 2006.

[32] W. Johnson and J. Lindenstrauss, "Extensions of lipschitz maps into hilbert space," *Contemp. Math.*, vol. 26, pp. 189–206, 1984.

[33] E. D. Gluskin, "Norms of random matrices and widths of finite-dimensional sets," *Math. USSR Sbornik*, vol. 48, pp. 173–182, 1984.

[34] S. Sarvotham, D. Baron, and R. G. Baraniuk, "Sudocodes–fast measurement and reconstruction of sparse signals," in *Proc. IEEE Int. Symp. Information Theory*, Seattle, WA, Jul. 2006, pp. 2804–2808.

[35] W. Xu and B. Hassibi, "Efficient compressive sensing with deterministic guarantees using expander graphs," in *Proc. IEEE Inform. Theory Workshop*, Lake Tahoe, CA, Sept. 2007, pp. 414–419.

[36] F. Zhang and H. D. Pfister, "Compressed sensing and linear codes over real numbers," in *Proc. 2008 Workshop on Inform. Theory and Appl.*, UCSD, La Jolla, CA, Feb. 2008, pp. 558–561.

[37] S. Sarvotham, D. Baron, and R. Baraniuk, "Compressed sensing reconstruction via belief propagation," *Rice University, Tech. Rep.* ECE-06-01, Jul. 2006.

[38] W. Dai and O. Milenkovic, "Weighted superimposed codes and constrained integer compressed sensing," 2008, submitted to *IEEE Trans. Inf. Theory* also available in Arxiv preprint cs.IT/0806.2682v1.

[39] D. J. C. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Trans. Inf. Theory*, vol. 45, no. 2, pp. 399–431, March 1999.

[40] T. Richardson, M. A. Shokrollahi, and R. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 619–637, Feb. 2001.

[41] C. Di, D. Proietti, E. Telatar, T. J. Richardson, and R. Urbanke, "Finite-length analysis of low-density parity-check codes on the binary erasure channel," *IEEE Trans. Inf. Theory*, vol. 48, no. 6, pp. 1570–1579, Jun. 2002.

[42] A. Orlitsky, K. Viswanathan, and J. Zhang, "Stopping set distribution of LDPC code ensembles," *IEEE Trans. Inf. Theory*, vol. 51, no. 3, pp. 929–953, 2005.

[43] F. Zhang and H. D. Pfister, "List-message passing achieves capacity on the q-ary

symmetric channel for large q," in *Proc. IEEE Global Telecom. Conf.*, Washington, DC, Nov. 2007, pp. 283–287.

[44] S. ten Brink, "Convergence behavior of iteratively decoded parallel concatenated codes." *IEEE Trans. Inf. Theory*, vol. 49, no. 10, pp. 1727–1737, Oct. 2001.

[45] E. J. Candès and T. Tao, "Decoding by linear programming," *IEEE Trans. Inf. Theory*, vol. 51, no. 12, pp. 4203–4215, 2005.

[46] J. A. Tropp and A. C. Gilbert, "Signal recovery from random measurements via orthogonal matching pursuit," *IEEE Trans. Inf. Theory*, vol. 53, no. 12, pp. 4655–4666, 2007.

[47] M. A. Shokrollahi, "New sequences of linear time erasure codes approaching the channel capacity," In *Applicable Algebra in Eng., Commun. Comp.*, 1999, pp. 65–76.

[48] M. A. Khajehnejad, A. G. Dimakis, W. Xu, and B. Hassibi, "Sparse recovery of positive signals with minimal expansion," 2009, available in Arxiv preprint cs.IT/0902.4045v1.

[49] J. K. Wolf, "Redundancy, the discrete Fourier transform, and impulse noise cancellation," *IEEE Trans. Commun.*, vol. 31, no. 3, pp. 458–461, March 1983.

[50] M. G. Luby, M. Mitzenmacher, and M. A. Shokrollahi, "Practical loss-resilient codes," in *Proc. 29th Annu. ACM Symp. Theory of Computing*, 1997, pp. 150–159.

[51] W. Dai and O. Milenkovic, "Subspace pursuit for compressive sensing signal reconstruction," 2008. [Online]. Available: http://arxiv.org/abs/0803.0811

[52] N. Deanna and V. Roman, "Uniform uncertainty principle and signal recovery via regularized orthogonal matching pursuit," *Foundations of Computational Mathematics*, vol. 9, no. 3, pp. 317–334, June 2009.

[53] R. Chartrand and W. Yin, "Iteratively reweighted algorithms for compressive sensing," In *Acoustics, Speech and Signal Processing, 2008. ICASSP 2008. IEEE International Conference on*, 2008, pp. 3869–108.

[54] R. Corless, G. Gonnet, D. Hare, D. Jeffrey, and D. Knuth, "On the Lambert W function," *Advances in Computational mathematics*, vol. 5, no. 1, pp. 329–359, 1996.

[55] R. G. Gallager, *Low-Density Parity-Check Codes*. Cambridge, MA, USA: The M.I.T. Press, 1963.

[56] A. Fiat and A. Shamir. Generalized write-once memories. *IEEE Trans. Inf. Theory*, vol. 30, no. 3, pp. 470–480, May 1984.

[57] R.L. Rivest and A. Shamir. How to reuse a write-once memory. *Information and Control*, vol. 55, pp. 227–231, 1984.

[58] C. Heegard and A. El Gamal. On the capacity of computer memory with defects. *IEEE Trans. Inf. Theory*, vol. 29, no. 5, pp. 731–739, Sept. 1983.

[59] J. Ziv J. K. Wolf, A.D. Wyner and J. Korner. Coding for a write-once memory. *AT&T Bell Laboratories Technical Journal*, vol. 63, no. 6, pp. 1089–1112, 1984.

[60] A. Jiang and J Bruck. Joint coding for flash memory storage. in *Proc. IEEE Int. Symp. Information Theory*, Toronto, Canada, Jul. 2008, pp. 1741–1745.

[61] M. Raab and A. Steger. "Balls into bins" - a simple and tight analysis. in *Proc. the Second International Workshop on Randomization and Approximation Techniques in Computer Science*, 1998, vol. 1518, pp. 159–170.

[62] A. Jiang, M. Langberg, M. Schwartz, and J. Bruck. Universal rewriting in constrained memories. in *Proc. IEEE Int. Symp. Information Theory*, South Korea, Jun. 2009, pp. 1219-1223.

[63] A. Barg and A. Mazumdar, "Codes in permutations and error correction for rank modulation," Available as Arxiv preprint cs.IT/0908.4094, 2009.

[64] R. P. Brent and B. D. McKay, "Determinants and ranks of random matrices over Z_m," In *Discrete Mathematics*, vol. 66, pp. 35–49, 1987.

[65] A. Jiang, R. Mateescu, M. Schwartz and J. Bruck, "Rank modulation for flash memories," *IEEE Trans. Inf. Theory*, vol. 55, no. 6, pp. 2659–2673, 2009.

[66] A. Jiang, M. Schwartz and J. Bruck, "Error-correcting codes for rank modulation," in *Proc. IEEE Int. Symp. Information Theory*,Toronto, Canada, Jul. 2008, pp. 1736–1740.

[67] Z. Wang, A. Jiang and J. Bruck, "On the capacity of bounded rank modulation for flash memories," in *Proc. IEEE Int. Symp. Information Theory*, South Korea, Jun. 2009, pp. 1234–1238.

[68] M. Luby and M. Mitzenmacher, "Verification-based decoding for packet-based low-density parity-check codes," In *IEEE Trans. Inf. Theory*, vol. 51, no. 1, pp. 120–127, 2005.

[69] F. Zhang and H. D. Pfister, "List-message passing achieves capacity on the q-ary symmetric channel for large q," submitted to *IEEE Trans. Inf. Theory*. Available

as Arxiv preprint cs.IT/0806.3243, 2008.

[70] H. Song and J. R. Cruz, "Reduced-complexity decoding of Q-ary LDPC codes for magnetic recording," *IEEE Trans. Magn.*, vol. 39, no. 3, pp. 1081–1087, 2003.

[71] M. Davey and D. J. C. MacKay, "Low density parity check codes over GF(q)," In *IEEE Commun. Lett.*, vol. 2, no. 6, pp. 165–167, Jun. 1998.

[72] Y. Azar, A. Z. Brodery, A. R. Karlinz, and E. Upfal. Balanced allocations. in *Proc. 26-th annual ACM symposium on Theory of computing*, pp. 593–602, 1994.

[73] H. Finucane, Z. Liu, and M. Mitzenmacher. Designing floating codes for expected performance. in *Proc. 46th Annual Allerton Conf. on Commun., Control, and Comp.*, Monticello, IL, September 2008, pp 968–978.

[74] A. Jiang. On the generalization of error-correcting WOM codes. in *Proc. IEEE Int. Symp. Information Theory*, Nice, France, Jun. 2007, pp. 1391–1395.

[75] A. Jiang, V. Bohossian, and J. Bruck. Floating codes for joint information storage in write asymmetric memories. in *Proc. IEEE Int. Symp. Information Theory*, Nice, France, Jun. 2007, pp. 1166–1170.

[76] M. D. Mitzenmacher. The power of two choices in randomized load balancing. *IEEE Tran. Par. Dist. Sys.*, vol. 12, no. 10, pp. 1094–1104, 1996.

[77] B. Vocking. How asymmetry helps load balancing. in *Foundations of Computer Science, 1999. 40th Annual Symposium on*, pp. 131–141, 1999.

[78] E. Yaakobi, A. Vardy, P. H. Siegel, , and J. K. Wolf. Multidimensional flash codes. in *Proc. 46th Annual Allerton Conf. on Commun., Control, and Comp.*, Monticello, IL, Sep. 2008, pp. 392–399.

[79] F. Zhang and H. D. Pfister. Modulation codes for flash memory based on load-balancing theory. in *Proc. 47th Annual Allerton Conf. on Commun., Control, and Comp.*, Monticello, IL, 2009, pp. 1039–1046.

APPENDIX A

Proof of theorem 1

Proof. Given $p\lambda(1-\rho(1-x)) < x$ for $x \in (0,1]$, we start by showing that both x_i and y_i go to zero as i goes to infinity. To do this, we let $\alpha = \sup_{x\in(0,1)} \frac{1}{x} p\lambda(1-\rho(1-x))$ and note that $\alpha < 1$ because $p < p^*$. It is also easy to see that, starting from $x_0 = 1$, we have $x_i \leq \alpha^i$ and $x_i \to 0$. Next, we rewrite (2.3) as

$$\begin{aligned} y_{i+1} &= \frac{1}{p} x_{i+1} + p\left(\rho(1-x_i) - \rho(1-y_i)\right)\lambda'(1-\rho(1-x_i)) \\ &\overset{(a)}{\leq} \frac{1}{p}\alpha^{i+1} + p\left(1-\rho'(1)\alpha^i - \rho(1-y_i)\right)\left(\lambda_2 + O(\alpha^i)\right) \\ &\overset{(b)}{\leq} \frac{1}{p}\alpha^{i+1} + p\lambda(1-\rho(1-y_i))\left(1+O(\alpha^i)\right) \\ &\overset{(c)}{\leq} \frac{1}{p}\alpha^{i+1} + \alpha y_i\left(1+O(\alpha^i)\right), \end{aligned}$$

where (a) follows from $\rho(1-x) \leq 1-\rho'(1)x$, (b) follows from $\lambda_2(1-\rho(1-y)) \leq \lambda(1-\rho(1-y))$, and (c) follows from $p\lambda(1-\rho(1-y)) \leq \alpha y$. It is easy to verify that $y_{i+1} < y_i$ as long as $y_i > \frac{\alpha^{i+1}}{p(1-\alpha(1+O(\alpha^i)))}$. Therefore, we find that $y_i \to 0$ because the recursion does not have any positive fixed points as $i \to \infty$. Moreover, one can show that y_i eventually decreases exponentially at a rate arbitrarily close to α.

Note that the decoding error comes from two reasons, one is the event that message is not verified and the other one is the event that the message is falsely verified. Next, we are going to show that the actual performance of a code converges to the ensemble average exponentially which means almost every code in a capacity-achieving ensemble has capacity-achieving performance. Note that the concentration

effect and the decay of FV probability hold regardless whether the error probability of the decoder converges to zero or not.

We can prove that the performance of a particular code converges to the threshold which is the average performance of a tree-like ensemble in a similar way in [17], where the average is over the graph ensemble $(\lambda(x), \rho(x))$ and all the channel inputs. There are two difference between our scenario and [17], i.e, our algorithm passes a list of values with unbounded list size, the second difference is the graph may be irregular in our case. Here we only mention the brief procedure of the proof. We can let $Z^{(l)}/E$ denote the fraction of *unverified* messages at the l-th iteration, where E is the number of edges in the graph. Note that $Z^{(l)}$ denotes the number of *incorrect* and *erasure* messages in [17]. Following [17], we can break the failure the probability into a tree-like neighborhood term and a Martingale concentration term to get

$$\Pr\left(\left|\frac{Z^{(l)}(\mathbf{s})}{E} - y_l\right| \geq \epsilon\right) \leq$$

$$\Pr\left(\left|\frac{Z^{(l)}(\mathbf{s})}{E} - \frac{\mathrm{E}\left[Z^{(l)}(\mathbf{s})\right]}{E}\right| \geq \epsilon/2\right) +$$

$$\Pr\left(\left|\frac{\mathrm{E}\left[Z^{(l)}(\mathbf{s})\right]}{E} - y_l\right| \geq \epsilon/2\right)$$

where $\mathbf{s}$ is an arbitrary codeword chosen from ensemble (λ, ρ), $Z^{(l)}(\mathbf{s})$ is the random variable that denotes the number of unverified variable-to-check messages after l decoding iterations. E is the number of edges in the graph. This means that the concentration bound consists two parts: concentration from a particular code to the ensemble with cycles and concentration from a ensemble with cycles to the tree-like ensemble. Notice that the proof of the later concentration and the proof of the probability of a tree-like neighborhood are not limited to the specific decoding algorithm and the definition of $Z^{(l)}$, the proof is omitted here. By forming a Doob's

martingale on the edge-exposure and applying Azuma's inequality, we can prove the concentration from a particular code to the ensemble in the same manner as [17]. In our scenario, the proof of bounded difference of the martingale, the right hand side of [17, Eq. (16)] should be the cardinality of depth $2l$ directed neighbor of e, $\frac{\left|\vec{\mathcal{N}}_e^{(2l)}\right|}{2}$. The right hand side of [17, Eq. (17)] should be $4\left|\vec{\mathcal{N}}_e^{(2l)}\right|$. The β in applying Azuma's inequality is $\sum_{k=1}^{E}\left(4\left|\vec{\mathcal{N}}_e^{(2l)}\right|\right)^2+\sum_{k=1}^{n}\left(4\left|\vec{\mathcal{N}}_e^{(2l)}\right|\right)^2$. So far, we prove that, for an arbitrary small constant $\epsilon/2$, there exist positive numbers β and γ, such that if $n>\frac{2\gamma}{\epsilon}$, then

$$\Pr\left(\left|\frac{Z^{(l)}(\mathbf{s})}{E}-y_l\right|\geq\epsilon\right)\leq e^{-\beta\epsilon^2 n}$$

Note that the similar proof can be found in [17] (the proof of Theorem 2) and [26] (the proof of Theorem 1). Note that [17] proves for the regular code ensemble and [26] extends the proof to the irregular code ensemble. So, for an arbitrary code $\mathbf{s}$ and an arbitrary small quantity ϵ, the fraction of unverified message is less than $\epsilon/2$ as n goes to infinity.

In [17] and [26], it is proved that, when a code graph is chosen uniformly at random from all possible graphs with degree distribution pair $(\lambda(x),\rho(x))$,

$$\Pr\left(\text{neighborhood of depth } 2l \text{ is not tree-like}\right)\leq\frac{\gamma}{n}$$

where γ is a constant independent of n. So, given ϵ, we can choose n large enough such that the number of variable nodes which are involved in cycles of length less than $2l$ is less than $n\epsilon/2$ with probability arbitrarily close to one as n goes to infinity. So the probability of error caused by type-II FV's is upper bounded by $\epsilon/2$ (for the notation of type-I and type-II FV, please refer to Section 2). Here, we don't consider the type-I FV's because the probability of type-I FV's can be forced arbitrarily close to zero by choosing a large enough q. □

APPENDIX B

Proof of Theorem 2

All verifications that occur in LM1-NB also occur in LM1-MB and vice versa. So LM1-NB and LM1-MB are equivalent.

Proof. The operations of LM1-MB and LM1-NB are different because they have different verification rules (see Table. I). We can prove they are equivalent by showing the verification occurs in LM1-MB also occurs in LM1-NB and vice versa, but in different decoding steps. Let's first look at the check node when the summation of all messages equals to zero but there are more than 1 messages are unverified. In this case, LM1-NB will verify all the messages. In LM1-MB none of them will be verified but all the values will be correct. In the following iteration, all these messages will be verified on their variable nodes. Notice that this is the only case verification occurs in LM1-NB but not in LM1-MB. So verification in LM1-NB also occurs in LM1-MB. Let's then look at the variable node when any incoming message is correct and the channel value is correct. In LM1-MB, the outgoing message will be verified. In LM1-NB the message will be correct but not verified. Notice that the incoming is correct means all the other messages are correct at the check node, so the unverified correct message will be verified in the next step on check node. Notice that this is the only case verification occurs in LM1-MB but not in LM1-NB. So verification in LM1-MB also occurs in LM1-NB. □

APPENDIX C

Proof of Lemma 1 Let y be the received symbol sequence assuming the all-zero codeword was transmitted. Let u be any codeword with exactly k non-zero symbols. It is easy to verify that the probability that ML decoder chooses u over the all-zero codeword is given by

$$p_{2,k} = \sum_{j=0}^{k} \sum_{i=0}^{j} \binom{k}{i, j, k-i-j} (1-p)^i \left(\frac{p}{q-1}\right)^j \left(\frac{p(q-2)}{q-1}\right)^{k-i-j}.$$

Using the multinomial theorem, it is also easy to verify that

$$\begin{aligned} A(x) &= \left((1-p) + \frac{p}{q-1}x^2 + \frac{p(q-2)}{q-1}x\right)^k \\ &= \sum_{j=0}^{k} \sum_{i=0}^{k-j} \binom{k}{i, j, k-i-j} (1-p)^i \left(\frac{p}{q-1}\right)^j \left(\frac{p(q-2)}{q-1}\right)^{k-i-j} x^{k-i+j} \\ &\triangleq \sum_{l=0}^{2k} A_l x^l, \end{aligned}$$

where A_l is the coefficient of x^l in $A(x)$. Finally, we observe that $p_{2,k} = \sum_{l=k}^{2k} A_l$ is simply an unweighted sum of a subset of terms in $A(x)$ (namely, those where $k - i + j \geq k$).

This implies that

$$x^k p_{2,k} = \sum_{l=k}^{2k} A_l x^k \leq A(x)$$

for any $x \geq 1$. Therefore, we can compute the Chernoff-type bound

$$p_{2,k} \leq \inf_{x \geq 1} x^{-k} A(x).$$

By taking derivative of $x^{-k}A(x)$ over x and setting it to zero, we arrive at the bound

$$p_{2,k} \leq \left(p\frac{q-2}{q-1} + \sqrt{\frac{4p(1-p)}{q-1}} \right)^k .$$

APPENDIX D

Proof of Lemma 2

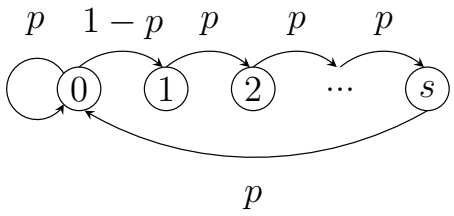

Fig. 31. Finite-state machine for Lemma 2.

Proof. An unverification event occurs on a degree-2 cycle of length-k when there is at most one correct variable node in any adjacent set of $s+1$ nodes. Let the set of all error patterns (i.e., 0 means correct and 1 means error) of length-k which satisfy the UV condition be $\Phi(s,p,k) \subseteq \{0,1\}^k$. Using the Hamming weight $w(z)$, of an error pattern as z, to count the number of errors, we can write the probability of UV as

$$\phi(s,p,k) = \sum_{z \in \Phi(s,p,k)} p^{w(z)}(1-p)^{k-w(z)}.$$

This expression can be evaluated using the transfer matrix method to enumerate all weighted walks through a particular digraph. If we walk through the nodes along the cycle by picking an arbitrary node as the starting node, the UV constraint can be seen as k-steps of a particular finite-state machine. Since we are walking on a cycle, the initial state must equal to the final state.

The finite-state machine, which is shown in Fig. 31, has $s+1$ states $\{0, 1, \ldots, s\}$. Let state 0 be the state where we are free to choose either a correct or incorrect symbol (i.e., the previous s symbols are all incorrect). This state has a self-loop associated with the next symbol also being incorrect. Let state $i > 0$ be the state where the past i values consist of one correct symbol followed by $i - 1$ incorrect symbols. Notice that only state 0 may generate correct symbols. By defining the transfer matrix with (2.18), the probability that the UV condition holds is therefore $\phi(s, p, k) = \mathrm{Tr}\left(B^k(p)\right)$.

□

APPENDIX E

Proof of Proposition 1

Proof. Starting with the convergence condition $\lambda\left(1-e^{-\overline{\alpha}_j jx}\right) \leq x$ for $x \in (0,1]$, we first solve for $\overline{\alpha}_j$ to get

$$\overline{\alpha}_j = \inf_{x\in(0,1)} -\frac{1}{jx}\ln\left(1-\lambda^{-1}(x)\right). \tag{E.1}$$

Next, we substitute $x = \lambda\left(1-e^{-y}\right)$ and simplify to get

$$\overline{\alpha}_j = \inf_{y\in(0,\infty)} \frac{y}{j\lambda\left(1-e^{-y}\right)}. \tag{E.2}$$

For $j \geq 3$, this function is unbounded as $y \to 0$ or $y \to \infty$, so the minimum must occur at an interior critical point y^*. Choosing $\lambda(x) = x^{j-1}$ and setting the derivative w.r.t. y to zero gives

$$\frac{j\left(1-e^{-y^*}\right)^{j-1} - j(j-1)y^*\left(1-e^{-y^*}\right)^{j-2}e^{-y^*}}{j^2\left(1-e^{-y^*}\right)^{2j-2}} = 0 \tag{E.3}$$

Canceling terms and simplifying the numerator gives $1-e^{-y^*}-(j-1)y^*e^{-y^*}=0$, which can be rewritten as $e^{y^*} = (j-1)y^*+1$. Ignoring $y^*=0$, this implies that y^* is given by the unique intersection of e^y and $(j-1)y+1$ for $y>0$. That intersection point can be written in closed form using the non-principal real branch of the Lambert W-function [54], $W_{-1}(x)$, and is given by, for $j \geq 2$,

$$y_j^* = -\frac{1}{j-1}\left(1+(j-1)W_{-1}\left(-\frac{1}{j-1}e^{-1/(j-1)}\right)\right). \tag{E.4}$$

Using this, the α-threshold for j-regular ensembles is given by $\overline{\alpha}_j = \frac{1}{j}y_j^*\left(1-e^{-y_j^*}\right)^{1-j}$. For $j=2$, the minimum occurs as $y_2^* \to 0$ and the limit gives $\overline{\alpha}_2 = \frac{1}{2}$. □

APPENDIX F

Proof of Theorem 4

Before proving the theorem, we first prove the following lemma.

Lemma .1. *Let X be a Poisson random variable with mean λ. The tail probability can be bounded by*

$$\Pr(X \geq x_0) \leq e^{-\lambda} \left(\frac{x_0}{\lambda}\right)^{-x_0} e^{x_0} \tag{F.1}$$

Proof. The moment generating function of Poisson distribution is $M_X(s) = e^{\lambda(e^s-1)}$. The Chernoff bound shows

$$\Pr(X \geq x_0) \leq e^{-sx_0} M_X(s)$$

for $s \geq 0$. Setting the derivative of the RHS over s to zero to minimize the RHS (since the second derivative is positive), we have the RHS of (F.1). Note that the RHS of (F.1) decays faster than e^{-Ax_0} for any constant A. □

Proof of Theorem 2. To show the concentration for our case, we modify the proof as following. In the first iteration, we reduce the graph to a smaller one by removing the edges which are not erased. Let $\delta = \frac{\alpha j}{k}$ be the channel erasure probability and $k = n^\omega$, the number of check nodes concentrates around $\frac{nj}{k} = jn^{1-\omega}$ and the number of variable nodes concentrates around $n' = n\delta = \alpha j n^{1-\omega}$. The probability that a check node has degree t after removal is $\binom{k}{t}\delta^t(1-\delta)^{k-t}$ which converges to Poisson distribution as $k \to \infty$. So the edge degree distribution of check node is $\rho(x) = \sum_t \frac{(\alpha j)^t}{t!} e^{-\alpha j} x^{t-1}$, the average check node degree is αj, and the edge degree distribution of the variable node is $\lambda(x) = x^{j-1}$.

By the similar edge-revealing argument [40], we can model the graph-uncovering process as a martingale and apply the Azuma's inequality, we can bound the probability that the performance of a particular code deviates the cycle-free case by more than ϵ,

$$\Pr\left\{\left|\frac{Z^{(\ell)}}{n'_e} - p'^{(\ell)}\right| \geq \epsilon\right\} \leq 2e^{-\frac{(n'j\epsilon)^2}{2\sum_{i=1}^{n'(1+j)}\alpha_i^2}} \tag{F.2}$$

where α_i is an upper-bound of the difference of the numbers of erasure messages between the graphs obtained by revealing the $(i-1)$-th and the i-th edge. n'_e is the number of edges in the *residual graph* which concentrates around $\alpha j^2 n^{1-\omega}$. $p'^{(\ell)}$ is the fraction of erasures of the residual graph given by DE. It is easy to see $n'p'^{(\ell)} = np^{(\ell)}$ where $p^{(\ell)}$ is the DE result of the original graph. For simplicity, we can pick α_i to be $2T_{\ell,i}$ where $T_{\ell,i}$ is the number of edges in the depth-2ℓ directed neighborhood of the i-th edge. Note that α_i is identically distributed but not independent. So the law of large numbers does not apply in general.

Since the check node degree is an i.i.d. random variable. By applying the union bound to Lemma .1, the probability that every check node in the residual graph has degree no more than cn'^{θ} can be bounded as follows

$$\Pr(\text{every CN has degree} \leq n'^{\theta}) < n'e^{-An'^{\theta}} \leq e^{-A'n'^{\theta}} \tag{F.3}$$

for sufficiently large n', where $A' = A - \epsilon$.

For a (j, n'^{θ}) regular ensemble, $T_{\ell,i} = T_\ell = \Theta\left(n'^{\ell\theta}\right)$ and $\alpha_t^2 = \Theta\left(n'^{2\ell\theta}\right)$. So we have

$$-\frac{(n'j\epsilon)^2}{2\sum_{i=1}^{n'(1+j)}\alpha_i^2} \leq -\frac{B'(n'j\epsilon)^2}{2n'jn'^{2\ell\theta}} = -B\epsilon^2 n'^{1-2\ell\theta}$$

given every check node has degree no more than n'^{θ} where B' and B are constants independent of n'. Note that the event that $|\frac{Z^{(\ell)}}{n'_e} - p'^{(\ell)}| \geq \epsilon$ can be caused by two reasons. The first one is the rare events that captured by Azuma's inequality. The

second reason is that some check nodes have very large degrees such that the RHS of (3.4) has positive exponent. Equation (F.2) shows the probability that $|\frac{Z^{(\ell)}}{n'_e} - p'^{(\ell)}| \geq \epsilon$ which is caused by the first reasons and (F.3) shows the probability that $|\frac{Z^{(\ell)}}{n'_e} - p'^{(\ell)}| \geq \epsilon$ which is caused by the second reason. Note that the event $|\frac{Z^{(\ell)}}{n'_e} - p'^{(\ell)}| \geq \epsilon$ can be caused by both of the first and the second reasons. By the union bound,

$$\Pr\left\{\left|\frac{Z^{(\ell)}}{n'_e} - p'^{(\ell)}\right| \geq \epsilon\right\} \leq n' e^{-An'^{\theta}} + 2e^{-B\epsilon^2 n'^{1-2\ell\theta}}. \tag{F.4}$$

Now we set $1 - 2\ell\theta = \theta$ to balance the probability that Azuma's inequality fails and the probability that any check node has too large degree. This gives $\theta = \frac{1}{1+2\ell}$. So (F.4) can be rewritten as

$$\Pr\left\{\left|\frac{Z^{(\ell)}}{n'_e} - p'^{(\ell)}\right| \geq \epsilon\right\} \leq n' e^{-\beta'\epsilon^2 n'^{\frac{1}{1+2\ell}}} = n' e^{-\beta'\epsilon^2 n^{\frac{1-\omega}{1+2l}}}$$

where β' is a constant independent of n. □

APPENDIX G

Proof of Lemma 4

Proof. All statements are implied to hold for all $k > \overline{\alpha}_j^{j-1}$, all $x \in [0,1]$, and all $\alpha \in [0, \overline{\alpha}_j]$. Since $1-(1-x)^k$ is concave for $k \geq 1$, the tangent upper bound at $x = 0$ shows that $1-(1-x)^k \leq kx$. This implies that

$$\left(1 - \left(1 - \frac{\overline{\alpha}_j x}{k^{j/(j-1)}}\right)^k\right)^{j-1} \leq \frac{\overline{\alpha}_j^{j-1} x^{j-1}}{k}. \tag{G.1}$$

Since $\frac{\alpha}{k^{j/(j-1)}} \leq \frac{\overline{\alpha}_j}{\overline{\alpha}_j^{j-1}} \leq 1$, we can use (G.1) to upper bound $g_{k+1}(x)$ with

$$\begin{aligned} g_{k+1}(x) &\leq \frac{\alpha}{\overline{\alpha}_j}\left(1 - \left[1 - \frac{\overline{\alpha}_j^{j-1} x^{j-1}}{k}\right.\right. \\ &\quad \left.\left. + \frac{\alpha}{k^{j/(j-1)}} \frac{\overline{\alpha}_j^{j-1} x^{j-1}}{k} - \frac{\overline{\alpha}_j x}{k^{j/(j-1)}}\right]^k\right)^{j-1} \\ &\leq \frac{\alpha}{\overline{\alpha}_j}\left(1 - \left[1 - \frac{\overline{\alpha}_j^{j-1} x^{j-1}}{k} - \frac{\overline{\alpha}_j x}{k^{j/(j-1)}}\right]^k\right)^{j-1}. \end{aligned}$$

This completes the proof of (i).

The fact that $\overline{g}_{k+1}(x)$ is monotonically decreasing follows from Lemma 3. This completes the proof of (ii). Lemma 3 also shows that the limit of $\overline{g}_{k+1}(x)$ is

$$g_*(x) \triangleq \frac{\alpha}{\overline{\alpha}_j}\left(1 - e^{-\overline{\alpha}_j^{j-1} x^{j-1}}\right)^{j-1}.$$

This proves the first part of (iii).

Next, we will show that

$$\lim_{k\to\infty} g_k(x) = \frac{\alpha}{\overline{\alpha}_j}\left(1 - e^{-\overline{\alpha}_j^{j-1}x^{j-1}}\right)^{j-1}.$$

First, we show that

$$\lim_{k\to\infty} k\left(1 - \left(1 - \frac{\overline{\alpha}_j x}{k^{j/(j-1)}}\right)^k\right)^{j-1} = \overline{\alpha}_j^{j-1}x^{j-1}. \tag{G.2}$$

In light of the the upper bound (G.1), the limit is clearly upper bounded by $\overline{\alpha}_j^{j-1}x^{j-1}$. Using the lower bound in Lemma 3, we see that

$$\begin{aligned}\left(1 - \frac{\overline{\alpha}_j x}{k^{j/(j-1)}}\right)^k &\geq \frac{e^{-\overline{\alpha}_j x k^{-1/(j-1)}}}{\left(1 + \overline{\alpha}_j x k^{-j/(j-1)}\right)^{\overline{\alpha}_j x k^{-1/(j-1)}}} \\ &\geq \frac{e^{-\overline{\alpha}_j x k^{-1/(j-1)}}}{\left(1 + \overline{\alpha}_j x k^{-j/(j-1)}\right)} \\ &\geq \left(1 - \frac{\overline{\alpha}_j x}{k^{j/(j-1)}}\right) e^{-\overline{\alpha}_j x k^{-1/(j-1)}}.\end{aligned}$$

This implies that

$$\left(1 - \left(1 - \frac{\overline{\alpha}_j x}{k^{j/(j-1)}}\right)^k\right)^{j-1} \geq \left(1 - \left(1 - \frac{\overline{\alpha}_j x}{k^{j/(j-1)}}\right) e^{-\overline{\alpha}_j x k^{-1/(j-1)}}\right)^{j-1}.$$

Together with

$$\lim_{k\to\infty} k\left(1 - \left(1 - \frac{\overline{\alpha}_j x}{k^{j/(j-1)}}\right) e^{-\overline{\alpha}_j x k^{-1/(j-1)}}\right)^{j-1} = \overline{\alpha}_j^{j-1}x^{j-1},$$

we see that the limit (G.2) holds.

To calculate the limit of $g_k(x)$, we can use the fact that $\lim_{k\to\infty}\left(1 - a_k + o\left(\frac{1}{k}\right)\right)^k = e^{-\lim_{k\to\infty} k a_k}$ whenever $\lim_{k\to\infty} k a_k$ exists. Using this, we see that $\lim_{k\to\infty} g_{k+1}(x)$ can be rewritten as

$$\lim_{k\to\infty} \frac{\alpha}{\overline{\alpha}_j} \left(1 - \left[1 - \left(1 - \left(1 - \frac{\overline{\alpha}_j x}{k^{j/(j-1)}} \right)^k \right)^{j-1} + o\left(\frac{1}{k}\right) \right]^k \right)^{j-1}$$
$$= \frac{\alpha}{\overline{\alpha}_j} \left(1 - e^{-\overline{\alpha}_j^{j-1} x^{j-1}} \right)^{j-1},$$

where the last step follows from (G.2). □

APPENDIX H

Proof of Lemma 2

Proof. Recall that $\bar{\alpha}_j$ is defined as the largest α s.t. $\left(1-e^{-\alpha^{j-1}x^{j-1}}\right)^{j-1} \leq x$ for $x \in (0,1]$. So $\bar{\alpha}_j$ can be written as

$$\bar{\alpha}_j = \inf_{x\in(0,1]} h_j(x) \tag{H.1}$$

where $h_j(x) = \left(-\ln\left(1-x^{1/(j-1)}\right)x^{(1-j)}\right)^{1/(j-1)}$. Notice that $h_j(x)$ is a monotonically increasing function of x when $j=2$. So we have

$$\bar{\alpha}_2 = \lim_{x\to 0} h_j(x) = 1. \tag{H.2}$$

When $j \geq 3$, $h_j(x)$ goes to infinity when x goes to either 0 or 1, so the infimum is achieved at an interior point x_j^*. By taking derivative of x and setting it to zero, x_j^* is the solution of

$$\frac{x^{\frac{1}{j-1}}}{\left(1-x^{\frac{1}{j-1}}\right)\ln\left(1-x^{\frac{1}{j-1}}\right)} = -(j-1)^2. \tag{H.3}$$

So

$$x_j^* = \left(1+\frac{1}{(j-1)^2 W_{-1}\left(-e^{-1/(j-1)^2}/(j-1)^2\right)}\right)^2. \tag{H.4}$$

By solving this numerically, we find that $x_3^* \approx 0.816042$, $x_4^* \approx 0.938976$ and $x_5^* \approx 0.971087$. Substituting x_j^* into (H.1), we have $\bar{\alpha}_3 \approx 1.87321$, $\bar{\alpha}_4 \approx 1.66455$ and $\bar{\alpha}_5 \approx 1.52073$. □

APPENDIX I

Proof of Lemma 5

Proof. Let us define the function $\hat{g}_k(x)$ with

$$\hat{g}_k(x) \triangleq \frac{\alpha}{\bar{\alpha}_j}\left(1-\left(1-\frac{\alpha j x}{k}\right)^{k-1}\right)^{j-1} + (j-1)\left(1-\left(1-\frac{\alpha j x}{k}\right)^{k-1}\right)^{j-2}\left(1-\frac{\alpha j x}{k}\right)^{k-1}.$$

To prove (i), we will show $g_k(x) < \hat{g}_k(x) < \bar{g}_k(x)$. To see that $g_k(x) < \hat{g}_k(x)$, we must simply observe that

$$1-\left(1-\frac{1-\frac{\alpha j}{k}}{1-\frac{\alpha j x}{k}}\left(1-\left(1-\frac{\alpha j x}{k}\right)^{k-1}\right)^{j-1}\right)^{k-1} < 1.$$

This can be seen by working from the inner expression outwards and using the facts that $0 < \frac{\alpha j}{k} < 1$ and $0 < x < 1$. Each step gives a result that is bounded between 0 and 1.

To show $\hat{g}_k(x) < \bar{g}_k(x)$, we first change variables to $z = \left(1-\frac{\alpha j x}{k}\right)^k$ where $z \in (0,1)$. This allows $\bar{g}_k(x)$ to be written as a function of z with

$$\bar{g}_k(z) = \frac{\alpha}{\bar{\alpha}_j}\left((1-z)^{j-1} + (j-1)(1-z)^{j-3} z\right). \tag{I.1}$$

Taking the derivative of $\bar{g}_k(z)$ with respect to z gives

$$\frac{\mathrm{d}\bar{g}_k(z)}{\mathrm{d}z} = -\frac{\alpha}{\bar{\alpha}_j}(j-2)(j-1)(1-z)^{j-3}z \tag{I.2}$$

which is negative for $j \geq 3$. So $\bar{g}_k(z)$ is a monotonically decreasing function of z. Using the inequality $\left(1-\frac{\alpha jx}{k}\right)^{k-1} > \left(1-\frac{\alpha jx}{k}\right)^{k}$, we find that $\hat{g}_k(x) < \bar{g}_k(x)$.

Next, we will prove (ii) by showing the limits of $g_k(x)$ and $\bar{g}_k(x)$ are the same. First, we take the the term by term limit of $\bar{g}_k(x)$ to see that

$$\begin{aligned}\lim_{k\to\infty}\bar{g}_k(x) =& \frac{\alpha}{\bar{\alpha}_j}\Big(\left(1-e^{-\alpha jx}\right)^{j-1} + \\ & (j-1)\left(1-e^{-\alpha jx}\right)^{j-2}e^{-\alpha jx}\Big) \\ =& \frac{\alpha}{\bar{\alpha}_j}\left(1-e^{-\alpha jx}\right)^{j-2}\left(1+(j-2)e^{-\alpha jx}\right).\end{aligned} \tag{I.3}$$

Next, we use the fact that $\left(1-\frac{\alpha jx}{k}\right)^{k-1} \to e^{-\alpha jx}$ to see that

$$\lim_{k\to\infty}\left(1-\frac{1-\frac{\alpha j}{k}}{1-\frac{\alpha jx}{k}}\left(1-\left(1-\frac{\alpha jx}{k}\right)^{k-1}\right)^{j-1}\right)^{k-1} = 0.$$

From this, we find that the term by term limit of $g_k(x)$ is also equal to (I.3).

To prove (iii), we recall that, using the change of variables $z = \left(1-\frac{\alpha jx}{k}\right)^{k}$, $\bar{g}_k(z)$ is a monotonically decreasing function of z. Moreover, $\bar{g}_k(z)$ does not depend on k and $z = \left(1-\frac{\alpha jx}{k}\right)^{k}$ is a monotonically increasing function of k (e.g., see Lemma 3). So $\bar{g}_k(x)$ is a monotonically decreasing function of k. □

APPENDIX J

Proof of Lemma 6

Proof. Consider whether the sequences x_k and y_k converge to zero or not. Clearly, there are only 4 possible cases.

If $x_k = o(1)$ and $y_k = \Omega(1)$, the limit

$$\lim_{k\to\infty} \beta_k = \lim_{k\to\infty} \frac{y_k(1+y_k)^{k-1}}{(1+y_k)^k} = \lim_{k\to\infty} y_k \tag{J.1}$$

contradicts $\beta_k = \Theta\left((k-1)^{-j/(j-2)}\right)$.

If $x_k = \Omega(1)$ and $y_k = o(1)$, the limit

$$\lim_{k\to\infty} k\beta_k = \lim_{k\to\infty} \frac{y_k\left(1+x_k\right)^{k-1}}{y_k\left(1+x_k\right)^{k-1}} = 1$$

contradicts $\beta_k = \Theta\left((k-1)^{-j/(j-2)}\right)$.

If $x_k = \Omega(1)$ and $y_k = \Omega(1)$, the limit satisfies

$$\begin{aligned}\lim_{k\to\infty} \beta_k &= \lim_{k\to\infty} \frac{y_k\left(1+x_k+y_k\right)^{k-1}}{\left(1+x_k+y_k\right)^k - \left(1+x_k\right)^k} \\ &> \lim_{k\to\infty} \frac{y_k}{1+x_k+y_k},\end{aligned}$$

and this contradicts $\beta_k = \Theta\left((k-1)^{-j/(j-2)}\right)$. □

APPENDIX K

Proof of Lemma 7

Since all stopping sets with size sublinear in n shrink to the zero point on the scaled curve, we must treat sublinear stopping sets separately. The proof proceeds by considering separately stopping sets of size $O(\ln n)$ and size δn for very small δ. The number of correct and incorrect variable nodes in a stopping set is denoted, respectively, a and b (i.e., $n\alpha = a$ and $n\beta = b$).

Proof. Using (3.12) and Lemma .2, we can bound $E_{n,j,k}(\alpha, \beta)$ with

$$E_{n,j,k}(\alpha,\beta) \leq j e^{\frac{1}{12jn}} e^{(1-j)nh(\alpha,\beta,1-\alpha-\beta)} S_{n,j,k}(\alpha n, \beta n).$$

The coefficient $S_{n,j,k}(a, b)$ can be bounded using a Chernoff-type bound and this gives

$$\begin{aligned}\ln S_{n,j,k}(a,b) &\leq \frac{jn}{k} \ln \frac{1+(1+x+y)^k - ky - (1+x)^k}{x^{ja} y^{jb}} \\ &\leq \frac{jn}{k} \ln \left((1+x+y)^k - ky - kx\right) - ja \ln x - jb \ln y\end{aligned}$$

for arbitrary $x \geq 0$ and $y \geq 0$. Choosing $x = \frac{1}{\sqrt{n}}$ and $y = \frac{1}{\sqrt{n}}$ gives the bound

$$\begin{aligned}S_{n,j,k}(a,b) &\leq e^{2j(k-1)+O(n^{-1/2})} n^{(a+b)j/2} \\ &\leq C n^{(a+b)j/2}, \end{aligned} \tag{K.1}$$

where C is a constant independent of n. Applying (K.1) to the $E_{n,j,k}(\alpha, \beta)$ bound

shows that $E_{n,j,k}\left(\frac{a}{n},\frac{b}{n}\right)$

$$\begin{aligned}
&\leq je^{\frac{1}{12nj}}\exp\left((1-j)nh\left(\frac{a}{n},\frac{b}{n},1-\frac{a}{n}-\frac{b}{n}\right)\right)S_{n,j,k}(a,b)\\
&\leq je^{\frac{1}{12nj}}\left(\frac{a}{n}\right)^{(j-1)a}\left(\frac{b}{n}\right)^{(j-1)b}S_{n,j,k}(a,b)\\
&\leq je^{\frac{1}{12j}}Cn^{(a+b)(j/2-(j-1)(1-\epsilon))}\left(\frac{a}{n^{\epsilon}}\right)^{(j-1)a}\left(\frac{b}{n^{\epsilon}}\right)^{(j-1)b} \qquad \text{(K.2)}
\end{aligned}$$

where $0<\epsilon<\frac{1}{4}$ and $j\geq 3$.

Now, we can use this to show that

$$\lim_{n\to\infty}\sum_{b=1}^{A\ln n}\sum_{a=0}^{n-b}E_{n,j,k}\left(\frac{a}{n},\frac{b}{n}\right)=0.$$

Since a stopping set cannot have a check node that attaches to only verified and correct edges, a simple counting argument shows that $S_{n,j,k}(a,b)=0$ if $a>(k-1)b$. Therefore, the above condition can be simplified to

$$\lim_{n\to\infty}\sum_{b=1}^{A\ln n}\sum_{a=0}^{(k-1)b}E_{n,j,k}\left(\frac{a}{n},\frac{b}{n}\right)=0. \qquad \text{(K.3)}$$

Starting from (K.2), we note that $b\leq A\ln n$ and $a\leq(k-1)b$ implies that $\left(\frac{a}{n^{\epsilon}}\right)^{(j-1)a}\left(\frac{b}{n^{\epsilon}}\right)^{(j-1)b}<1$ for large enough n. Therefore, we find that the double sum in (K.3) is upper bounded by

$$je^{\frac{1}{12j}}Cn^{(j/2-(j-1)(1-\epsilon))}(k-1)(A\ln n)^2$$

for large enough n. Since the exponent $(a+b)(j/2-(j-1)(1-\epsilon))$ of n is negative as long as $\epsilon<\frac{1}{4}$ and $j\geq 3$, we also find that the limit of the double sum in (K.3) goes to zero as n goes to infinity for any $A>0$.

Now, we consider stopping sets of size greater than $A\ln n$ but less than $\delta_{j,k}n$. Combining (3.15) and Lemma .2 shows that $E_{n,j,k}(\alpha,\beta)\leq je^{\frac{1}{12jn}}e^{n\gamma_{j,k}(\alpha,\beta)}$. Notice that (3.19) is an accurate upper bound on $\gamma_{j,k}(\alpha,\beta)$ for small enough β and its maximum

over α is given parametrically by (3.18). Moreover, $v(d)$ is strictly decreasing at $d = 0$, and this implies that $\gamma_{j,k}(\alpha, \beta)$ is strictly decreasing in β at $\beta = 0$ for all valid α. Therefore, there is a $\delta_{j,k} > 0$ and $\eta > 0$ such that

$$\gamma_{j,k}(\alpha, \beta) < -\eta\beta$$

for all $0 \leq \beta \leq \delta_{j,k}$. From this, we conclude that $\frac{A \ln n}{n} < \beta < \delta_{j,k} n$ which implies that

$$\begin{aligned} E_{n,j,k}(\alpha, \beta) &\leq j e^{\frac{1}{12jn}} e^{n\gamma_{j,k}(\alpha,\beta)} \leq j e^{\frac{1}{12jn}} e^{-n\eta\beta} \\ &\leq j e^{\frac{1}{12jn}} e^{-\eta A \ln n} \leq j e^{\frac{1}{12jn}} n^{-A\eta}, \end{aligned}$$

where $A\eta$ can be made arbitrarily large by increasing A. Choosing $A = \frac{3}{\eta}$ so that $A\eta = 3$ shows that

$$\lim_{n\to\infty} \sum_{b=\log n}^{\delta_{j,k} n} \sum_{a=0}^{n-b} E_{n,j,k}\left(\frac{a}{n}, \frac{b}{n}\right) \leq \lim_{n\to\infty} n^2 j e^{\frac{1}{12jn}} n^{-3} = 0.$$

This completes the proof. □

APPENDIX L

Lemma .2

Lemma .2. *The ratio* $D \triangleq \dfrac{\binom{n}{a,b,n-a-b}}{\binom{nj}{aj,bj,(n-a-b)j}}$ *can be bounded with*

$$j \exp\left((1-j)nh(\frac{a}{n},\frac{b}{n},1-\frac{a}{n}-\frac{b}{n})-\frac{1}{12n}\right) \leq D$$
$$\leq j \exp\left((1-j)nh(\frac{a}{n},\frac{b}{n},1-\frac{a}{n}-\frac{b}{n})+\frac{1}{12jn}\right).$$

Proof. Let D be defined by

$$D = \frac{\binom{n}{a+b}\binom{a+b}{a}}{\binom{nj}{(a+b)j}\binom{(a+b)j}{aj}}.$$

Using Stirling's approximation, the binomial coefficient can be bounded using

$$\frac{1}{\sqrt{2\pi n\lambda(1-\lambda)}}\exp\left(nh(\lambda)-\frac{1}{12n\lambda(1-\lambda)}\right)\leq\left(\begin{array}{c}n\\ \lambda n\end{array}\right)$$
$$\leq\frac{1}{\sqrt{2\pi n\lambda(1-\lambda)}}\exp\left(nh(\lambda)\right),$$

where $h(\cdot)$ is the entropy function in nats [55]. Applying this bound to D gives, after some manipulation, that

$$j\exp\Bigg((1-j)\Bigg(nh\left(\frac{a+b}{n},1-\frac{a+b}{n}\right)+$$
$$(a+b)h\left(\frac{a}{a+b},\frac{b}{a+b}\right)\Bigg)-\frac{1}{12n}\Bigg)\leq D$$
$$\leq j\exp\Bigg((1-j)\Bigg(nh\left(\frac{a+b}{n},1-\frac{a+b}{n}\right)+$$
$$(a+b)h\left(\frac{a}{a+b},\frac{b}{a+b}\right)\Bigg)+\frac{1}{12jn}\Bigg).$$

Finally, we notice that

$$nh\left(\frac{a+b}{n}, 1-\frac{a+b}{n}\right)+(a+b)h\left(\frac{a}{a+b}, \frac{b}{a+b}\right)= nh\left(\frac{a}{n}, \frac{b}{n}, 1-\frac{a}{n}-\frac{b}{n}\right).$$

This completes the proof. □

VITA

Fan Zhang received his B.S. and M.S. degrees both in Electrical Engineering at University of Science and Technology of China, Hefei, Anhui, China in 2003 and 2006, respectively. During 2003 and 2006, he also worked with UTStarcom Inc. R&D at Hefei, Anhui China. From 2006-2010, Zhang was a research assistant at Texas A&M University under Professor Henry D. Pfister. His contact address is Mailbox 407, Dept. ECE, TAMU, College Station, TX, 77840.

The typist for this dissertation was Fan Zhang.