Coding Techniques for Data-Storage Systems

Thesis by

Yuval Cassuto

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

California Institute of Technology

Pasadena, California

2008

(Defended December 3, 2007)

This thesis is dedicated to my father, Moshe, who did not enjoy the longevity to read it, but whose tacit drive toward high education has been the key to its completion.

Acknowledgements

Acknowledging all of Jehoshua (Shuki) Bruck's contributions to this thesis is a pleasure but simultaneously a challenge. His fingerprints are vividly present on all aspects of research chronicled in this thesis – problem selection, motivations, solutions and presentation. Yet Shuki's contributions to my professional advancement transcend far beyond what found in this bulky text – a guide to creative innovation in a human-friendly environment being the most notable of all.

Through enlightening teachings and numerous discussions, Robert J. McEliece has introduced me to, and guided me in, the exciting field of error-control codes – maybe the shortest bridge to connect deep Mathematics with useful engineering.

Enjoyable conversations with Mario Blaum all stirred strong impact on my research directions. It took some time to believe that this friendly and funny persona is the one behind most major advancements in the area of array codes.

Lihao Xu, a leading expert in data-storage codes, contributed his encyclopedic knowledge and sharp observations to help in focusing this research and aligning it with the broader data-storage community.

Tracey Ho, despite being preoccupied in a journey to sustained stardom, has always volunteered her breadth and depth in the Information Sciences to offer insights pertaining to this work.

Last but not least in this fine list, is Michelle Effros whose wide shoulders in Information Theory have helped me see farther and better, to contribute to the area of Network Coding – an inspiring field with obvious and less obvious links to the material of this thesis.

In addition to the gratitude that goes to my Thesis Committee, I feel indebted to many other individuals for their positive impact on my Caltech experience. Mates in the Paradise Laboratory have filled my work days with enjoyable conversations, educated advice and occasional laughs. Friends and academic soul-mates in the first floor's Communications, Data-Compression, and Signal-Processing Laboratories, with whom I shared research interests, ideas and opinions. Many friends from across Caltech, all happen to be excellent scientists, for the great after-school activities seasoned with cross-disciplinary scientific discussions.

Speaking of friendship, wisdom and amusement, my true friend Miki Lustig has shown that, similarly to degrees on a circle, 360 miles can be a very short distance away.

A reliable, remotely operated support system throughout my studies has been my loving family in Israel, as well as my good old friends. My mother Sara, my sister Merav, my brother Yair, my parents-in-law Zvi and Rachel and siblings-in-law Nili and Danny – I hope that reading this thesis will provide good answers to the question "What is he doing there all that time?"

Finally, the ultimate words of recognition go to my sweet and joyful wife Ruth. With just me and four mid-size suitcases, she happily joined this academic adventure and had made it the success that it has been. On days when this thesis rocketed forward, and on days when it was buried in trenches, the unconditional love from her and from our two marvels Asaf and Gili, has been the fuel that kept this research going.

Contents

List of Figures

List of Tables

Abstract

As information-bearing objects, data-storage systems are natural consumers of information-theoretic ideas. For many issues in data-storage systems, the best trade-off between cost, performance and reliability, passes through the application of error-correcting codes. Error-correcting codes that are specialized for data-storage systems is the subject studied by this thesis. On the practical side, central challenges of storage systems are addressed, both at the individual-device level and higher at the enterprise level for disk arrays. The results for individual devices include a new coding paradigm for Multi-Level Flash storage that benefits storage density and access speed, and also a higher-throughput algorithm for decoding Reed-Solomon codes with large decoding radii. The results for storage arrays address models and constructions to combat correlated device failures, and also introduce new highly-regular array-code constructions with optimal redundancy and updates. On the theoretical side, the research stretches across multiple layers of coding theory innovation: new codes for new error models, new codes for existing error models, and new decoding techniques for known codes. To bridge the properties and constraints of practical systems with the mathematical language of coding theory, new well-motivated models and abstractions are proposed. Among them are the models of t *asymmetric* ℓ*-limited-magnitude errors* and *Clustered erasures*. Later, after maximizing the theory's power in addressing the abstractions, the performance of storage systems that employ the new schemes is analytically validated.

Chapter 1

Introduction

Error-correcting codes are a cardinal component of any modern information-bearing system. In highly optimized systems, it is either impossible or inefficient to guarantee perfectly reliable information throughout the system, and thus error-correcting codes are employed to protect the user's data from the aggregate system imperfection. It may be fair to say that among all system components, the error-correcting code is the least accessible and least understood one by engineering professionals outside this specific expertise. This fact can be attributed to the inherent exponential blowout of the code space, that renders impractical any bottom-up design technique that may be very effective for other system blocks. Simulations are partially effective in predicting some of the behaviors of the coding sub-system, but they largely fail in providing sufficient insight that would assist the synthesis of good codes. That often leads to shifting the design efforts away from the error-correcting code, and resorting to codes that had proved successful in other systems – overlooking the special characteristics of the particular system that may allow the incorporation of more efficient codes. Contributing to that phenomenon is the exceptional success that Coding Theory has already achieved: finding good codes that efficiently approach various theoretical limits.

Yet those victories of Coding Theory in combating a few channel models should not raise the misconception that all the good codes have already been found. As important as those channels may be, they represent only the tip of the general theory of information. Claude E. Shannon, in his founding article of Information Theory, formulated his ideas in rigorous mathematical terms, but also included a general recipe to obtain reliable information, put in layman's words [Sha48, Section 14]:

”The redundancy must be introduced in the proper way to combat the particular noise structure involved.”

Hence a core foundation in theory, as well as a promising strategy in practice, is to understand the system’s underlying unreliability sources, both natural and man(engineer)-made, and use that knowledge in the construction of better codes.

While the idea of tailoring the solution to the specificities of the problem may sound trivial, in the context of Coding Theory it enfolds a primary challenge. In order to provide precise error-control characterizations of combinatorial structures, there is a need to introduce new *abstractions*, that on one hand represent the system realities, and on the other hand are amenable to analysis and design. Thus an essential step between understanding the problem and finding a solution, is the search for useful abstractions that will constitute the bridge from practice to theory. Later, after maximizing the theory’s power in addressing the abstractions, performance *validation* will constitute the return bridge from theory to practice (See Figure 1.1). Each chapter of this thesis embarks on such a round trip from practice to theory and then back – leveraging new theoretical methods to the improvement of storage-system performance.

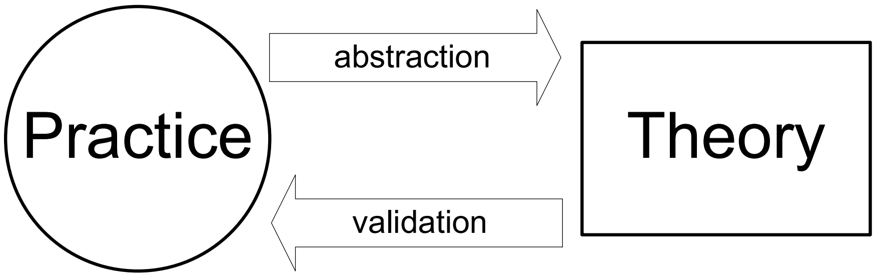


Figure 1.1: Coding Theory and Practice

1.1 Coding for Data-Storage Systems

“Pundits have proclaimed it for years; articles in the popular press have plumbed its implications for every imaginable enterprise; businesses are enamored with it; on-line and print magazines are devoted to it; government is

wrestling with it, movies have been made about it; people are talking about it–can there be any doubt?" –James A. Dewar

There is no doubt that mankind has entered the "Information Age". In any area of life, we are immersed in information. The most obvious product of information is... more information, hence orders of magnitudes growth in its quantities are exhibited in short time scales. For the data-storage industry, a consistent, steep increase in demand introduces technological challenges, since leaps in storage densities require changes that considerably alter the design framework and implementation constraints. Truths, wisdoms and arts that were highly effective yesterday, may be secondary or obsolete today. Unfortunately, market pressures often do not allow an orderly rethinking process for the new challenges at hand, and instead crude adaptations of previous schemes are pursued.

Storage systems in general, have some common properties that affect the implementation of error-correcting codes within them. Such properties, those that are the most relevant to the results of subsequent chapters, are listed below.

- **High access speeds.** Storage devices provide information transfer rates in the order of 100-MB/sec (Mega-Byte per second). Such high access speeds enforce stringent constraints on the complexity of the coding modules, and disallow coding schemes that are viable options in systems with significantly lower transfer rates.

- **Dynamic updates.** Information stored in dynamic-storage systems changes frequently in unpredictable patterns. Therefore, re-encoding the information after each small update is inefficient, and codes are required to minimize the number of parity updates needed per small information update.

- **Flexibility.** In storage devices the encoder and decoder are implemented in the same physical module, obviating issues of standardization and interoperability that hinder coding novelty in communication applications.

- **Controlled error sources.** Error-correcting codes can be used in storage systems to combat errors that are intentionally introduced in a controlled way. The introduction of controlled errors, and correction thereof, allow more flexibility in the performance requirements from other system blocks. The error-correcting code thus allows system components to dynamically trade-off different performance parameters.

- **Media variety.** There are numerous types of storage media and architectures, each with dissimilar properties and challenges. Emerging storage technologies, with their unique imperfections, keep storage error-correcting codes a vibrant and diverse research area.

This list suggests that common abstractions addressed by Coding Theory (e.g. minimum distance, code rate, decoding complexity), are insufficient to capture the diverse properties and constraints of storage systems, and warrants the introduction and study of useful new ones.

1.2 Storage-System Challenges

Delivering cost-effective reliable data storage to users is a paramount mission that involves a variety of efforts. As in other competitive technological markets, the numerous engineering challenges of large-scale storage systems are divided and encapsulated in standardized layers, allowing vendors to offer highly specialized solutions for small parts of the general problem. At the device level, the main challenge is to tame a chosen physical media (e.g. Magnetic, Semiconductor, Micro-mechanical) into a dense and reliable storage unit. At the enterprise level, multiple devices of different kinds and characteristics are combined into a storage array that protects the data from failures of individual units. Error-correcting codes are a major ingredient in driving performance and reliability of both storage devices and storage arrays. Higher layers of storage systems handle a variety of non-trivial services such as virtualization, backups and security. The results of this thesis address immediate concerns of storage systems, both at the device level (codes for Multi-level Flash memories, improved decoding of Reed-Solomon codes) and at the enterprise level (efficient array codes for Clustered failures, highly regular array codes with optimal redundancy and updates). Therefore, it is hoped and believed that the fast-evolving and innovation-demanding data-storage technology will benefit from the proposed methods and ideas.

1.3 Summary of Contributions

1.3.1 Contribution Hierarchy

From a coding-theoretic perspective, three layers of novelty comprise the results of this thesis. Clearly no layer is generally more important than others to the advancement of storage error-correcting codes, but this classification helps in ordering the chapters of the thesis to follow some hierarchy. As depicted in Figure 1.2, the top layer, codes for new

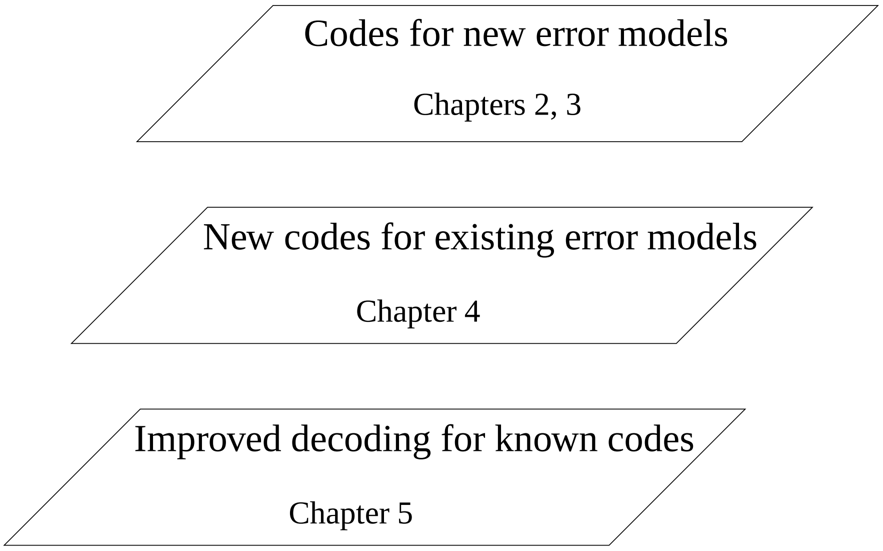

Codes for new error models
Chapters 2, 3
New codes for existing error models
Chapter 4
Improved decoding for known codes
Chapter 5

Figure 1.2: Three Layers of Novelty

error models, consists of Chapters 2 and 3. The middle layer, new codes for existing error models, includes the new MDS codes of Chapter 4. The bottom layer, improved decoding for known codes, is the subject of Chapter 5.

What follows next is a summary of the subjects studied in this thesis. For each subject we note the main observations that triggered its investigation, and summarize the impact on this subject by differentiating our research contributions from previously known results.

1.3.2 Asymmetric Limited-Magnitude Error-Correcting Codes for Multi-Level Flash Memories

In Multi-Level Flash Memories, the cell's range of threshold levels is discretized to q levels. Programming a cell to one particular level thus represents $\log q$ bits of information. Representing multiple bits in a single cell improves the storage density, with an obvious toll on error margins whose shrinking affects device reliability and access speeds. The inherent separation between cell programming and cell erasing in the operation of Flash devices makes the dominant error sources *asymmetric* – changing the threshold level in one known direction. Moreover, properties of the physical mechanisms utilized for programming cause errors of low magnitudes to be much more likely than higher magnitude ones. These observations on Multi-Level Flash characteristics are illustrated (for $q = 8$) in Figure 1.3. Level number 1 (circled) is stored by a Flash cell, and is predominantly prone to small errors in the rightward direction. This unique behavior of Flash errors motivates

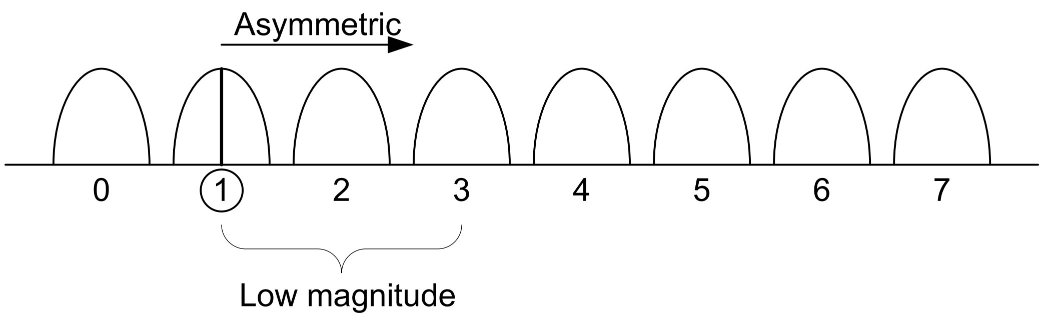


Figure 1.3: Common Errors in Flash Storage

the study of q-ary codes that correct t errors that are both *asymmetric* and have *limited magnitudes*.

The following example illustrates the correction of asymmetric limited-magnitude errors as a special case of the methods of Chapter 2. Suppose we have a group of 5 symbols, each taken from the alphabet $\{0, 1, \ldots, 7\}$. To correct $t = 2$ errors of magnitude $\ell = 1$ in the upward direction, the code is defined as follows. As illustrated by the sample words in Figure 1.4 below, if the codewords are restricted to have either all symbols with odd parity or all symbols with even parity, then the required protection is achieved. For each of the two sample codewords in row (a) of the figure, the channel introduces two upward errors of magnitude 1 (b). By finding the minority between even/odd symbols, the locations of the errors are detected (c)-in bold, and the original symbols are recovered by decrementing the erroneous symbols (d).

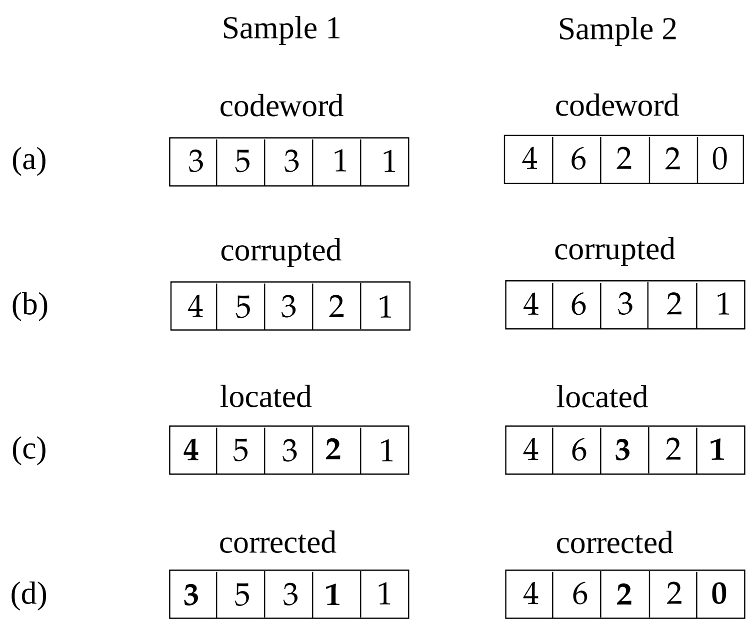


Figure 1.4: Example of correcting asymmetric limited-magnitude errors. (a) Two sample codewords. (b) Introduction of $t = 2$ asymmetric errors with magnitude $\ell = 1$ to each of the sample codewords. (c) Error location by finding the minority between even/odd symbols. (d) Error correction by decrementing the symbols on the error locations.

In Chapter 2, the t asymmetric ℓ-limited-magnitude error model undergoes a comprehensive coding-theoretic treatment. Starting from the error-model definition, sufficient and necessary conditions are proved for codes under that error model, and are then used to construct codes and prove upper bounds on code sizes. For some families of parameters, the main code construction is shown to be optimal. The results are summarized in Table 1.1 below.

Error model	$w_H(\boldsymbol{e}) \leq t,\ 0 \leq e_i \leq \ell$
Sufficient condition	$d_\ell \geq t+1$
Necessary condition	$d_\ell \geq t+1$
Constructions	Constructions 2.1, 2.2, 2.3
Upper bounds	Theorems 2.6, 2.8

Table 1.1: The theory of correcting asymmetric limited-magnitude errors

Beyond its theoretical thrust, Chapter 2 contains multiple contributions to the application of asymmetric limited-magnitude codes in Flash storage devices. Using additional insights on the Flash media, a refined error model is considered; efficient *systematic* code constructions are proposed, and an implementation architecture is described. Maybe the most interesting aspect of applying asymmetric limited-magnitude codes to Flash storage, is that they can be used to speed-up memory write operations by allowing clever introduction of controlled errors by Flash programming algorithms. This aspect is studied in detail at the end of Chapter 2, furnishing that opportunity with both qualitative and quantitative reasoning.

1.3.3 Codes for Random and Clustered Device Failures

Traditionally, MDS (Maximum Distance Separable) array codes are used to protect disk arrays against device failures. Using MDS codes for that purpose implicitly endorses the following two statements:

1. All failure patterns are equally likely for a given number of failed devices.
2. The amount of redundancy has the dominant effect on the implementation cost.

The practical merit of the research detailed in Chapter 3 lies upon the premise that for high-order failure-resilient disk arrays, both statements are not true in practice. Alternative statements that motivate this study are:

1. In failure events that affect many devices, combinations that include clustered failures are more likely than completely isolated failures.
2. Because of severe I/O constraints, the limiting factors on the deployment of high-order failure-correcting codes is their encoding, decoding, and most critically: update complexity.

The first of these observations motivates a new classification of failure combinations – based on both the number of failures and the number of *clusters* that the failures occupy. The well known Random-failure model and Burst-failure model are both special cases at

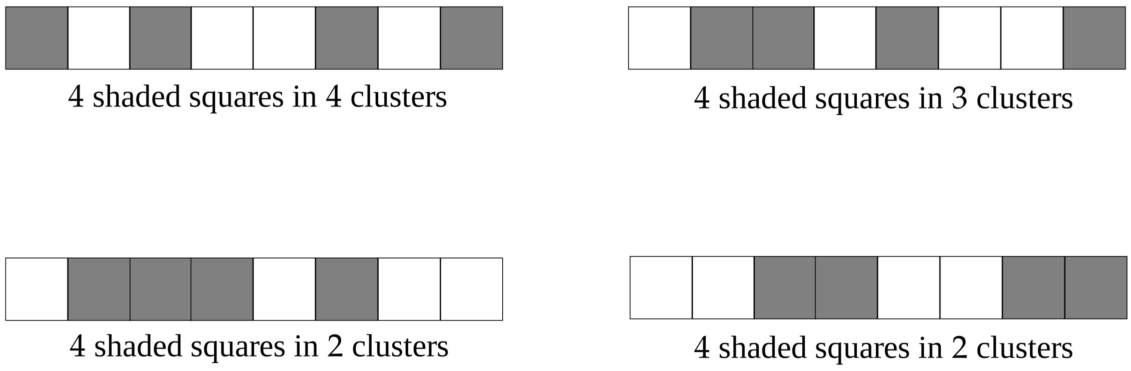

4 shaded squares in 4 clusters
4 shaded squares in 3 clusters
4 shaded squares in 2 clusters
4 shaded squares in 2 clusters

Figure 1.5: Classification of patterns by their respective numbers of clusters. For each array, the number of clusters that contain the four shaded squares is indicated.

the two extremes of this new classification. An abstract classification of patterns according to the number of clusters is given in Figure 1.5.

Compared to the previously studied model of *multiple bursts*, this model seems to better capture correlated failure patterns in disk arrays, since it does not predefine the *size* of the clusters, only their number. Consequently, for the model proposed here, the two patterns at the bottom of Figure 1.5 have the same classification, even though they have distinct cluster sizes. Those two patterns seem like equally plausible outcomes of two “independent” failure events, each affecting multiple disks in a single cluster.

Through a new array-code construction called RC (Random/Clustered) codes, Chapter 3 combines the two alternative observations above to offer a very attractive coding scheme that combines good reliability performance with low implementation complexity. This is done by prioritizing failure combinations based on their cluster classification, and finding more efficient codes that are specialized for the higher priority failures. Proving and illuminating the merits of RC codes is the focus of Chapter 3, yet by taking a broader view they can be regarded as a sample demonstration of the general potential in considering error models that are based on the new error-clustering classification.

1.3.4 Cyclic Lowest-Density MDS Array Codes

Structure is a blessing to an error-correcting code. While random codes usually have unmatched error-probability performance, their usage in practical systems is inconceivable due to implementation-complexity issues. There are many examples where more regular code designs are preferred over unstructured codes, even at the cost of some degree of performance loss. Low Density Parity Check (LDPC) codes is one such area where the challenge of bridging the theoretical state-of-the-art with practical systems involves a careful introduction of structure. It is not just the runtime complexity of the coding blocks that benefits from conforming to some structural constraints, but also the ease of the system specification, implementation and verification.

Put in that light, the three new code constructions of Chapter 4 demonstrate clear value. New highly regular codes with the same favorable properties as known, less structured ones, are an obvious design alternative that can reduce complexity in more than one manner, without compromising the other code properties. The regularity of the proposed array codes is manifested in their *systematically-cyclic* property, which is an especially attractive sub-class of the well known class of *cyclic* codes. An example of a systematically-cyclic array code is given in the Figure 1.6 below.

a_0	a_1	a_2	a_3	a_4	a_5
$a_2+a_3+a_4$	$a_3+a_4+a_5$	$a_4+a_5+a_0$	$a_5+a_0+a_1$	$a_0+a_1+a_2$	$a_1+a_2+a_3$

Figure 1.6: Sample systematically-cyclic lowest-density MDS array code. Each column can be obtained from the column to the left by adding 1 (modulo 6) to all its indices.

This sample code has the property that any of its columns can be obtained by adding 1 (modulo 6) to every index in the column to the left. This property translates to many advantages in the implementation of systematically-cyclic codes in different data-storage systems.

Putting in concrete terms, the codes constructed in Chapter 4 are lowest-density MDS array codes that are also systematically-cyclic. The MDS property means that these codes have optimal redundancy. The lowest-density property means that these codes are optimal in terms of the number of parity-bit updates needed for a single information-bit update. Codes that are both lowest density and MDS are known in the literature, but they are still relatively rare combinatorial structures. Therefore, it comes with some degree of surprise, that there exist codes that enjoy the lowest density and MDS qualities, while simultaneously having a very nice and useful structure of being systematically-cyclic codes.

1.3.5 Decoding Beyond Half the Minimum Distance

Instances of failed decoding are especially undesirable in data-storage systems, since they cost a permanent loss of user data. Thus increasing the decoding radius of error-correcting codes beyond half their minimum distance is an attractive prospect in practice. The two main challenges of decoding beyond half the minimum distance, called in the literature *list decoding*, are the algorithmic feasibility of such decoders, and the effect of non-unique decoding on the post-decoding error probability. Chapter 5 advances our understanding of both issues, and offers constructive algorithmic improvements to decoding Reed-Solomon codes beyond half their minimum distance.

With the objective to improve the average decoding complexity of Reed-Solomon list decoders [GS99], the analysis of interpolation polynomials is refined to understand how their degrees depend on the number of *instantaneous* errors. Previous analyses only considered the number of *worst-case* errors correctable by the code. By bounding polynomial degrees from above given an error weight, a strong such dependence is revealed. That phenomenon then motivates finding an interpolation algorithm whose running time depends on the instantaneous interpolation degree, thus improving the average decoding time and the decoder throughput. A conceptual comparison between the decoding complexity of list-decoding algorithms before and after the contributions of Chapter 5 is illustrated in Figure 1.7 below.

The other major thrust of Chapter 5 is to analyze how decoding beyond the unique-decoding bound affects the miscorrection probability of list decoders. A high miscorrection probability means that in practice increasing the decoding radius comes with the cost of occasionally introducing additional errors instead of correcting existing ones. A new lower bound on the miscorrection probability of list decoders reveals cases where decoding beyond the unique-decoding bound provably and significantly increases the probability of miscorrection. More light on the behavior of list decoders is shed using a new combinatorial upper bound on the codeword-list size output by a list decoder of a general q-ary code.

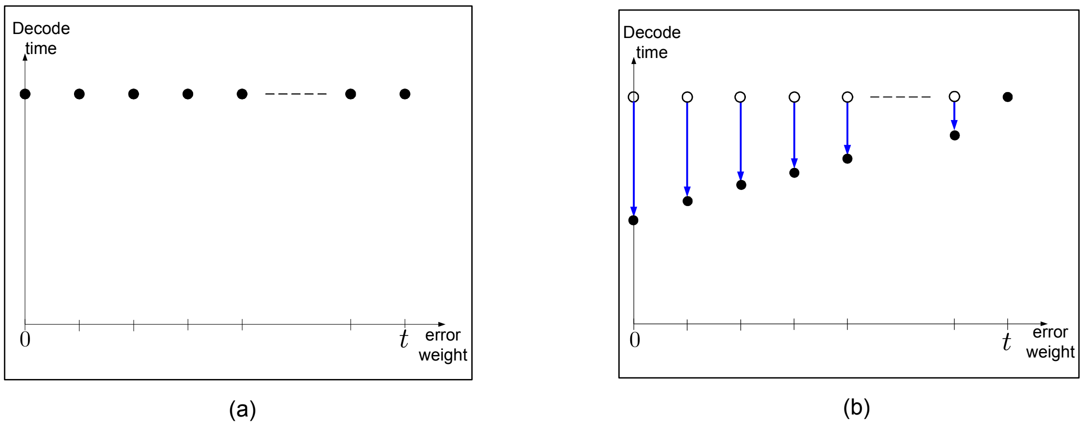


Figure 1.7: Decoding time as a function of the Hamming weight of the error vector. (a) Previously known algorithm whose running time depends only on the worst-case error weight t. (b) A new algorithm and analysis yield running times that decrease as the error weights decrease.

A closed-form bound is derived that improves over the best-known bound for moderate and large alphabet sizes. Curiously, the same proof can be used to obtain an upper bound on the sizes of constant-weight codes, that is better than the classical q-ary Johnson bound for moderate and large alphabet sizes. This improvement is accomplished by proving the following inequality on fundamental coding theoretic entities[1]:

$$A_q(n,d,t) \leq A_2(n,2(d-t),t)$$

1.4 The Audience of the Thesis

The author perceives himself as both a scientist and an engineer. Moreover, the precedence order of the two subjective definitions is variable and may change between one day and the next. Consequently, a blend of practical and theoretical insight has been carefully interwoven to form a cohesive presentation, which hopefully would make it accessible and enjoyable for both types of audiences. In the parts that discuss the engineering aspects of the results, sensible conjectural argumentation was often allowed; but whenever exact mathematical statements appear, their treatment is carried out with uncompromised rigor.

[1]$A_q(n,d,t)$ is the size of the largest q-ary constant-weight code of length n, weight t and minimum distance d. $A_2(n,2(d-t),t)$ is the size of the largest binary constant-weight code of length n, weight t and minimum distance $2(d-t)$.

The academic prerequisites to access the thesis material are not high. Some very basic terminology and general understanding of error-control codes may be found helpful.

Part I

Error Models and Code Constructions

Chapter 2

Asymmetric Limited-Magnitude Error-Correcting Codes for Multi-Level Flash Memories

The observation of physical behaviors that significantly and consistently deviate from the implicit assumptions taken by known models, naturally calls upon new models that better describe the observed behaviors. When that happens, our common wisdom that has helped us to understand and tackle old models is abandoned, and a new theory and design tools need to be developed. The success of a new model, as an object of study, depends on both the practical and the theoretical opportunity spaces that it opens. On the practical side, it should improve matters compared to previously available solutions. On the theoretical side, it should encompass sufficient structure to allow the formulation and manipulation of meaningful mathematical statements that advance its understanding. This chapter presents a comprehensive study of a new error model that is motivated by Multi-Level Flash Memories. The main contributions of the chapter are summarized below.

- Definition and motivation of a new error model: t asymmetric ℓ-limited-magnitude errors.
- A combinatorial necessary and sufficient condition for correctability under the new error model.
- A general and efficient code construction that is shown to be optimal for useful families of parameters, and to outperform the previously best known codes.

- Construction of efficient *systematic* codes to benefit practical implementation.
- Construction of codes for simultaneous asymmetric *and symmetric* ℓ-limited-magnitude errors.
- Analytic study of the Flash-programming speed-up offered by asymmetric limited-magnitude error-correcting codes.

Part of the results hereof have appeared in [CSBB07].

2.1 Introduction

A *channel*, as a mathematical entity [Sha48], specifies the probabilistic relationships between its inputs and its outputs. The Theory of Information studies ways and limitations to attain (communicate/store) reliable information, despite the intrinsic unreliability imposed by the channel. To move from the probabilistic setup of Information Theory to obtain error-control guarantees, an *error model* is derived from the channel model. An error model renounces the probabilistic description of the errors and instead, specifies combinatorial constraints on the error-introducing process, in a precise and deterministic way (usually assuming a specific finite block length).

The most well studied channel model for error-correcting codes is the symmetric channel. According to this model, a symbol taken from the code alphabet is changed by an error event to another symbol from the same alphabet, and all such transitions are equally probable. The natural error model that corresponds to the symmetric channel is the model of *symmetric errors*, whereby the Hamming weight is used as a constraint on legal error vectors. The popularity of the symmetric channel model, and the corresponding Hamming error model, stem from their applicability to practical applications, but more so from the powerful construction techniques that were found to address them. In addition to the symmetric error model, many other models, variations and generalizations were studied, each motivated by a behavior of practical systems or applications. Examples that are most relevant to this chapter are the *binary asymmetric*, *q-ary asymmetric* , and Varshamov's *q-ary asym-*

metric with bounded L1 norm error models, detailed, respectively, in [Klø81], [Web92] and [Var73].

This chapter studies block codes that correct *Asymmetric Limited-Magnitude* errors. This model is parametrized by two integer parameters: t is the maximum number of symbol errors within a code block, and ℓ is the maximal magnitude of an error on any code location. The following example illustrates the coding problem and introduces the main idea of the code construction. Suppose we have a group of 5 symbols, each taken from the alphabet $\{0, 1, \ldots, 7\}$. To correct $t = 2$ errors of magnitude $\ell = 1$ in the upward direction, the code is defined as follows. As illustrated by the sample words in Figure 2.1 below, if the codewords are restricted to have either all symbols with odd parity or all symbols with even parity, the required protection is achieved. For each of the two sample codewords in row (a) of the figure, the channel introduces two upward errors of magnitude 1 (b). By finding the minority between even/odd symbols, the locations of the errors are detected (c)-in bold, and the original symbols are recovered by decrementing the erroneous symbols (d).

	Sample 1					Sample 2				
	codeword					codeword				
(a)	3	5	3	1	1	4	6	2	2	0
	corrupted					corrupted				
(b)	4	5	3	2	1	4	6	3	2	1
	located					located				
(c)	**4**	5	3	**2**	1	4	6	**3**	2	**1**
	corrected					corrected				
(d)	**3**	5	3	**1**	1	4	6	**2**	2	**0**

Figure 2.1: Example of correcting asymmetric limited-magnitude errors. (a) Two sample codewords. (b) Introduction of $t = 2$ asymmetric errors with magnitude $\ell = 1$ to each of the sample codewords. (c) Error location by finding the minority between even/odd symbols. (d) Error correction by decrementing the symbols on the error locations.

As will soon be argued, the model of asymmetric limited-magnitude errors is motivated by the unique error mechanisms that affect reliability and access speed in Multi-Level Flash Memories. Before clearing the stage for that interesting error model, we summarize in Figure 2.2 various asymmetric channels and error models. The top row of Figure 2.2 gives graphical descriptions of three asymmetric channels: the binary asymmetric, the q-ary asymmetric, and the q-ary asymmetric ($\ell = 1$) limited-magnitude channels. The bottom row specifies error models that are derived from the corresponding channel models, with references to the first published result for each model. To the best of our knowledge, no results pertaining to the model of t asymmetric limited-magnitude errors had been published prior to this chapter's contributions.

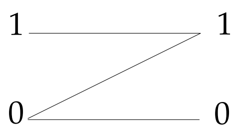
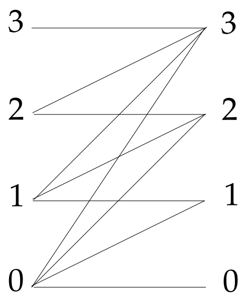
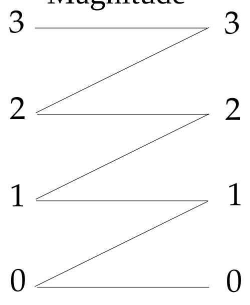
	Binary Asymmetric	q-ary Asymmetric	Asymmetric Limited-Magnitude
Channel			
Error Models	t Asym. errors [KF59]	t Asym. errors [Web92] Bounded L1-norm error [Var73]	All errors [AAK02] ***t* errors** (this chapter)

Figure 2.2: Asymmetric Channels and Error Models. At the top row are channel diagrams representing transitions with non-zero probabilities. At the bottom row are combinatorial error models derived from the corresponding channel models. The citations refer to the first work that considered each error model.

A natural application for asymmetric limited-magnitude error-correcting codes, and the primary motivator for their study here, is the ubiquitous *Flash* data-storage technology. The term Flash Memory or Flash Device refers to a Non-Volatile Memory (NVM) technology that is both electrically programmable and electrically erasable. This property, together with high storage densities and high speed programming, has made Flash Memory the dominant non-volatile memory technology and a prominent enabler for many portable applications and technologies. To scale the storage density of Flash memories, the *Multi-Level Flash Cell* concept is used to increase the number of stored bits in a cell [ER99]. Thus each Multi-Level Flash cell stores one of q levels and can be regarded as a symbol over a discrete alphabet of size q. The most conspicuous property of Flash storage is its inherent asymmetry between cell programming (charge placement) and cell erasing (charge removal). This asymmetry causes significant error sources to change cell levels in one dominant direction. Moreover, all reported common Flash error mechanisms induce errors whose magnitudes (the number of level changes) are significantly smaller than the overall programming window (the alphabet size). These two error characteristics combined, strongly motivate the model of asymmetric limited-magnitude errors studied in this chapter. In addition to the (uncontrolled) errors that challenge Flash Memory design and operation, codes for asymmetric limited-magnitude errors can be used to speed-up memory access by allowing less precise programming schemes that introduce errors in a controlled way. For a more detailed discussion of the ways Flash Memories can benefit from the new codes herein, please refer to section 2.7.

Asymmetric limited-magnitude error-correcting codes were proposed in [AAK02] for the special case $t = n$ (n is the code-block size). These codes follow Shannon's general method for achieving zero-error communication over noisy channels [Sha56], and they turn out to be a special case of the general construction method provided in this chapter.

The (all even/all odd) sample code described earlier in the chapter is one instantiation of a general construction method that provides codes for all possible code parameters. The main strength of this method is that for any target alphabet size (determined by the number of threshold levels), asymmetric limited-magnitude error correctability is inherited from *symmetric* error correctability of codes over alphabets of size $\ell + 1$ (in the case of the example above, it is the binary repetition code.). Thus a rich selection of known symmetric-error-correcting codes becomes handy to offer codes that are optimized for the asymmetric limited-magnitude channel. As a favorable by-product of the construction method, encoding and decoding of the resulting codes are performed on alphabets whose sizes depend only on ℓ, irrespective of the code alphabet (which may be much larger than ℓ). This is a major advantage in both redundancy and complexity, compared to other proposed codes for

Multi-level Flash memories (e.g. [GCKT03]), whose encoding and decoding are performed over the large code alphabet.

After discussing the asymmetric ℓ-limited-magnitude error model in Section 2.2, the main code construction is presented in Section 2.3, together with encoding and decoding procedures. Evaluation of the resulting codes is performed in Section 2.4, where asymptotic optimality is shown for $\ell = 1$ and for a general ℓ when t grows "slowly" relative to the code length n. A more conclusive optimality is shown by constructing codes that are "perfect" in the asymmetric ℓ-limited-magnitude error model. In addition, Section 2.4 compares the code sizes to sizes of codes for a related error model. Section 2.5 and Section 2.6 discuss extensions of the code construction with motivations from practical applications. Those include the construction of systematic codes (Section 2.5), and codes for simultaneous asymmetric and symmetric limited-magnitude errors (Section 2.6).

2.2 t Asymmetric ℓ-Limited-Magnitude Error-Correcting Codes

An alphabet Q of size q is defined as the set of integers modulo q: $\{0, 1, 2, \ldots, q-1\}$. For a codeword $\boldsymbol{x} \in Q^n$ and a channel output $\boldsymbol{y} \in Q^n$, the definition of asymmetric limited-magnitude errors now follows.

Definition 2.1 *A vector of integers $\boldsymbol{e} = (e_1, \ldots, e_n)$ is called a **t asymmetric ℓ-limited-magnitude error word** if $|\{i : e_i \neq 0\}| \leq t$, and for all i, $0 \leq e_i \leq \ell$.*

Given a codeword $\boldsymbol{x} \in Q^n$, a t asymmetric ℓ-limited-magnitude model outputs a vector $\boldsymbol{y} \in Q^n$, such that $\boldsymbol{x} + \boldsymbol{e} = \boldsymbol{y}$, and $\boldsymbol{e}$ is a t asymmetric ℓ-limited-magnitude error word. The $+$ symbol denotes addition over the reals. A generalization of the above definition is when we allow asymmetric errors to wrap around (from $q-1$ back to 0), whereby we interpret the $+$ symbol above as addition modulo q.

The q-ary asymmetric ℓ-limited-magnitude error model studied in this chapter is a generalization of the binary asymmetric error model studied by numerous authors (see [Klø81] for a detailed treatment of this model). Another generalization, proposed by Varshamov [Var73], studies q-ary asymmetric errors that have no magnitude limit for individual coordinates, but the sum of the error vector elements is bounded by some integer T. When $T = t\ell$, codes for the Varshamov model in particular correct t asymmetric ℓ-limited-magnitude errors. However, for many applications, such as Multi-Level Flash memories, the Varshamov model may be a too strong error model. These applications can greatly benefit from the constructions presented here, which give better codes in terms of size, and also enjoy simple encoding and decoding algorithms (the number-theoretic Varshamov codes have no efficient encoding or decoding algorithms).

The discussion of codes for the asymmetric ℓ-limited-magnitude channel model is commenced with the definition of a distance that captures the correctability of t asymmetric ℓ-limited-magnitude errors.

Definition 2.2 *For $\boldsymbol{x} = (x_1, \ldots, x_n) \in Q^n$ and $\boldsymbol{z} = (z_1, \ldots, z_n) \in Q^n$, define $N(\boldsymbol{x}, \boldsymbol{z}) = |\{i : x_i > z_i\}|$ and $N(\boldsymbol{z}, \boldsymbol{x}) = |\{i : x_i < z_i\}|$. **The distance d_ℓ** between the words $\boldsymbol{x}, \boldsymbol{z}$ is defined*

$$d_\ell(\boldsymbol{x}, \boldsymbol{z}) =$$
$$= \begin{cases} n+1 & \text{if } \max_i\{|x_i - z_i|\} > \ell \\ \max(N(\boldsymbol{x}, \boldsymbol{z}), N(\boldsymbol{z}, \boldsymbol{x})) & \text{otherwise} \end{cases}$$

The d_ℓ distance defined above allows to determine the number of ℓ-limited-magnitude errors, correctable by a code $\mathcal{C}$.

Proposition 2.1 *A code $\mathcal{C} \subset Q^n$ can correct t asymmetric ℓ-limited-magnitude errors if and only if $d_\ell(\boldsymbol{x}, \boldsymbol{z}) \geq t + 1$ for all distinct $\boldsymbol{x}, \boldsymbol{z}$ in $\mathcal{C}$.*

Proof: A code fails to correct a t asymmetric ℓ-limited-magnitude error word if and only if there exist two distinct codewords $\boldsymbol{x}$,$\boldsymbol{z}$ and two t asymmetric ℓ-limited-magnitude error words $\boldsymbol{e}$,$\boldsymbol{f}$, such that $\boldsymbol{x} + \boldsymbol{e} = \boldsymbol{z} + \boldsymbol{f}$, or equivalently, $\boldsymbol{x} - \boldsymbol{z} = \boldsymbol{f} - \boldsymbol{e}$.

($\Leftarrow$) Assume that for a pair $\boldsymbol{x}$,$\boldsymbol{z}$, $d_\ell(\boldsymbol{x}, \boldsymbol{z}) \geq t + 1$. Then at least one of the following holds:

1. $N(\boldsymbol{x}, \boldsymbol{z}) > t$ or $N(\boldsymbol{z}, \boldsymbol{x}) > t$

2. $|x_i - z_i| > \ell$ for at least one index $i \in \{1, \dots, n\}$.

Case 1 implies that $\boldsymbol{f} - \boldsymbol{e}$ has either more than t positive elements or more than t negative elements, none of which is possible by the definition of the error vectors $\boldsymbol{e}$,$\boldsymbol{f}$.

Case 2 implies that for some i, either $e_i > \ell$ or $f_i > \ell$, both impossible by the definition of $\boldsymbol{e}$,$\boldsymbol{f}$.

Since the same arguments apply to any $\boldsymbol{x}$,$\boldsymbol{z}$ in the code, the code necessarily corrects all possible t asymmetric ℓ-limited-magnitude errors.

$(\Rightarrow)$ Assume there exist a pair of codewords $\boldsymbol{x}$,$\boldsymbol{z}$, for which $d_\ell(\boldsymbol{x}, \boldsymbol{z}) \leq t < n$. Then both $N(\boldsymbol{x}, \boldsymbol{z}) \leq t$ and $N(\boldsymbol{z}, \boldsymbol{x}) \leq t$ are true, and $|x_i - z_i| \leq \ell$ at all indices i. In that case we can set $f_i = x_i - z_i$ at all indices i such that $x_i > z_i$ and $e_i = z_i - x_i$ at all indices i such that $z_i > x_i$. With zeros at all other indices, such $\boldsymbol{e}$,$\boldsymbol{f}$ satisfy $\boldsymbol{x} - \boldsymbol{z} = \boldsymbol{f} - \boldsymbol{e}$ without violating the conditions of t asymmetric ℓ-limited-magnitude errors. $\square$

Although the asymmetric ℓ-limited-magnitude distance measure d_ℓ is not a metric, i.e. the triangle inequality does not hold, it still provides a necessary and sufficient condition for the correctability of asymmetric ℓ-limited-magnitude errors. In subsequent sections, it will be used both to prove the correction capability of code constructions, and to obtain upper bounds on the size of codes.

2.3 Construction of t Asymmetric ℓ-Limited-Magnitude Error-Correcting Codes

We now provide the main construction of the chapter. For notational convenience, given $\boldsymbol{x} = (x_1, \dots, x_n)$, the vector $(x_1 \bmod q', x_2 \bmod q', \dots, x_n \bmod q')$ will be denoted by $\boldsymbol{x} \bmod q'$. To obtain a code over alphabet Q that corrects t or less asymmetric errors of ℓ-limited-magnitude, one can use codes for symmetric errors over small alphabets as follows.

Construction 2.1 *Let Σ be a code over the alphabet Q' of size $q' = \ell + 1$. The code $\mathcal{C}$*

over the alphabet Q of size q ($q > \ell + 1$) is defined as

$$\mathcal{C} = \{\boldsymbol{x} = (x_1, \ldots, x_n) \in Q^n : \boldsymbol{x} \bmod (\ell + 1) \in \Sigma\}. \tag{2.1}$$

In words, the codewords of $\mathcal{C}$ are the subset of the words of Q^n that are mapped to codewords of Σ, when their symbols are reduced modulo $q' = \ell + 1$.

Codes obtained by Construction 2.1 have the following error correction capability.

Theorem 2.2 *$\mathcal{C}$ corrects t asymmetric ℓ-limited-magnitude errors if Σ corrects t symmetric errors. If $q > 2\ell$,[1] the converse is true as well.*

Proof: The proof proceeds by showing that any pair of codewords $\boldsymbol{x}, \boldsymbol{z}$ in $\mathcal{C}$ is at d_ℓ distance of at most $t + 1$ apart. By Proposition 2.1, this would conclude that $\mathcal{C}$ corrects all t asymmetric ℓ-limited-magnitude errors. We distinguish between two cases.
1. $\boldsymbol{x} \bmod (\ell + 1) = \boldsymbol{z} \bmod (\ell + 1)$
2. $\boldsymbol{x} \bmod (\ell + 1) \neq \boldsymbol{z} \bmod (\ell + 1)$

Since $\boldsymbol{x} \neq \boldsymbol{z}$, Case 1 implies that for at least one index $i \in \{1, \ldots, n\}$, $|x_i - z_i| > \ell$, settling their d_ℓ distance to be $n + 1$.

Case 2, and the fact that Σ has minimum Hamming distance of at least $2t+1$, imply that $\boldsymbol{x}$ and $\boldsymbol{z}$ differ in at least $2t + 1$ locations and thus, in particular, $\max(N(\boldsymbol{x}, \boldsymbol{z}), N(\boldsymbol{z}, \boldsymbol{x})) \geq t + 1$.

For the converse, if Σ does not correct all t symmetric errors, then there exists a quadruple $(\boldsymbol{\chi} \in \Sigma, \boldsymbol{\zeta} \in \Sigma, \boldsymbol{e}, \boldsymbol{f})$, such that $\boldsymbol{\chi} + \boldsymbol{e} = \boldsymbol{\zeta} + \boldsymbol{f} \pmod{\ell + 1}$, and $\boldsymbol{e}, \boldsymbol{f}$ are t asymmetric ℓ-limited-magnitude error vectors. Therefore, the vectors $\boldsymbol{x} = \boldsymbol{\chi} + (\ell + 1) \cdot \Delta(\boldsymbol{\zeta} + \boldsymbol{f} - \boldsymbol{\chi} - \boldsymbol{e})$ and $\boldsymbol{z} = \boldsymbol{\zeta} + (\ell + 1) \cdot \Delta(\boldsymbol{\chi} + \boldsymbol{e} - \boldsymbol{\zeta} - \boldsymbol{f})$, (where $\Delta(\boldsymbol{v})$ is a vector with ones where $v_i > 0$ and zeros elsewhere), are codewords of $\mathcal{C}$ and they satisfy $\boldsymbol{x} + \boldsymbol{e} = \boldsymbol{z} + \boldsymbol{f}$. Since $q > 2\ell$, the last sum is a valid channel output. We conclude that there exists an uncorrectable error word for $\mathcal{C}$, and the converse follows. $\square$

[1]a reasonable assumption since the best codes are obtained when $q >> q'$

Construction 2.1 is clearly useful as it leverages the comprehensively studied theory of codes for symmetric errors, to obtain codes for asymmetric limited-magnitude errors. However, Construction 2.1 is a special case of the following construction.

Construction 2.1A. *Let Σ be a code over the alphabet Q' of size q'. The code $\mathcal{C}$ over the alphabet Q of size q ($q > q' > \ell$) is defined as*

$$\mathcal{C} = \{\boldsymbol{x} = (x_1, \ldots, x_n) \in Q^n : \boldsymbol{x} \bmod q' \in \Sigma\}. \tag{2.2}$$

The relationship between $\mathcal{C}$ and Σ in the general case are summarized below. The proof is almost identical to that of Theorem 2.2.

Theorem 2.3 *$\mathcal{C}$ corrects t asymmetric ℓ-limited-magnitude errors if Σ corrects t asymmetric ℓ-limited-magnitude errors with wrap-around. If $q \geq q' + \ell$, the converse is true as well.*

Remark: If $q' \mid q$ then $\mathcal{C}$ corrects t asymmetric ℓ-limited-magnitude errors *with wrap-around* for Σ with the same properties as above.

It is easy to see how Construction 2.1 is a special case of Construction 2.1A. When $q' = \ell + 1$, an asymmetric ℓ-limited-magnitude error with wrap-around is equivalent to a symmetric error (with no magnitude limit).

2.3.1 Discussion and analysis of the code constructions

The size of the code $\mathcal{C}$ is bounded from below and from above by the following theorem.

Theorem 2.4 *The number of codewords in the code $\mathcal{C}$ is bounded by the following inequalities.*

$$\left\lfloor \frac{q}{q'} \right\rfloor^n \cdot |\Sigma| \leq |\mathcal{C}| \leq \left\lceil \frac{q}{q'} \right\rceil^n \cdot |\Sigma| \tag{2.3}$$

Proof: Let $\boldsymbol{\chi} = (\chi_1, \ldots, \chi_n)$ be a codeword of Σ. A valid codeword of $\mathcal{C}$ can be obtained by replacing each χ_i by any element of the set $\{x \in Q : x = \chi_i \pmod{q'}\}$. The size of this set is $\lceil q/q' \rceil$ if $\chi_i < q \bmod q'$ and $\lfloor q/q' \rfloor$ otherwise. Thus for any code Σ, the lower and upper bounds above follow. $\square$

In the special case when $q' = 2$, the size of $\mathcal{C}$ can be obtained exactly from the weight enumerator of Σ.

Theorem 2.5 *Let $q' = 2$ and Σ be a code over $Q' = \{0, 1\}$. Then the size of the code $\mathcal{C}$, as defined in* (2.2), *is given by*

$$|\mathcal{C}| = \sum_{w=0}^{n} A_w \left\lceil \frac{q}{2} \right\rceil^{n-w} \left\lfloor \frac{q}{2} \right\rfloor^{w}$$

where A_w is the number of codewords of Σ with Hamming weight w.

Proof: When $2 \mid q$, the right hand side equals $(q/2)^n \cdot |\Sigma|$, as the matching lower and upper bounds of (2.3) predict. When $2 \nmid q$, a 0 in χ can be replaced by $\lceil q/2 \rceil$ different symbols of Q and a 1 in χ can be replaced by $\lfloor q/2 \rfloor$ different symbols. Using the weight enumerator of Σ we obtain the exact value for the size of $\mathcal{C}$ above. $\square$

This theorem can be extended to $q' > 2$, but in such cases knowing the weight distribution of Σ does not suffice, and more detailed enumeration of the code is needed for an exact count.

The ℓ-AEC codes suggested in [AAK02], that correct all asymmetric ℓ-limited-magnitude errors, can also be regarded as a special case of this construction method. To show that, let $\mathbf{0}$ be the trivial length n code, over the alphabet Q' of size $\ell + 1$, that contains only the all-zero codeword. Define

$$\begin{aligned} \mathcal{C} &= \{\boldsymbol{x} \in Q^n : \boldsymbol{x} \bmod (\ell+1) \in \mathbf{0}\} \\ &= \{\boldsymbol{x} \in Q^n : x_i \equiv 0 \bmod (\ell+1) \text{ for } i = 1, 2, \ldots, n\} \text{ [AAK02]}. \end{aligned}$$

Since $\mathbf{0}$ can correct $t = n$ symmetric errors, $\mathcal{C}$ can correct $t = n$ asymmetric ℓ-limited-magnitude errors.

2.3.2 Decoding

The main construction of this chapter (Construction 2.1) reduces the problem of constructing asymmetric ℓ-limited-magnitude error-correcting codes, to the problem of constructing codes for symmetric errors. The correction capability of the code constructions was proved in section 2.3 using arguments on their minimum d_ℓ distance, arguments that have a non-algorithmic character. We next show that a similar reduction applies also to the algorithmic problem of efficiently decoding asymmetric ℓ-limited-magnitude error-correcting codes.

In the following, we describe how, given a decoding algorithm for the code Σ, one can obtain a decoder for the code $\mathcal{C}$, that has essentially the same decoding complexity, with only a few additional simple arithmetic operations. The decoding procedure herein refers to the more general Construction 2.1A, and it clearly applies to the special case of Construction 2.1 ($q' = \ell + 1$).

Let $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathcal{C}$ be a codeword and $\boldsymbol{y} = (y_1, \ldots, y_n) \in Q^n$ be the channel output when up to t asymmetric ℓ-limited-magnitude errors have occurred. Denote the corresponding Σ codeword by $\boldsymbol{\chi} = \boldsymbol{x} \bmod q'$, and also define $\boldsymbol{\psi} = \boldsymbol{y} \bmod q'$ and $\boldsymbol{\epsilon} = (\boldsymbol{\psi} - \boldsymbol{\chi}) \pmod{q'}$. First we observe that since $q' > \ell$, if $0 \le y_i - x_i \le \ell$ then $y_i - x_i = (y_i - x_i) \bmod q'$. Using the simple modular identity

$$\begin{aligned}(y_i - x_i) \bmod q' &= (y_i \bmod q' - x_i \bmod q') \bmod q' \\ &= (\psi_i - \chi_i) \bmod q' = \epsilon_i,\end{aligned}$$

we get that $y_i - x_i = \epsilon_i$, and in particular, if $0 \le y_i - x_i \le \ell$, then $0 \le \epsilon_i \le \ell$. In other words, if the codeword $\boldsymbol{x}$ over Q suffered an asymmetric ℓ-limited-magnitude error at location i, then the codeword $\boldsymbol{\chi}$ over Q' suffered an asymmetric ℓ-limited-magnitude error with wrap-around at the same location i, and with the same magnitude. Given at most t asymmetric ℓ-limited-magnitude errors with wrap-around, a decoder for Σ can recover $\boldsymbol{\epsilon}$ from $\boldsymbol{\psi}$. Thus, the equality $y_i - x_i = \epsilon_i$ allows the same decoder to recover $\boldsymbol{x}$ from $\boldsymbol{y}$.

A schematic decoder of an asymmetric ℓ-limited-magnitude error-correcting code $\mathcal{C}$ that uses a decoder for a symmetric error-correcting code Σ is given in Figure 2.3. Given a channel output $\boldsymbol{y} \in Q^n$, the decoder takes the symbol-wise modulo q' of $\boldsymbol{y}$ to obtain $\boldsymbol{\psi} \in {Q'}^n$. Then a decoder for Σ is invoked with the input $\boldsymbol{\psi}$ and an error estimate $\hat{\boldsymbol{\epsilon}}$ is obtained such that $\hat{\boldsymbol{\chi}} + \hat{\boldsymbol{\epsilon}} \equiv \boldsymbol{\psi} \bmod q'$, and $\hat{\boldsymbol{\chi}}$ is a codeword of Σ within the correction radius of the decoder for Σ. Note that the codeword estimate $\hat{\boldsymbol{\chi}}$ is discarded and not used for the decoding of $\mathcal{C}$. Finally, $\hat{\boldsymbol{\epsilon}}$ is subtracted from $\boldsymbol{y}$ to obtain the codeword estimate $\hat{\boldsymbol{x}} \in \mathcal{C}$.

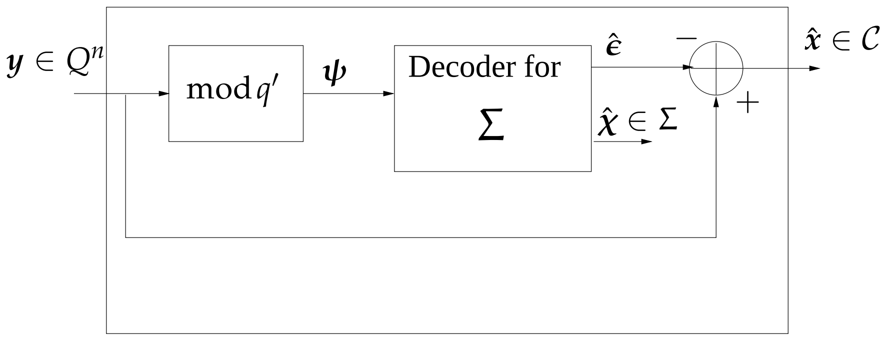

$\boldsymbol{y} \in Q^n$
$\bmod q'$
$\boldsymbol{\psi}$
Decoder for Σ
$\hat{\boldsymbol{\epsilon}}$
$-$
$+$
$\hat{\boldsymbol{\chi}} \in \Sigma$
$\hat{\boldsymbol{x}} \in \mathcal{C}$

Figure 2.3: Decoding asymmetric ℓ-limited-magnitude error-correcting codes. A decoder for Σ is run on the received symbols modulo q', and the error estimate of the decoder is subtracted from the original received word $\boldsymbol{y}$.

2.3.3 Encoding

Construction 2.1 (and 2.1A) defined the code $\mathcal{C}$ as a subset of Q^n, without specifying how information symbols are mapped to codewords. There are many ways to map information to codewords of $\mathcal{C}$, and the simplest one, that applies to any q, q' such that $q \mid q'$, is detailed below. For an alphabet of size $q = A \cdot q'$, where A, q' are integers, information is mapped to a length n codeword of $\mathcal{C}$ as follows. n symbols, $(a_1, \ldots, a_n)$, over the alphabet of size A are set as pure information symbols. Additionally, k information symbols over the alphabet of size q' are input to an encoder of Σ to obtain n symbols, $(\chi_1, \ldots, \chi_n)$, over the same alphabet. Finally, each code symbol x_i over Q is calculated by $a_i \cdot q' + \chi_i$.
Other encoding functions can map information symbols to codewords of $\mathcal{C}$ in a different way than the simple encoding function above. Different mappings with good properties are discussed in Section 2.5 and Section 2.6.

Example 2.1 now attempts to convey the main ideas of the encoding and decoding of asymmetric ℓ-limited-magnitude error-correcting codes.

Example 2.1 *Let Σ_H be the binary*[2] *Hamming code of length $n = 2^m - 1$, for some integer*

[2]Non-binary Hamming codes can be used as well when $\ell > 1$.

m. First we define the code $\mathcal{C}_H$ in the way of Construction 2.1.

$$\mathcal{C}_H = \{\boldsymbol{x} = (x_1, \ldots, x_n) \in Q^n : \boldsymbol{x} \bmod 2 \in \Sigma_H\}.$$

By the properties of Σ_H, the code $\mathcal{C}_H$ corrects a single asymmetric $\ell = 1$ limited-magnitude error. When the code alphabet size is $q = 2^b$, for some integer b, the perfect code $\mathcal{C}_H$, whose size equals $|\mathcal{C}_H| = A^n \cdot q'^{n-m} = 2^{(b-1)n} \cdot 2^{n-m} = 2^{nb-m}$ by equation (2.3)*, admits a simple function from $nb - m$ information bits to codewords of $\mathcal{C}_H$ over Q, as illustrated in Figure* 2.4 *below. In Figure* 2.4 *(a), $nb - m$ information bits are input to the encoder. The encoder then uses a binary Hamming encoder to encode $n - m$ of the information bits into a length n Hamming codeword (Figure* 2.4 *(b)). Finally, in Figure* 2.4 *(c), each q-ary symbol of the codeword $\boldsymbol{x} \in \mathcal{C}_H$ is constructed from b bits using the usual binary-to-integer conversion, the top row being the least-significant bits of $x_i \in Q$. Decoding is carried out*

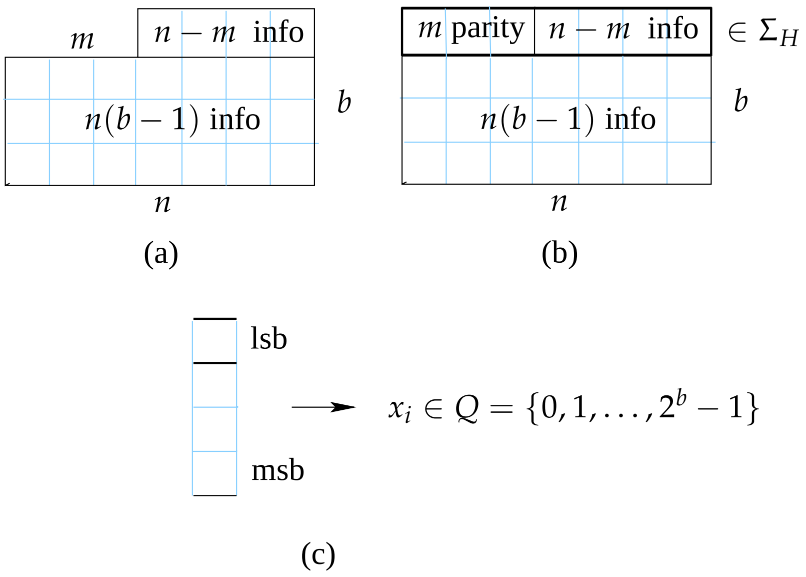


Figure 2.4: Encoding Procedure for $\mathcal{C}_H$. (a) Input information bits. (b) Encoding the top row of bits. (c) Mapping bit column vectors to symbols over the alphabet of the code $\mathcal{C}_H$.

by using a Hamming decoder on the top row to find the limited-magnitude error location and magnitude (for binary Hamming codes the magnitude is always 1*). The top row word is* not *corrected by the Hamming decoder, but rather the error magnitude is subtracted from the Q-ary word $\boldsymbol{y}$ to obtain a decoded codeword. To recover the information bits after* *decoding, the Q symbols are converted back to bits in the usual way, and the m parity bits are discarded.*

2.4 Optimality of the Code Construction and Comparison to Related Codes

2.4.1 Perfect Codes

For some parameters, the codes constructed in the previous section are the best conceivable ones for the asymmetric ℓ-limited-magnitude error model. These codes are *perfect* codes in the sense that they attain the sphere-packing bound for asymmetric ℓ-limited-magnitude errors. The q-ary symmetric sphere-packing bound is first generalized to asymmetric ℓ-limited-magnitude errors (with wrap-around), and then it is shown that asymmetric ℓ-limited-magnitude error-correcting codes that meet this bound with equality can be obtained by using other known perfect codes, e.g., perfect codes in the Hamming metric.

Theorem 2.6 *If $\mathcal{C}$ is a t asymmetric ℓ-limited-magnitude (with wrap-around) error-correcting code, of length n over an alphabet of size q, then*

$$|\mathcal{C}| \cdot \sum_{i=0}^{t} \binom{n}{i} \ell^i \leq q^n \tag{2.4}$$

Proof: The proof uses the same arguments as the proof for symmetric errors. Assume the number of $(\boldsymbol{x}, \boldsymbol{\epsilon})$ pairs exceeds q^n, where $\boldsymbol{x}$ is a codeword and $\boldsymbol{\epsilon}$ is a t asymmetric ℓ-limited-magnitude error word. Then there exists a vector $\boldsymbol{y} \in Q^n$ such that

$$\boldsymbol{y} = \boldsymbol{x} + \boldsymbol{\epsilon} = \boldsymbol{x}' + \boldsymbol{\epsilon}'$$

where either $\boldsymbol{x} \neq \boldsymbol{x}'$ or $\boldsymbol{\epsilon} \neq \boldsymbol{\epsilon}'$ (or both). If $\boldsymbol{x} \neq \boldsymbol{x}'$ then we have a contradiction since given $\boldsymbol{y}$ the decoder will not be able to distinguish between $\boldsymbol{x}$ and $\boldsymbol{x}'$. If $\boldsymbol{x} = \boldsymbol{x}'$, the additive property of the channel implies $\boldsymbol{\epsilon} = \boldsymbol{\epsilon}'$ as well, in contradiction to the assumption that $(\boldsymbol{x}, \boldsymbol{\epsilon}) \neq (\boldsymbol{x}', \boldsymbol{\epsilon}')$. Therefore the product of the code size and the number of admissible errors cannot exceed q^n which gives the bound provided. $\square$

Perfect t asymmetric ℓ-limited-magnitude error-correcting codes are obtained through the following proposition.

Proposition 2.7 *If there exists a perfect asymmetric ℓ-limited-magnitude code over an alphabet of size q', then there exists a perfect asymmetric ℓ-limited-magnitude code with the same length, over an alphabet of any size q, such that $q' \mid q$, that corrects the same number of errors.*

Proof:

Let $\mathcal{C}$ and Σ be as in Construction 2.1A. We first substitute the expression for the code size from (2.3) into the left side of the sphere packing bound

$$|\mathcal{C}| \cdot \sum_{i=0}^{t} \binom{n}{i} \ell^i = \left(\frac{q}{q'}\right)^n \cdot |\Sigma| \cdot \sum_{i=0}^{t} \binom{n}{i} \ell^i.$$

If the code Σ over the alphabet of size q' is perfect, then its size satisfies

$$|\Sigma| \cdot \sum_{i=0}^{t} \binom{n}{i} \ell^i = q'^n$$

Substituting the latter into the former we get

$$|\mathcal{C}| \cdot \sum_{i=0}^{t} \binom{n}{i} \ell^i = \left(\frac{q}{q'}\right)^n \cdot q'^n = q^n,$$

which completes the proof.

$\square$

Alternatively, perfect codes are codes which induce a partition of the space into error spheres. As was already noted, when $q' = \ell + 1$, the t asymmetric ℓ-limited-magnitude error sphere coincides with the Hamming metric t symmetric error sphere. Thus, taking Σ to be a perfect code in the Hamming metric (e.g., Hamming or Golay codes), produces perfect asymmetric ℓ-limited-magnitude error-correcting codes over an alphabet of size q, where $q' \mid q$.

Other perfect codes may exist even when $q' \neq \ell + 1$. For example, when $t = 1$, the asymmetric ℓ-limited-magnitude error sphere is the semi-cross examined by Stein in [Ste84].

One may wonder if any *inherently* new perfect code are produced by Construction 2.1A. The answer, unfortunately, is no: Construction 2.1A simply takes translations of the tiling provided by the base code Σ to accommodate for the larger alphabet. This is depicted in the following example.

Example 2.2 *Let Σ be the perfect ternary length $n = 2$ code capable of correcting one asymmetric 1-limited-magnitude error, $\Sigma = \{00, 11, 22\}$. The code induces a tiling of $\mathbb{Z}_3^2$ with the error sphere, and is shown in Figure 2.5. Since this tiling is with wrap-around, it also induces a natural tiling with wrap-around of $\mathbb{Z}_{3k}^2$ for every $k \in \mathbb{N}$. Specifically, for $\mathcal{C}$, the code over an alphabet of size 6 produced from Σ by Construction 2.1A, the resulting tiling is also shown in Figure 2.5.*

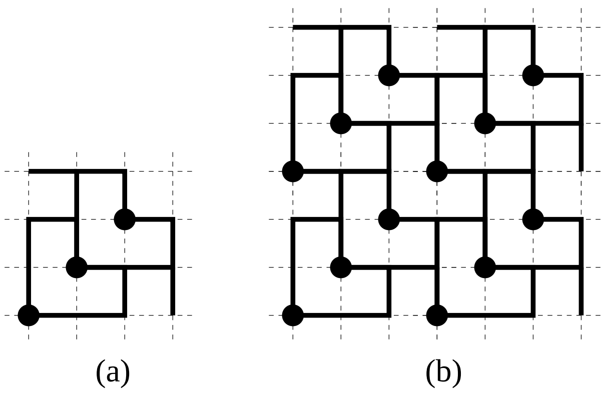


Figure 2.5: Asymmetric limited-magnitude error-correcting codes as tilings. In Example 2.2, the tilings induced by (a) the code Σ, and (b) the code $\mathcal{C}$.

2.4.2 Asymptotic optimality of Construction 2.1

The implication of Construction 2.1 is that "large" codes for symmetric errors over an alphabet of size $\ell + 1$ imply "large" codes for asymmetric ℓ-limited-magnitude errors over any larger alphabet. Showing the reverse implication, namely that "large" codes for asymmetric ℓ-limited-magnitude errors imply "large" codes for symmetric errors, would conclude that Construction 2.1 is optimal in terms of the resulting code sizes. Optimality is achieved in this case since given the “large” code for symmetric errors implied by the reverse direction, Construction 2.1 can be invoked to yield code of the same size as the original “large” code for asymmetric ℓ-limited-magnitude errors. The purpose of this subsection is to show that asymptotically, Construction 2.1 gives the largest possible codes for asymmetric ℓ-limited-magnitude errors.

Definition 2.3 *Define the **rate** R of a code $\mathcal{C}$ of length n over an alphabet of size q as*

$$R = \frac{1}{n} \log_q |\mathcal{C}|$$

where $|\mathcal{C}|$ is the number of codewords in $\mathcal{C}$.

Theorem 2.8 *If $\tilde{\mathcal{C}}$ is a t asymmetric ℓ-limited-magnitude error-correcting code with rate R and block-length n that tends to infinity, then*
1) When $\ell = 1$ and for arbitrary t, there exists a code $\mathcal{C}$, constructed by Construction 2.1, with the same rate R.
2) For a general ℓ and for $t = o(n/\log n)$ (i.e. $\lim_{n\to\infty} t \log n/n = 0$: t has a slower growth compared to $n/\log n$), there exists a code $\mathcal{C}$, constructed by Construction 2.1, with the same rate R.

Proof: We first give an overview of the proof technique using the diagram of Figure 2.6. Construction 2.1 allows obtaining codes for asymmetric ℓ-limited magnitude errors from codes for symmetric errors with an inflation factor $K_{q,\ell}(n) \triangleq q^n/(\ell+1)^n$ (upper part of Figure 2.6). To prove that the construction is optimal, we need to prove the converse – that codes for symmetric errors can be obtained from codes for asymmetric ℓ-limited magnitude errors with asymptotically equivalent deflation factor (lower part of Figure 2.6). The way this is done in the proof is by first proving the existence of codes for *asymmetric* errors with deflation factor $K_{q,\ell}(n)$, and then showing that codes for symmetric errors are equivalent in rate to codes for asymmetric errors. The composition of these two steps establishes the asymptotic rate optimality of Construction 2.1. To exercise the proof ideas above we introduce the following notation. Let $\mathrm{A}\ell M_a(n,t)$ be the size of the largest length n code that corrects t asymmetric ℓ-limited-magnitude errors over an alphabet of size a.

$$\begin{tikzcd}
 & \boxed{\text{Symmetric}} & & \boxed{\text{A}\ell\text{M}} \\
\text{Construction:} & {|\Sigma|} \arrow[rr, Rightarrow, "\times K_{q,\ell}(n)"] & & {|\mathcal{C}|} \\
\text{Converse:} & {|\Sigma|} & {|\Sigma_A|} \arrow[l, Leftarrow, "{\begin{array}{c}/const(n)\\ \text{or}\\ /poly(n)\end{array}}"'] & {|\mathcal{C}|} \arrow[l, Leftarrow, "/K_{q,\ell}(n)"'] \\
 & & \boxed{\text{Asymmetric}} &
\end{tikzcd}$$

Figure 2.6: Idea of rate-optimality proof. The converse at the lower part of the diagram is proved in two steps. First large codes for asymmetric errors are proved to exist; then small (constant or polynomial) gaps between codes for asymmetric and symmetric errors are proved.

Let $\mathsf{A}sym_a(n,t)$ be the size of the largest length n code that corrects t asymmetric errors (symbols change only in the upward direction, with no magnitude limit), over an alphabet of size a. Finally, let $\mathsf{A}_a(n,t)$ be the size of the largest length n code that corrects t symmetric errors, over an alphabet of size a. $\mathsf{A}_a(n,t)$ used here is a replacement of the more commonly used $A_a(n,d)$ [HP03, Ch.2], whereby the parameter d stands for the minimum Hamming distance of the code instead of the number of correctable symmetric errors (therefore $\mathsf{A}_a(n,t) = A_a(n, 2t+1)$).

To avoid the excessive use of the $\lceil\cdot\rceil$ operator, assume that $(\ell+1) \mid q$. The set of all q^n words over the alphabet of size q is partitioned into $q^n/(\ell+1)^n$ subsets, each of size $(\ell+1)^n$, as follows. Each subset contains a single word whose symbol-wise modulo $\ell+1$ equals the all zero vector. In addition to this vector, the subset contains the sum of that vector with all $(\ell+1)^n - 1$ non-zero vectors over the alphabet of size $\ell+1$. Each subset has the property that no two words within it differ in any coordinate by more than ℓ. A sample such partition for $n=2$, $q=4$ and $\ell=1$ is given below.

0 0	0 2	2 0	2 2
0 1	0 3	2 1	2 3
1 0	1 2	3 0	3 2
1 1	1 3	3 1	3 3

This property is equivalent to having $d_\ell(\boldsymbol{x},\boldsymbol{z}) < n+1$ for every $\boldsymbol{x},\boldsymbol{z}$ in the subset.

Suppose there is a code $\tilde{\mathcal{C}}$ that corrects t asymmetric ℓ-limited-magnitude errors. Then there exists at least one subset, with at least $|\tilde{\mathcal{C}}|(\ell+1)^n/q^n$ codewords of $\tilde{\mathcal{C}}$. Since any two codewords $\boldsymbol{x},\boldsymbol{z}$ in that subset satisfy $d_\ell(\boldsymbol{x},\boldsymbol{z}) < n+1$, each such pair has to satisfy $\max(N(\boldsymbol{x},\boldsymbol{z}),N(\boldsymbol{z},\boldsymbol{x})) > t$. In other words, the codewords of $\tilde{\mathcal{C}}$ that belong to the same subset, form a code that corrects t asymmetric errors with no magnitude limit of size at least $|\tilde{\mathcal{C}}|(\ell+1)^n/q^n$. Without loss of generality, the subset with many codewords is the one that contains the all zero codeword. Generality is maintained since neither $N(\boldsymbol{x},\boldsymbol{z})$ nor $N(\boldsymbol{z},\boldsymbol{x})$ are changed when a constant vector is subtracted from both $\boldsymbol{x}$ and $\boldsymbol{z}$. Consequently, the codewords of this subset imply the existence of a code over an alphabet of size $\ell+1$ that corrects t asymmetric errors with no magnitude limit. Formally,

$$\mathsf{A}sym_{\ell+1}(n,t) \geq \left(\tfrac{\ell+1}{q}\right)^n \mathsf{A}\ell M_q(n,t)$$

on the other hand, Construction 2.1 and Theorem 2.4 provide the following lower bound on $\mathsf{A}\ell M_q(n,t)$:

$$\mathsf{A}\ell M_q(n,t) \geq (\tfrac{q}{\ell+1})^n \mathsf{A}_{\ell+1}(n,t)$$

Combining the lower and upper bounds we obtain

$$\mathsf{A}_{\ell+1}(n,t) \leq \left(\tfrac{\ell+1}{q}\right)^n \mathsf{A}\ell M_q(n,t) \leq \mathsf{A}sym_{\ell+1}(n,t) \tag{2.5}$$

which is consistent with the trivial inequality $\mathsf{A}_{\ell+1}(n,t) \leq \mathsf{A}sym_{\ell+1}(n,t)$ (any code for symmetric errors is also a code for asymmetric errors). The proof of the theorem is achieved by bounding the gap between $\mathsf{A}_{\ell+1}(n,t)$ and $\mathsf{A}sym_{\ell+1}(n,t)$ using the following lemmas.

Lemma 2.9 *[Bor81]:* $\mathsf{A}_2(n,t) \geq \frac{1}{t+1}\mathsf{A}sym_2(n,t)$.

Proof: See [Kløl81]. □

Lemma 2.10 $\mathsf{A}_{\ell+1}(n,t) \geq \frac{1}{(n\ell)^{2t}}\mathsf{A}sym_{\ell+1}(n,t)$.

Proof: We will show that a code for symmetric errors can be obtained from a code for asymmetric errors by expurgating all but at least a $1/(n\ell)^{2t}$ fraction of codewords of the asymmetric-error-correcting code. Any two codewords in a t asymmetric-error-correcting code have Hamming distance of at least $t+1$. Any two codewords in a t symmetric-error-correcting code have Hamming distance of at least $2t+1$. The number of words (and in particular, an upper bound on the number of codewords) that are at distance between $t+1$ and $2t$ from a codeword of a t asymmetric-error-correcting code is

$$\sum_{i=t+1}^{2t} \binom{n}{i} \ell^i = \ell^t \sum_{i=1}^{t} \binom{n}{t+i} \ell^i$$

Since $\binom{n}{t+i} < n^{t+i}/t$,

$$\ell^t \sum_{i=1}^{t} \binom{n}{t+i} \ell^i < (n\ell)^{2t}$$

and thus expurgating all but at least $1/(n\ell)^{2t}$ of the codewords, yields a code for t symmetric errors:

$$\mathsf{A}_{\ell+1}(n,t) > \frac{1}{(n\ell)^{2t}} \mathsf{A}sym_{\ell+1}(n,t)$$

$\square$

Combining Lemma 2.9 with (2.5), for $\ell = 1$ we obtain

$$\left(\frac{q}{2}\right)^n \mathsf{A}_2(n,t) \leq \mathsf{A}\ell M_q(n,t) \leq n \left(\frac{q}{2}\right)^n \mathsf{A}_2(n,t)$$

While Lemma 2.10 end (2.5) imply, for general ℓ,

$$(\tfrac{q}{\ell+1})^n \mathsf{A}_{\ell+1}(n,t) \leq \mathsf{A}\ell M_q(n,t) \leq (n\ell)^{2t} (\tfrac{q}{\ell+1})^n \mathsf{A}_{\ell+1}(n,t)$$

Taking the logarithm, dividing by n and taking the limit $n \to \infty$, the upper and lower bounds of $\mathsf{A}\ell M_q(n,t)$ are identical for both $\ell = 1$ and for general ℓ (under the restrictions on t of part 2 of the theorem). Hence asymmetric ℓ-limited-magnitude codes obtained from symmetric codes by Construction 2.1 are asymptotically optimal. $\square$

2.4.3 Comparison to Varshamov codes

Prior to this chapter's introduction of the t asymmetric ℓ-limited-magnitude error model, the closest error model that achieves this correction capability is the q-ary asymmetric error model proposed by Varshamov [Var73]. In particular, the known codes for the Varshamov model are better than known codes for symmetric errors. According to the Varshamov model, parametrized by an integer parameter T, if a vector $\boldsymbol{x} = (x_1, \ldots, x_n)$ over Q^n is transmitted, the channel output is a vector $\boldsymbol{x} + \boldsymbol{e}$ over Q^n, such that $e_i \geq 0$ and $\sum_{i=1}^{n} e_i \leq T$ (the addition and summation are over the reals). When $T = t\ell$, a T error correcting code for the Varshamov channel is also a t asymmetric ℓ-limited-magnitude error-correcting code. Since the $T = t\ell$ Varshamov model allows errors that are not allowed by the t asymmetric ℓ-limited-magnitude channel (i.e. errors with high magnitudes, which are unlikely in the target application), we expect the code constructions of this chapter to yield better codes compared to the best known Varshamov codes. This section thus compares between sizes of codes that are obtained using Construction 2.1, and lower bounds, provided in [McE73], on the sizes of various Varshamov codes. This comparison is incomplete since it only discusses the *sizes* of the codes. Evidently, t asymmetric ℓ-limited-magnitude codes enjoy efficient encoding and decoding procedures, a property which Varshamov codes are not known to have in general. We also do not discuss the restrictions on the block sizes n of the code constructions, in order to avoid overloading the discussion with secondary details.

2.4.3.1 Comparison for $\ell = 1$

When the asymmetric errors have a magnitude limit of $\ell = 1$, we compare the codes of Construction 2.1 to Varshamov codes with $T = t$. When $t = 1$, the two error models are identical and both constructions yield codes that are perfect in that metric, whose sizes are $q^n/(n+1)$. When $t = 2$, Varshamov codes are known to have $q^n/(n^2+n+1)$ codewords, while using the (punctured) Preparata codes [MS77, Ch.15] in Construction 2.1 gives $2q^n/(n+1)^2$, roughly twice as many codewords. For a general t, there exist Varshamov codes with sizes $q^n/(n+1)^t$. If we apply Construction 2.1 with BCH codes with designed distance $2t+1$, we get the same code size. However, using the Goppa codes [MS77, Ch.12] instead gives a superior size of q^n/n^t codewords.

2.4.3.2 Comparison for a general ℓ

While for $\ell = 1$ the advantage of the codes for asymmetric ℓ-limited-magnitude errors in terms of the code sizes is small, for larger ℓ values these codes are significantly larger than Varshamov codes. Even if we only use $(\ell+1)$-ary BCH codes in Construction 2.1, codes of sizes $q^n/(n+1)^{t'}$ are obtained, where $t' = 2t\ell/(\ell+1)$. Comparing to $q^n/(n+1)^{\ell t}$ of Varshamov codes shows a significant advantage to the favor of Construction 2.1 since $t' \leq \min(\ell t, 2t)$.

2.5 Systematic Asymmetric Limited-Magnitude Error-Correcting Codes

All its advantages notwithstanding, Construction 2.1 suffers the shortcoming of not admitting a systematic representation over Q. A code $\mathcal{C}$ over an alphabet Q is said to be in systematic form if its coordinates $\{x_1, \ldots, x_n\}$ can be partitioned into an information set $I = \{x_1, \ldots, x_k\}$ and a parity set $P = \{x_{k+1}, \ldots, x_n\}$, such that each symbol in P is a function of symbols in I only. As seen in Figure 2.4(b), m code symbols contain parity contribution. Each of these m symbols also has a pure-information component, so it can neither belong to the P set, nor to the I set of a systematic-code coordinate set. This non-systematic structure implies that “many” code symbols contain some parity contribution: a bad property in practice as it dictates accessing many Flash cells for each information update. In this section we propose *systematic* asymmetric limited-magnitude error-correcting codes that have fewer parity symbols.

2.5.1 Systematic codes for $\ell = 1$ limited-magnitude errors

When the error magnitude ℓ is bounded by 1, the code Σ in Construction 2.1 is a binary code. As we show next for this case, a modification of any code $\mathcal{C}$ can be carried out, that yields a systematic code with the same correction capability. The construction method of systematic codes for $\ell = 1$ is first presented in Example 2.3.

Example 2.3 *In this example we propose a systematic variant to the code $\mathcal{C}_H$, given in Example* 2.1. *The encoding function given below generates a code that has the same correction capabilities as $\mathcal{C}_H$, namely any single $\ell = 1$ asymmetric error is correctable, though the resulting code is different. Specifically, the dimensions of the systematic code are different. For this example we assume that the alphabet size of the code is 2^m (m – the number of parity bits in the binary code), compared to 2^b for arbitrary b in $\mathcal{C}_H$. This assumption can be lifted with a small increase in redundancy that depends on the actual code parameters. For an $[n, k = n - m]$ binary Hamming code Σ_H, the length of the systematic code is $n - m + 1$, compared to n in the non-systematic case. The systematic code is encoded as follows. In Figure* 2.7 *(a), km information bits are input to the encoder. The encoder then uses a binary Hamming encoder to encode the k information bits of the top row into a length $n = k + m$ Hamming codeword (Figure* 2.7 *(b)). The parity bits of the Hamming codeword are now placed as a separate column. The mapping of bits to Q symbols, shown in Figure* 2.7 *(c), is the usual (positional) mapping for the k information symbols and the Gray mapping for the parity symbol.*

To decode, a word of Q^{k+1} is converted back to bits using the same mappings, and a binary Hamming decoder is invoked for the n coded bits. By construction, a single $\ell = 1$ asymmetric error over Q translates to a single bit error in the Hamming codeword: in the k information symbols, an $\ell = 1$ error flips the least-significant bit that is part of the Hamming codeword, and in the parity symbol, an $\ell = 1$ error flips exactly one parity bit in the column, thanks to the Gray code used in the mapping.

The code proposed in Example 2.3, together with its encoding/decoding, can be generalized to any $\ell = 1$ limited-magnitude t asymmetric error-correcting code as stated by the following proposition.

Proposition 2.11 *Let Σ be a binary systematic code of length n and $m \leq b \cdot r$ parity bits, for any two integers r and $b > 1$. If Σ corrects t symmetric errors, then it can be used to construct a* systematic *t asymmetric $\ell = 1$ limited-magnitude error-correcting code over*

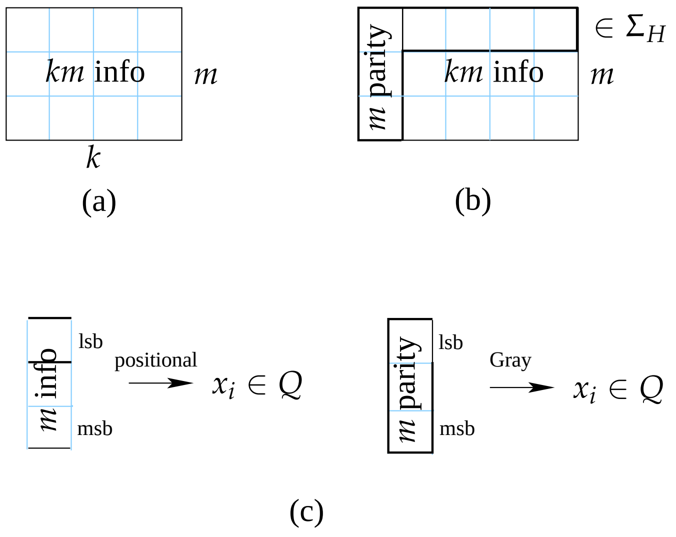


Figure 2.7: Encoding procedure for a systematic code with $\ell = 1$. (a) Input information bits. (b) Encoding the top row of information bits followed by the placement of parity bits in a separate column. (c) Mapping bit column vectors to symbols over the alphabet of the code. The standard positional mapping for information columns and the Gray mapping for parity columns.

an alphabet of size $q = 2^b$. This code has length $n - m + r$, of which r symbols are parity symbols.

Proof: The general construction follows closely the one in Example 2.3. $n - m$ information bits are used to encode a codeword of Σ. The $m \leq br$ parity bits are grouped into r columns of b bits each. Then these r columns are mapped to Q symbols using the Gray mapping and information bits are mapped to symbols using the positional mapping. The property that each limited-magnitude error results in one symmetric error in the codeword of Σ is preserved for this general case. □

2.5.2 Systematic codes for $\ell > 1$ limited-magnitude errors

If we try to extend the construction of the previous sub-section to codes for $\ell > 1$ limited-magnitude errors, we immediately face a stumbling block. Although generalized Gray codes exist for non-binary alphabets, their properties do not suffice to guarantee a similar general construction. The crucial property, that a single asymmetric limited-magnitude error translates to a single symmetric error in the $(\ell+1)$-ary code, is lost for the general case. For example, if for $\ell = 2$ a symbol represents the ternary reflected Gray codeword 0001, then an error of magnitude 2 will result in the Gray codeword 0012, whose Hamming distance to 0001 is 2 and not 1 as required. Thus, a limited-magnitude error at this symbol may induce 2 errors for the ternary code Σ. Evidently, this effect is not unique to the $(\ell+1)$-ary reflected Gray code, and there is no mapping between q-ary symbols $\{0, 1, \ldots, (\ell+1)^b - 1\}$ and $(\ell+1)$-ary b-tuples with this property. This sub-section proposes a construction for systematic asymmetric ℓ-limited-magnitude error-correcting codes, for arbitrary ℓ.

The construction of systematic asymmetric ℓ-limited-magnitude error-correcting codes builds on the non-systematic Construction 2.1. Two modifications of Construction 2.1 need to be instituted to yield a systematic code. The first is using a code Σ' that has different correction properties than Σ used before. The second is a special mapping between parity symbols of Σ' and code symbols of $\mathcal{C}'$ over Q.

Let q and $q' = \ell + 1$ be the alphabet sizes of the codes $\mathcal{C}'$ and Σ, respectively. Assume for simplicity that $q = 2(\ell+1)^s$, for some integer s. If this is not the case, the same construction can still be used, only the mappings between Q' and Q will be slightly more complicated.

The code Σ'

Let Σ be a linear systematic code over an alphabet of size $q' = \ell + 1$. The number of information symbols of Σ is denoted κ, and the number of parity symbols is m. The parity-check matrix of Σ is denoted by H. Columns $\{0, \ldots, m-1\}$ of H correspond to the m parity symbols of the code Σ. Let H' be the parity-check matrix that is obtained from H by replicating all columns in $i \in \{0, \ldots, m-1\}$ such that $i \not\equiv 0 \pmod{s}$, and appending them to H. H' is the parity-check matrix of the linear code Σ' that has m parity symbols and $\kappa + \lfloor m(s-1)/s \rfloor$ information symbols.

The mapping $Q' \leftrightarrow Q$ for parity symbols

From the m parity symbols of Σ', each group of s parity symbols, denoted $\phi_0^{(j)}, \ldots, \phi_{s-1}^{(j)}$, is mapped to a single parity symbol of $\mathcal{C}'$ using the following formula

$$x_j = \phi_0^{(j)} + 2 \sum_{i=1}^{s-1} \phi_i^{(j)} (\ell+1)^i. \qquad (2.6)$$

The systematic code $\mathcal{C}'$ is now specified using its encoding function.

Construction 2.2 *Let Σ be a $[\kappa + m, \kappa]$ linear code over the alphabet Q' of size $q' = \ell + 1$. The systematic code $\mathcal{C}'$ over the alphabet Q of size $q = 2(\ell+1)^s$ has $\kappa + \lfloor m(s-1)/s \rfloor$ information symbols and $\lceil m/s \rceil$ parity symbols. The parity symbols are computed by taking the modulo $\ell + 1$ of the information symbols, encoding them using a systematic encoder for Σ', and mapping the resulting m parity symbols over Q' to $\lceil m/s \rceil$ symbols over Q, as described in* (2.6).

Note that the length of the code $\mathcal{C}'$ is $\kappa + \lfloor m(s-1)/s \rfloor + \lceil m/s \rceil = \kappa + m$, the length of the non-systematic code $\mathcal{C}$.
Codes obtained by Construction 2.2 have the following error correction capability.

Theorem 2.12 *$\mathcal{C}'$ corrects t asymmetric ℓ-limited-magnitude errors if Σ corrects t symmetric errors.*

Proof: The key point in the proof is that an asymmetric ℓ-limited-magnitude error in a parity symbol j of $\mathcal{C}'$ may only change $\phi_0^{(j)}$ out of the s parity symbols $\phi_0^{(j)}, \ldots, \phi_{s-1}^{(j)}$ of Σ', mapped to this symbol. The way Σ' was extended from Σ allows correcting errors in the added information symbols, as long as the parity symbols whose H columns were replicated are guaranteed to be error free. This fact can be verified by using a decoder for Σ' that first computes the syndrome using H' and then inputs this syndrome to a decoder for Σ. Thus t or less asymmetric ℓ-limited-magnitude errors in any combination of information and parity symbols will result in a correctable error for the code Σ'. $\square$

To clarify Construction 2.2 an example is now provided.

Example 2.4 *Suppose we want to protect* 20 *information bits with a systematic code that corrects $t = 1$ asymmetric $\ell = 3$ limited magnitude error, over an alphabet of size $q = 32$. Since $t = 1$ and $\ell = 3$, we take Σ to be the quaternary Hamming code. More specifically, we choose Σ to be the $[5, 3]$ Hamming code over the alphabet of size $q' = 4$ whose parity-*

check matrix is given below.

$$H = \left[\begin{array}{cc|ccc} 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 2 & 3 \end{array}\right]$$

The $m = 2$ *left columns of* H *correspond to the parity symbols of* Σ. *Note that* $q = 2(q')^s$ *and* $s = 2$.

Replicating the right parity column we obtain H', *the parity-check matrix of* Σ'.

$$H' = \left[\begin{array}{cc|cccc} 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 2 & 3 \end{array}\right]$$

Encoding

The encoding of 20 *bits of information into a codeword of a systematic code* $\mathcal{C}'$ *with the specified parameters is described in Figure* 2.8. *Shaded cells represent parity symbols and unshaded cells represent information symbols. In Figure* 2.8*(a), the top two bit rows are used to encode a word of* Σ' *over the Finite Field of size* 4. *In the right part of Figure* 2.8*(b), information bits are mapped to symbols of* Q *using the usual binary to integer conversion. In the left part, the parity symbols of* Σ' *are mapped to a symbol of* Q *using the mapping defined in equation* (2.6). *Figure* 2.8*(c) shows the final codeword of* $\mathcal{C}'$.

As implied by the constant 2 in equation (2.6), only half of the alphabet Q is used in the parity symbols. That is equivalent to 1 extra redundant *bit* for each parity symbol of $\mathcal{C}'$. Note that the half factor is true for arbitrary ℓ. Whenever $\ell > 1$, that amount of additional redundancy compares favorably to using the Ahlswede et al. "all error correcting" scheme [AAK02] for the parity symbols, which allows using only a $1/(\ell + 1)$ fraction of the alphabet Q.

To better understand Construction 2.2, it may be beneficial to view it as a concatenated coding scheme. The code $\mathcal{C}'$ is a concatenation of the outer code Σ' and an inner code for each symbol (the mapping $Q' \leftrightarrow Q$) that partially corrects an asymmetric ℓ-limited-magnitude error, to have the outer code Σ' observe at most one symmetric error. Figure 2.9 illustrates this view of the systematic-code construction.

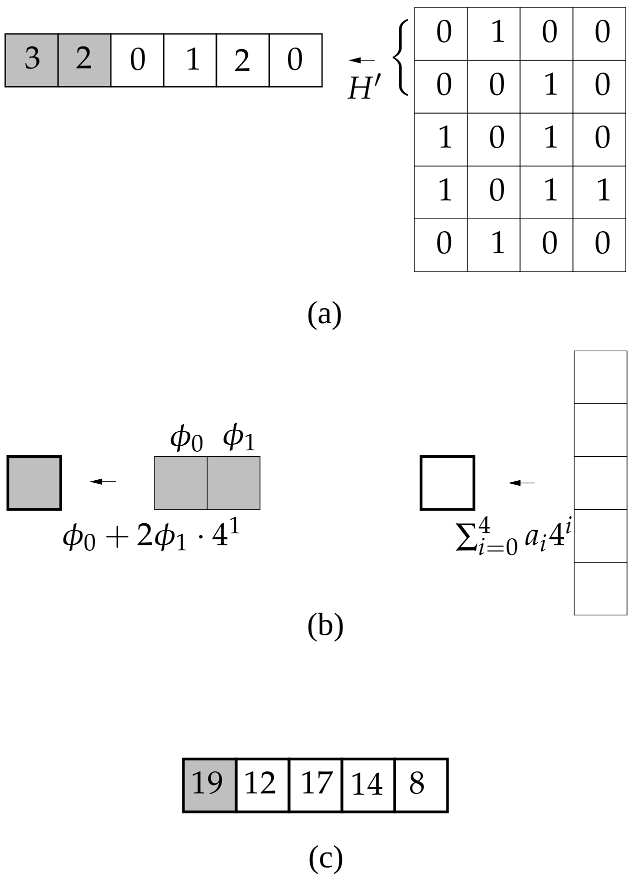


Figure 2.8: Encoding of a systematic sample code with $t = 1$ and $\ell = 3$. (a) Encoding the top two rows using the modified quaternary Hamming code (parity symbols in shaded boxes). (b) Mapping information bits using the standard positional mapping and mapping parity quaternary symbols using the mapping in (2.6). (c) The resulting codeword over the alphabet of size 32.

2.6 Codes for Asymmetric and Symmetric Limited-Magnitude Errors

In Flash memory applications, a dominant error source may cause most of the errors to be in one known direction. However, other, more secondary error sources can inject errors that are more symmetrical in nature, but still have low magnitudes. To answer such plausible scenarios, we address a variation of the asymmetric ℓ-limited-magnitude error model to include a (small) number of *symmetric* ℓ-limited-magnitude errors.

Definition 2.4 *A* $(t_\uparrow, t_\updownarrow)$ ***asymmetric/symmetric ℓ-limited magnitude error*** *is a vector e such that* $|\{i : e_i \neq 0\}| \leq t_\uparrow + t_\updownarrow$. *In addition,* $t_\updownarrow$ *of the indices of e satisfy* $-\ell \leq e_i \leq \ell$,

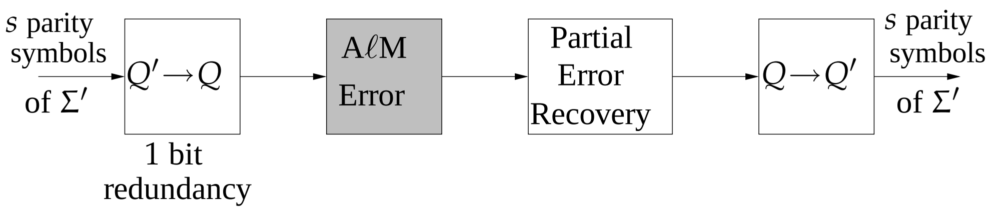


Figure 2.9: Concatenated-Coding view of Construction 2.2. The mapping $Q' \to Q$ can be viewed as an Inner code in a concatenated coding scheme in which Σ' is the Outer code.

and the remaining $n - t_{\updownarrow}$ indices satisfy $0 \le e_i \le \ell$.

In the following, we present a construction method for codes $\mathcal{C}_{\uparrow,\updownarrow}$ that correct $(t_\uparrow, t_\updownarrow)$ asymmetric/symmetric ℓ-limited-magnitude errors. This enhanced error correctability is achieved by modifying Construction 2.1 with the addition of an auxiliary binary code and a special mapping from information bits to q-ary symbols. We assume for simplicity that $q = 2^s(\ell + 1)$, for some integer s.

Construction 2.3 *Let $\boldsymbol{\sigma} = (\sigma_1, \ldots, \sigma_n)$ be a codeword of a code Σ, over an alphabet of size $\ell + 1$, that corrects $t = t_\uparrow + t_\updownarrow$ symmetric errors. Let $V = (\vec{v}_1, \ldots, \vec{v}_n)$ be a two-dimensional binary array of size $s \times n$, taken from an array code $\mathbb{C}$ that corrects a single bit error in each of at most $t_\updownarrow$ columns*[3]. *Each symbol of $\boldsymbol{x} \in \mathcal{C}_{\uparrow,\updownarrow}$ is composed from a symbol of the codeword $\boldsymbol{\sigma}$ and a bit vector of the codeword V as follows.*
For any i,

$$x_i = (\ell + 1) \cdot \mathrm{Gray}(\vec{v}_i) + \sigma_i$$

where $\mathrm{Gray}(\vec{u})$ is the sequential number of the vector $\vec{u}$ in a binary Gray code on s bits. The code $\mathcal{C}_{\uparrow,\updownarrow}$ contains all $|\Sigma| \cdot |\mathbb{C}|$ compositions of the codewords of Σ and $\mathbb{C}$.

Proposition 2.13 *The code $\mathcal{C}_{\uparrow,\updownarrow}$ is a $(t_\uparrow, t_\updownarrow)$ asymmetric/symmetric ℓ-limited-magnitude error-correcting code.*

Proof: Decoding of $\mathcal{C}_{\uparrow,\updownarrow}$ is performed in two steps. Firstly, $\mathcal{C}_{\uparrow,\updownarrow}$ is decoded as if it was a plain t asymmetric ℓ-limited-magnitude error-correcting code (of Construction 2.1). For the $t_\updownarrow$ coordinates that possibly suffered errors in the downward direction, the first decoding

[3]Such codes can be obtained by length sn, binary $t_\updownarrow$ error-correcting codes, or more cleverly, using J.K Wolf's Tensor-Product code construction method [Wol06]

step miscorrects these errors to *exactly* $\ell + 1$ levels below their correct levels. Thus, for each of these $t_{\updownarrow}$ miscorrections, the Gray mapping guarantees that the error resulting from this $\ell + 1$ level discrepancy will be observed by the code $\mathbb{C}$ as a single bit error. $\square$

Example 2.5 below illustrates the encoding and decoding of a code originating from Construction 2.3.

Example 2.5 *In this example we protect* 7 *symbols over an alphabet of size* $q = 12$ *against* $t_{\uparrow} = 2$ *asymmetric errors plus* $t_{\updownarrow} = 1$ *symmetric error. Both the asymmetric and symmetric errors have magnitude limit of* $\ell = 2$*. In Figure* 2.10*,* $\boldsymbol{\sigma}$ *is a codeword of the ternary repetition code that corrects* $t_{\uparrow} + t_{\updownarrow} = 3$ *symmetric errors. The bits of* V *, placed in two rows, are a codeword of the (shortened) binary Hamming code of length* 14*. Each column of* V *is mapped to an integer in* $\{0, 1, 2, 3\}$ *using the Gray code, and the final codeword* $\boldsymbol{x}$ *combines* V *and* $\boldsymbol{\sigma}$ *through the formula*

$$\boldsymbol{x} = 3 \cdot \mathrm{Gray}(V) + \boldsymbol{\sigma}$$

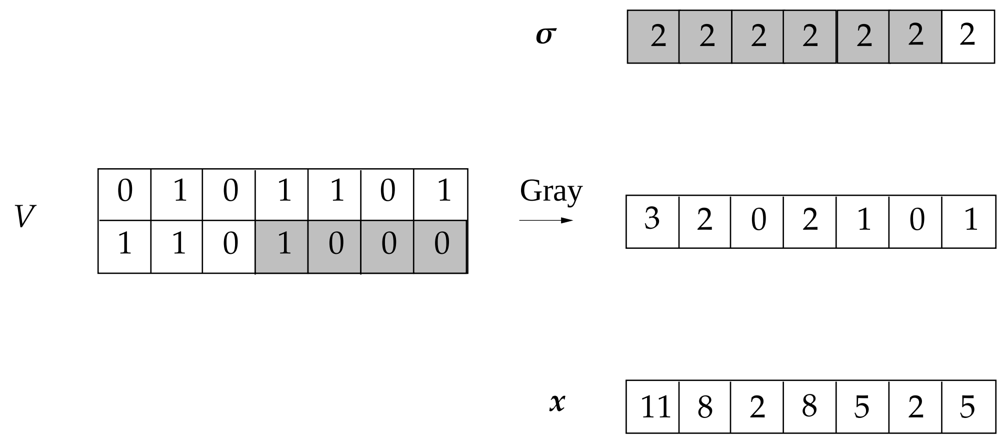


Figure 2.10: Example of a code for asymmetric and symmetric limited-magnitude errors. From top to bottom: a codeword $\boldsymbol{\sigma}$ of the ternary repetition code; a binary Hamming codeword arranged into a 2×7 array and its Gray mapping; the final codeword $\boldsymbol{x}$ obtained by combining $\boldsymbol{\sigma}$ and V.

Decoding of the sample code above is illustrated in Figure 2.11*. The codeword in (a) is corrupted by* 2 *asymmetric (upward) errors and* 1 *downward error; the resulting word is given in (b). In (c) the result of correcting* 3 *<u>asymmetric</u> limited-magnitude errors is given.*

The "corrected" array $\tilde{V}$ is shown in (d), and the top bit of the third column from right (marked with a bold-face **0***) is found to be in error. Finally, in (e) the third symbol from right (in bold face) is adjusted* 3 *levels upward after a miscorrection was detected at the previous step.*

(a) codeword

11	8	2	8	5	2	5

(b) corrupted

11	10	2	8	4	3	5

(c) AℓM decoded

11	8	2	8	2	2	5

(d) corrected $\tilde{V}$

0	1	0	1	**0**	0	1
1	1	0	1	0	0	0

(e) SℓM adjusted

				$+3$		
11	8	2	8	**5**	2	5

Figure 2.11: Example of decoding asymmetric and symmetric limited-magnitude errors. (a) Codeword. (b) Codeword corrupted by asymmetric and symmetric limited-magnitude errors. (c) First decoding step: correction of asymmetric limited-magnitude (AℓM) errors. (d) Resulting corrected codeword $\tilde{V}$ is decoded using a Hamming decoder. (e) Adjusting the miscorrection of the symmetric error found in the previous step.

Note that the amount of redundancy (of both σ and V) required in the example to correct $(2, 1)$ asymmetric/symmetric errors is smaller than if V is not restricted and the repetition code is taken over an alphabet of size $2\ell + 1 = 5$ (that scheme would correct 3 *symmetric $\ell = 2$ limited magnitude errors).*

The counter-intuitive part of Construction 2.3 is that *binary* Gray mappings are used regardless of the error-magnitude ℓ. This fact implies that the codes Σ and $\mathbb{C}$ cooperate with each other to achieve the prescribed correction capability, otherwise $\mathbb{C}$ would need to operate over a larger alphabet for $\ell > 1$.

2.7 Asymmetric Limited-Magnitude Error-Correcting Codes in Flash Devices

While the majority of the results of this chapter are formulated in mathematical terms, their great practical promise should not be overlooked. The gap between a good coding scheme from a theoretical perspective and an attractive coding scheme in practice is deep and often daunting – it was proved historically that improved error resilience, lower redundancy, and even efficient decoding do not suffice to attract technology providers to implement a coding scheme. In this section our intention is to project the coding results above, on the design and operation of real Flash devices, thus showing their value for that particular application. To do that we first show how asymmetric limited-magnitude error-correcting codes can be deployed with minimal excess hardware over current Flash architectures. Next we analyze, as a function of ℓ, the savings in programming time offered by asymmetric ℓ-limited-magnitude error-correcting codes.

2.7.1 Implementation architecture

The codes proposed in this chapter enjoy a key property that seems to allow a relatively painless access to them by commercial Flash products. The fact that the error-correcting engine of the new code constructions are codes for the symmetric channel, which are used anyway by common Flash devices to correct memory errors, permits a simple modification of the Flash architecture to obtain a much more targeted treatment of the observed errors. In Figure 2.12, a simplified architecture of a typical Flash device is presented. The Flash cell contents are measured and converted to discrete levels using the A/D (Analog to Digital converter) block. Then, to match the chosen error-correcting code for symmetric errors, the discrete levels are represented in the appropriate alphabet (using the Alphabet Converter) and fed to the ECC (Error-Correcting Code) decoder. The outputs of the decoder are then delivered to the device user. By converting the cell programmed levels to a lower alphabet the structure of the Flash errors is lost and cannot be utilized by the ECC decoder. In comparison, for the coding scheme proposed in this chapter, a similar architecture provides guaranteed error control against common asymmetric limited-magnitude errors. In Figure 2.13, the cell levels are similarly measured and converted to discrete levels. The modulo $\ell + 1$ of these levels are fed to the same ECC decoder as in Figure 2.12, whose *error estimates* are now subtracted from the discrete measured levels over the *full alphabet* (the subtraction is represented by the $\bigoplus$ adder blocks). The corrected symbols are then passed to the user after a possible alphabet conversion.

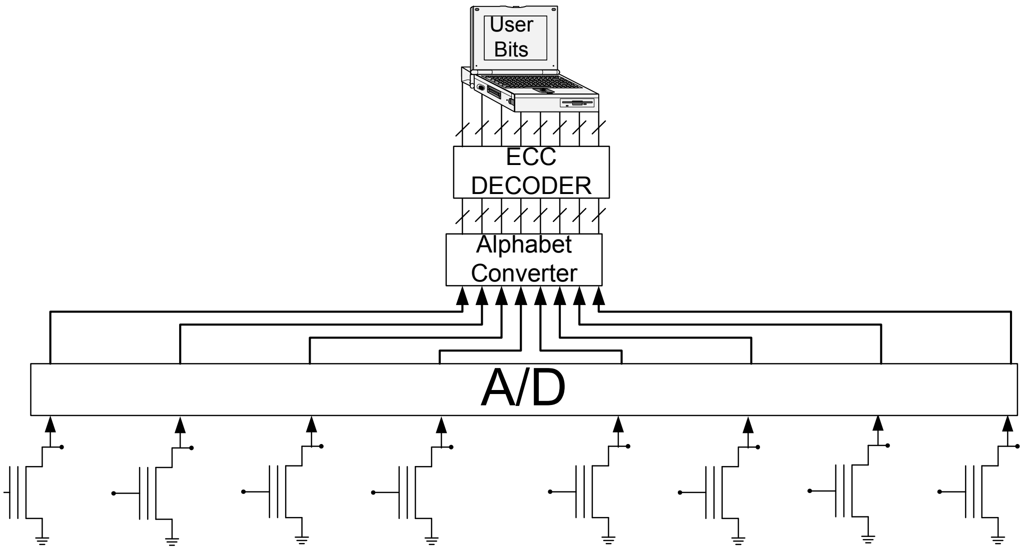


Figure 2.12: Flash architecture with symmetric error-correcting codes. The correction of errors is performed on the low-alphabet symbols, thus not utilizing the specific characteristics of Multi-level Flash errors.

By installing circuitry to support the modulo operation and simple additions, the device designer is free to choose variable ECC Decoder blocks to obtain any error correction capability specified by t and ℓ.

2.7.2 Programming speed-up

As mentioned in the introduction of this chapter, asymmetric limited-magnitude error-correcting codes can be used to speed up the cell programming process by allowing faster, less precise programming pulse sequences. The behavior of a typical optimized Flash programming sequence is shown in the graphs of Figure 2.14, which is taken from [BSH05]. The integers of the horizontal axis represent the program-pulse sequential numbers and the

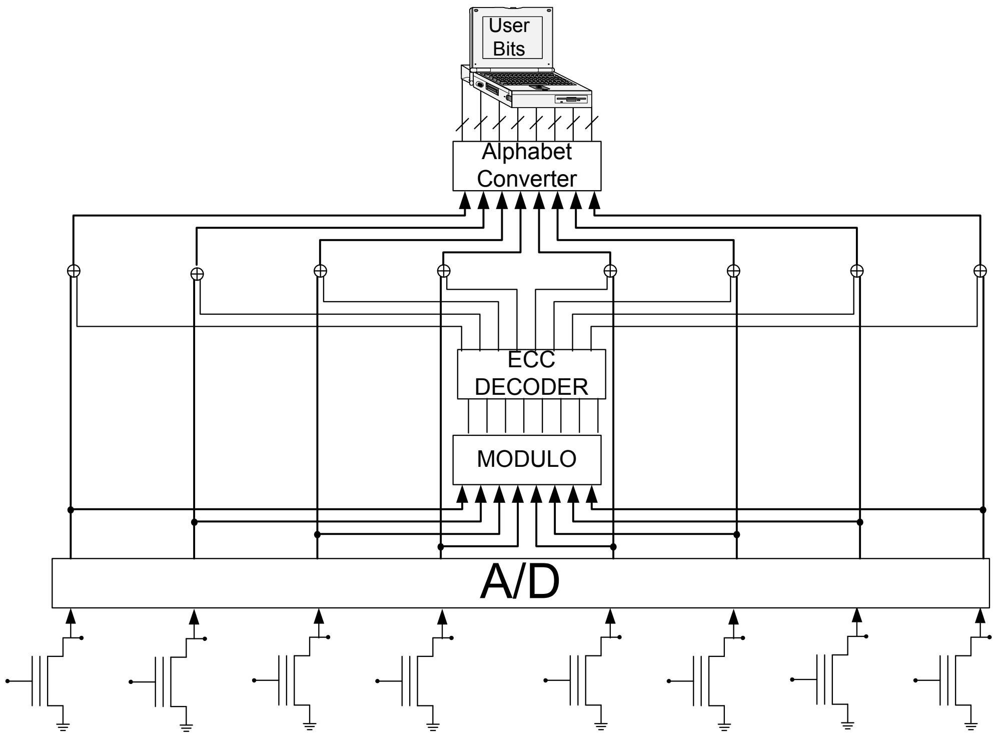


Figure 2.13: Proposed Flash architecture with asymmetric limited-magnitude error-correcting codes. Here the error estimates from the decoder are subtracted from the symbols over the original alphabet thus utilizing the error structure for targeted error correction.

vertical axis represents electric-current levels to which Flash cells are programmed. A circle on the a graph represents a current level achieved by a pulse at some point along the programming sequence. The different graphs in Figure 2.14 represent program sequences with different target current values. As can be clearly seen, most of the progress toward the target value is achieved by the early pulses, and the numerous later pulses are used for a fine asymptotic convergence to a value very close to the target. Therefore, having even a small error resiliency against asymmetric limited-magnitude errors can allow the programming sequence to terminate long before hitting the target value (due to the asymptotic nature of the programming curves) thus significantly speeding up memory access. Increasing the error resiliency beyond the flat part of the curve does not add significant benefits as at the steeper part of the curve the vertical concentration of programming points becomes sparser.

To supplement the experimental evidence above, that tolerance to asymmetric limited-magnitude errors can speed-up the programming sequence, a quantitative analysis of the

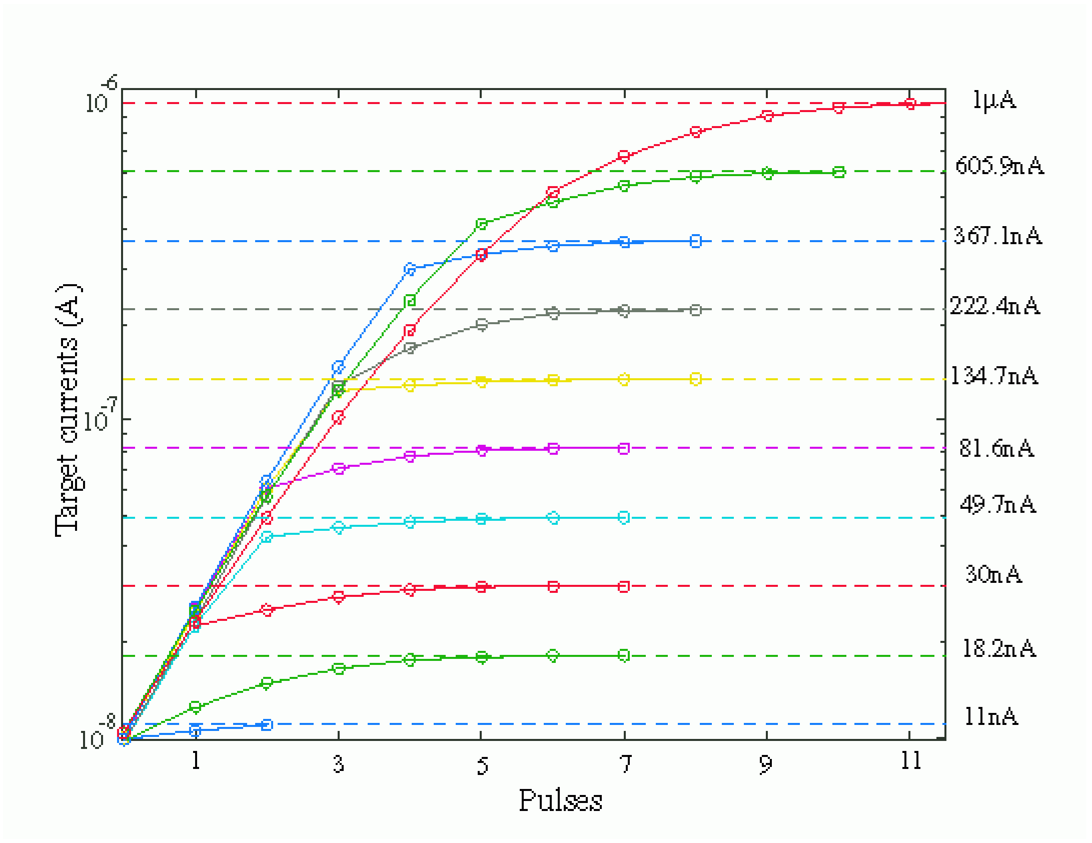


Figure 2.14: Performance of a Flash adaptive program sequence [BSH05]. The circles on each curve describe the results of an iterative programming algorithm for a given target value.

time savings is now carried out. The inputs to a Flash programming algorithm are the *initial* and *target* current levels; its output is a programming pulse of some width and amplitude, that attempts to move closer to the target level, under some constraints. To have an analytic description of the programming sequence, we need to model the programming algorithm in a way that captures its main design constraints in practice. In Flash devices, preventing *over-programming*, whereby the programming result exceeds the target level, is a crucial consideration taken by the programming algorithm. The reason for that being that Flash devices do not support single-cell erasures, and an over-programming instance requires erasing a full Flash *block*, an operation that is very costly in terms of time and device wear. The analysis that follows, strongly builds on that property of Flash devices.

Suppose a Flash cell is to be programmed from a lower level I_i to a higher target level I_F. Since the change δ in the current level is a random variable whose distribution depends on the chosen programming pulse, we model it as an *exponentially distributed* random variable with mean $1/\mu$. μ will be determined by the programming algorithm as a function of I_i, I_F, and subject to a constraint of fixing a low probability of over-programming. Specifically, μ will be taken such that

$$\Pr(I_i + \delta > I_F) = \epsilon$$

ϵ is a global parameter that specifies the allowable probability of over-programming. Substituting the exponential distribution of δ, we get the integral equation

$$\int_{I_F - I_i}^{\infty} \mu \exp(-\mu\delta) d\delta = \epsilon \tag{2.7}$$

(See Figure 2.15 for illustration.)

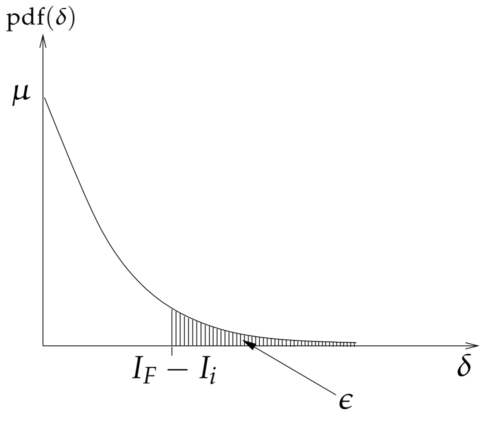


Figure 2.15: Choice of a programming distribution based on the specified probability of over-programming. For starting level I_i and target level I_F the parameter μ of the exponential distribution is chosen such that the marked area under the probability density function graph equals ϵ (the specified probability of over-programming)

Solving (2.7) and rearranging we get

$$\mu = -\frac{\ln(\epsilon)}{I_F - I_i}$$

Hence we have the following relationship between the lower level I_i and the final (higher) level I_{i+1}:

$$I_{i+1} = I_i + \delta_i\,, \quad \delta_i \sim \text{Exponential}[-\ln(\epsilon)/(I_F - I_i)] \tag{2.8}$$

Note that the parameter of the exponential distribution of δ_i at each step i depends on the starting level I_i that is itself a random variable.

Starting from an initial level I_0, the programming algorithm recursively updates the cell level according to (2.8), and stops after the n^{th} step if $I_n \geq I_F - \Delta$, where Δ is the maximum allowed deviation from the target level I_F. Discussed in detail later, the parameter Δ specifies the device tolerance to programming errors in the downward direction. A pictorial illustration of the modeled programming sequence is given in Figure 2.16.

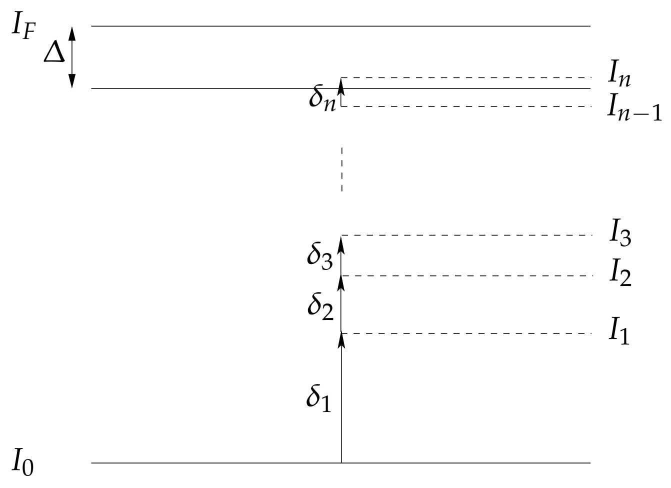


Figure 2.16: A pictorial illustration of the modeled programming sequence. On the left side are the initial level I_0, the target level I_F and the tolerance parameter Δ. In the middle is a sequence of exponentially distributed level increments $\delta_1, \delta_2, \ldots, \delta_n$ resulting from the programming algorithm. On the right side are the instantaneous levels I_i until the process terminates at I_n.

To analyze the performance of the programming algorithm, we need to find the expected number of steps n, such that

$$I_{n-1} < I_F - \Delta \leq I_n$$

However, given the complex[4] structure of the random process I_i, finding the mean of n is hard. Instead, we will approximate I_i's mean crossing time by the (deterministic) crossing time of the mean of I_i. This latter calculation is significantly easier since we can use the linearity of expectation to obtain a recursive formula for the mean of I_i. The accuracy of that approximation can be established using concentration bounds (e.g. Chebyshev inequality), however for the discussion here a first order approximation should suffice.

[4] I_i is a Markov process with an uncountable number of states

Now taking the mean of equation (2.8) we write

$$\overline{I_{i+1}} = \overline{I_i} + E\left[\frac{1}{\mu_i}\right] = \overline{I_i} + K_\epsilon(I_F - \overline{I_i}) \tag{2.9}$$

where $K_\epsilon \triangleq -1/\ln(\epsilon)$. Rewriting (2.9) provides a recurrence relation on the expected programmed levels

$$\overline{I_{i+1}} = \overline{I_i}(1 - K_\epsilon) + K_\epsilon I_F$$

Solving the recurrence for initial level I_0 we get the expression

$$\overline{I_n} = I_0(1 - K_\epsilon)^n + I_F K_\epsilon \sum_{i=1}^{n} (1 - K_\epsilon)^{i-1}$$

which after simplification becomes

$$\overline{I_n} = I_F - (1 - K_\epsilon)^n (I_F - I_0) \tag{2.10}$$

Now, by equating (2.10) to $I_F - \Delta$ we can calculate the time N when the sequence of means $\overline{I_n}$ crosses $I_F - \Delta$:

$$I_F - (1 - K_\epsilon)^N (I_F - I_0) = I_F - \Delta$$

that gives

$$N = \frac{\log(I_F - I_0) - \log(\Delta)}{-\log(1 - K_\epsilon)} \tag{2.11}$$

The importance of (2.11) is that it describes how the number of required pulses N depends on the error margin Δ. To compare the programming speed of Flash devices with and without an asymmetric limited-magnitude error-correcting code, we define two different error margins, Δ_c and Δ_{uc}, respectively (the subscript c stands for *coded* and the subscript uc stands for *uncoded*, and obviously $\Delta_c > \Delta_{uc}$). The difference between the corresponding numbers of pulses N_{uc} and N_c is then

$$N_{uc} - N_c = \frac{\log(\Delta_c/\Delta_{uc})}{-\log(1 - K_\epsilon)}$$

A conservative assumption is to set $\Delta_c = (\ell + 1)\Delta_{uc}$, where ℓ is the parameter of the asymmetric ℓ-limited-magnitude error-correcting code. This assumption corresponds to allowing the uncoded device a tolerance of one level (over the discrete alphabet Q), and the coded device a tolerance of ℓ additional levels for the total of $\ell + 1$ levels. Under that assumption, the saving in the number of programming pulses equals

$$N_{uc} - N_c = \frac{\log(\ell + 1)}{-\log(1 - K_\epsilon)} \tag{2.12}$$

For an over-programming probability $\epsilon = 0.01$ the above equals

$$N_{uc} - N_c = 4.08 \log(\ell + 1)$$

Values of savings for different values of ℓ are given in Table 2.1.

ℓ	$N_{uc} - N_c$
1	2.84
2	4.48
3	5.66
4	6.57
5	7.31
6	7.94

Table 2.1: Approximate average savings in programming pulses for sample values of ℓ

Another quantity of interest is the percentage of savings $(N_{uc} - N_c)/N_{uc} \times 100$, which depends on the particular difference $I_F - I_0$. For a programming window of $I_F - I_0 = a\Delta$, a is an integer specifying the target increase in discrete levels, the part of the programming duration saved by the code equals

$$\frac{\log(\ell + 1)}{\log a},$$

as long as $a < q - \ell$. The median[5] saving percentage is obtained by taking $a = q/2$ and is equal to

$$\frac{\log(\ell + 1)}{\log(q/2)}.$$

[5]The median savings is a simple approximation to the average savings, which has an unwieldy expression. For small ℓ (compared to q) it is a relatively good approximation.

For a sample number of levels $q = 32$, the median savings in programming time suggested by the model is plotted in Figure 2.17.

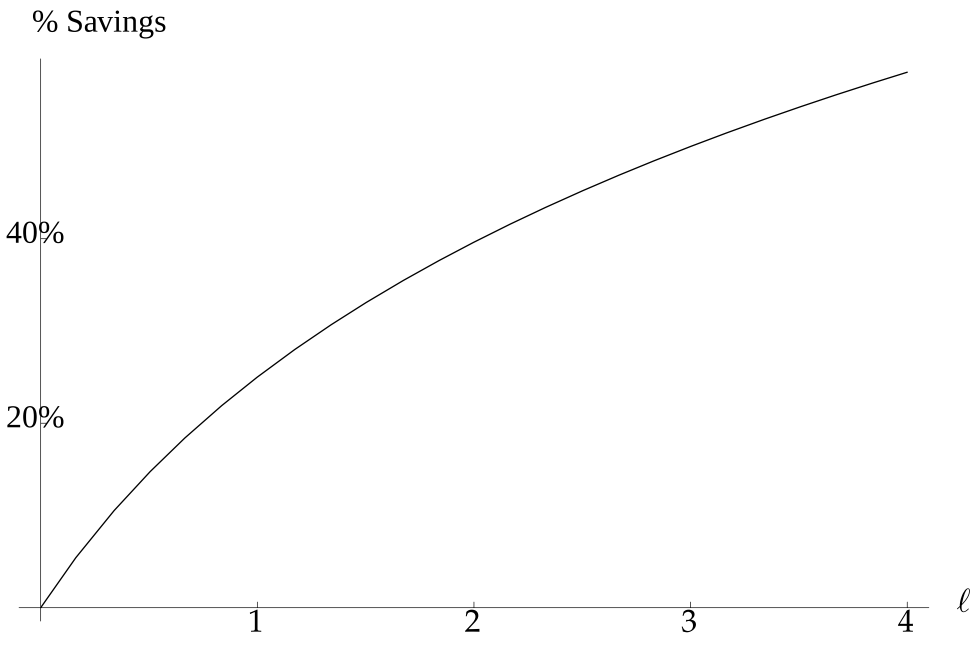


Figure 2.17: Percentage of program-time savings as a function of the code's magnitude limit parameter ℓ. Significant savings are suggested even for small ℓ and returns are diminishing for growing ℓ.

As seen in both Figure 2.17 and Table 2.1, while even small ℓ values suggest significant savings, increasing ℓ beyond some point exhibits diminishing returns and does not significantly contribute to increased savings in programming time. Note that this last qualitative observation is one we have already made when discussing Figure 2.14 earlier in the sub-section. Thus both analytical and experimental evidence motivate the application of asymmetric limited-magnitude error-correcting codes (with small ℓ), as clearly codes for symmetric errors will not be an efficient solution for programming speed-up.

Note that while our model successfully predicts the asymptotic behavior of the programming algorithm (through the $(1 - K_\epsilon)^n$ sequence in (2.10)), it stops short of accounting for some of the properties of the curves in Figure 2.14. For example, the expression for $N_{uc} - Nc$ in (2.12) suggests that the numbers of saved pulses are independent of the initial and target levels. Whereas comparing the uppermost and middle curves of Figure 2.14 clearly concludes that this is not the case in practice, and implies that there may be other constraints on the programming algorithm not included in our model. The design of real programming algorithms in the presence of asymmetric limited-magnitude error-correcting codes is thus an interesting and promising research avenue.

2.8 Conclusions and Future Research

This chapter proposes a new coding technique that is motivated by Multi-Level Flash Memories. Defining a natural new error model has opened the way to a simple but powerful construction method that enjoys good storage and implementation efficiencies. By an interplay between symbol mappings and constraints on the full code block, several useful extensions to the basic code construction are achieved. An attractive property of the codes herein is that the coding parameters n, t, ℓ need not be fixed for a Flash Memory device family. After implementing the simple circuitry to support this coding method in general (modulo and other arithmetic operations), different code parameters can be chosen, by using varying external coding modules for the symmetric error-correcting code. Many of the strengths of this construction method were not explored in this chapter. When the reading resolution is larger than the code alphabet size (e.g., readers that give a real number rather than an integer), improved decoding techniques can be readily applied using *limited-magnitude erasures* or other soft interpretations of the read symbols. Better systematic codes may be obtained by observing the relationship between the limited-magnitude errors and the errors they induce on the low-alphabet code, and then replacing the symmetric error-correction properties required here (which are too strong) with various Unequal Error Protection properties.

Proving the asymptotic optimality of Construction 2.1 for all values of ℓ and t lies upon the existence of a proof to the following conjecture.

Conjecture 2.1 *For any a and t, $A_a(n,t)$ (size of largest a-ary code for symmetric errors) and $\mathrm{A}sym_a(n,t)$ (size of largest a-ary code for asymmetric errors) satisfy the following equality.*

$$\lim_{n\to\infty} \frac{1}{n} \log_a |A_a(n,t)| = \lim_{n\to\infty} \frac{1}{n} \log_a |\mathrm{A}sym_a(n,t)|$$

This was proved for $a = 2$ and for restricted t if $a > 2$.

More on the practical side, there is a need to devise optimization algorithms that “budget” controlled errors to achieve maximal savings in programming time. An experimental study on commercial standard Flash devices may also be helpful to quantify the improvement in density that these codes can achieve.

Chapter 3

Codes for Random and Clustered Device Failures

Similarly to the previous chapter, this chapter shows how refined design goals, in conjunction with clever construction methods, can provide efficient coding schemes that match more realistic characteristics and constraints of practical data-storage systems. If in the previous chapter it was found useful to renounce the *symmetry* assumption on the channel, this time around it is the *memorylessness* assumption on the channel that is shown to be limiting and inessential. When array codes are used as abstractions of failure-protected storage arrays, each array column represents a physical device or storage unit. The traditional MDS (Maximum Distance Separability) requirement on the codes assumes device failures that are uniformly distributed across the storage array, without taking into account the effects of the physical layout of the devices within the array. The main motivation to depart from the MDS model in this chapter, is the premise that device failures tend to cluster, and therefore, a failure pattern of the form of Figure 3.1(a) is more likely than the isolated failures of Figure 3.1(b). The main contributions of this chapter can be summarized as follows.

- A new classification of error combinations based on the number of clusters that they occupy.
- Construction of a family of codes with excellent Clustered-erasure correction and good Random-erasure correction capabilities. The new codes have superior encoding, decoding and update complexities compared to the best known codes for Ran-

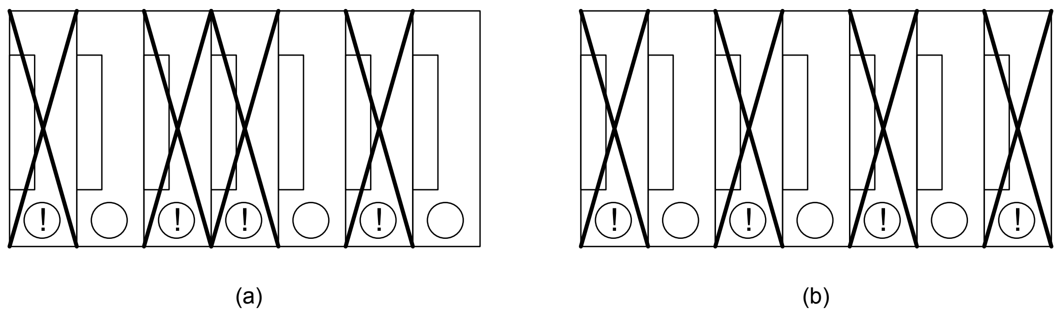


Figure 3.1: Clustered and Non-clustered patterns of 4 device failures. (a) Clustered pattern includes two adjacent failures. (b) Non-clustered pattern has isolated failures only.

dom erasures.

- Statistical analysis of the reliability of redundant disk arrays under Random and Clustered failures.

3.1 Introduction

Protecting disk arrays and other dynamic-storage systems against device failures has long become an essential part of their design. Implemented solutions to data survivability in the presence of failed hardware have progressed considerably in the last two decades. In the dawn of failure-protected storage systems, relatively simple schemes were implemented. In RAID [PGK88] arrays, a redundant disk is used to store parity bits of the information disks, which allows recovering from any single disk failure. Simple data replication and data striping are also commonly used to avoid data loss. Meanwhile, storage requirements are growing rapidly and at the same time, device reliability was reduced to control the implementation cost. Consequently, recovering from only a single failure has become inadequate while data replication is turning infeasible. Schemes that are based on the Reed-Solomon codes [MS77] can recover from more failures, and with a good resiliency-redundancy trade-off, but they require complex decoding algorithms (in either space or time) and they also have high update complexity – many parity writes are needed for small data updates. These shortcomings left such schemes out of reach of many storage applications.

The class of codes called *array codes* [BFvT98] addresses both issues of simple decoding and efficient updates, while maintaining good storage efficiency. The idea behind array codes is that the code is defined on two-dimensional arrays of bits, and encoding and decoding operations are performed over the binary alphabet, using simple Exclusive OR operations. An example of an array code with two parity columns that can recover from any two column erasures is given below. The $+$ signs represent binary Exclusive OR operations. The three left columns contain pure information and the two right columns contain parity bits that are computed from the information bits as specified in the chart below.

a	b	c	$a+b+c$	$a+f+e+c$
d	e	f	$d+e+f$	$d+b+e+c$

Like encoding, decoding is also performed using simple Exclusive OR operations. For example, recovering the bits a, b, d, e at the two leftmost columns is done by the following chain of computations.

$$\begin{aligned} e &= c+(a+b+c)+(d+e+f)+(a+f+e+c)+(d+b+e+c) \\ d &= e+f+(d+e+f) \\ a &= c+f+e+(a+f+e+c) \\ b &= c+a+(a+b+c) \end{aligned}$$

It is left as an exercise to verify that any two column erasures can be recovered by the code above. The small-write update complexity (the qualifier *small-write* is often omitted) of an array code is the number of parity-bit updates required for a single information-bit update, averaged over all the array information bits. In the sample code above, each of the bits a, b, d, f requires 2 parity-bit updates, and each of e, c requires 3 parity-bit updates. The update complexity of that sample code is hence $(4 \cdot 2 + 2 \cdot 3)/6 = 2.333$.

In the literature of array codes a column serves as an abstraction to a disk or other physical storage unit, and column erasures represent disk failures. The sample array code considered above, belongs to an infinite family of array codes called EVENODD codes [BBBM95], that protect p information columns against two column erasures, for any prime p (in the example we took $p = 3$). The EVENODD family of codes and its relatives (e.g. [CEG+04]), can recover from any two erasures with optimal redundancy (MDS), and enjoy simple decoding and fairly low update complexity. EVENODD codes for more than two erasures exist [BBV96], but their decoding becomes more complex for growing numbers of erasures, and their update complexity grows as fast as $2r - 1$, for r correctable erasures. A high update complexity limits the system performance as it imposes excess disk I/O operations, even if no failures occur. High update complexity also implies high wear of parity disks whose direct consequence is the shortening of disk lifetimes.

The primary incentive to move to higher order failure resilience in disk arrays is to combat "catastrophic" events, in which multiple disks fail simultaneously. For such events, the assumption that device failures occur independently of each other is no longer true, and many failure mechanisms cause multiple disk failures that are highly correlated. Since adjacent disks in the array share cabling, cooling, power and mechanical stress, failure combinations that are clustered are more likely than completely isolated multiple failures. Consequently, array codes that have excellent Clustered-failure correctability, good Random-failure correctability, and low update complexity are desirable for high order failure protection of disk arrays.

Motivated by correlated disk failures in high-order failure events, a new classification of erasure combinations is proposed. Each combination of column erasures will be classified by the number of erased columns, and by the number of *clusters* in which the erased columns fall. The number of clusters captures the number of "independent" failure events, each possibly affecting multiple disks in a single cluster. This model is related to the model of *multiple burst* erasure correction, however the new (and stronger) model seems to better capture the correlated failure patterns in disk arrays, since it does not predefine the *size* of the clusters, only their number.

This chapter pursues the first attempt to improve the performance of array codes by prioritizing the correctable failures based on their relative locations in the array. By doing that, we part from the abstraction of a fault tolerant storage-array as an MDS code, in the hope to achieve a more realistic trade-off between redundancy and performance. A more general "black box" framework to trade-off storage space and access efficiency, discussed in [HCL07], does not offer benefits comparable to the results herein. The main contribution of this chapter is the construction of an array-code family, called RC codes (for Random/Clustered), that corrects essentially all Clustered, and 7 out of 8 non-Clustered, 4-failure combinations. The RC codes are better than EVENODD($r = 4$) in all implementation complexity parameters. They improve the encoding and decoding complexities by 25% and the small-write update complexity by 28.57%. They also support twice as many disks compared to EVENODD codes, for the same column size. To compare RC-coded disk arrays to EVENODD-coded ones in terms of their reliability performance, analysis of the Mean Time To Data Loss ($MTTDL$), under Random and Clustered failures is carried out.

3.2 Definitions and Notations

3.2.1 Array Codes

The definitions in this sub-section are standard coding-theory terminology that provides a good abstraction for failure-resilient disk arrays. A *length* n array code consists of n columns. A column is a model for, depending on the exact application, either a whole disk or a different physical data unit (such as a sector) in the disk array. In the codes discussed here, there are k columns that store uncoded information bits and r columns that store redundant parity bits (thus $n = k + r$). This array structure has the advantage that information can be read off a disk directly without decoding, unless it suffered a failure, in which case a decoding process is invoked. An array code that admits this structure is called *strongly systematic*. A *column erasure* occurs when, for some physical reason, the contents of a particular column cannot be used by the decoder. An erasure is a model for a device failure whereby all the data on a disk (or other physical unit) is known to have become unusable. We say that an array with given column erasures is *correctable* by the array code if there exists a decoding algorithm that, independent on the specific array contents, can reconstruct the original array from unerased columns only. An array code is called MDS (*Maximum Distance Separable*) if it has r redundant columns and it can correct all possible combinations of r column erasures. MDS codes obviously have the strongest conceivable erasure correction capability for a given redundancy, since the k information columns can be recovered from *any* k columns. Beyond space efficiency of the code, one should also consider its I/O efficiency. I/O efficiency of a disk array is determined by the *small-write* and *full-column* update complexities of the array code used. The small-write update complexity (often simply called update complexity) is defined as the number of parity-bit updates required for a single information bit update, averaged over all information bits. The full-column update complexity is the number of parity columns that have to be modified per a single full-column update. Another crucial performance marker of an array code is its *erasure-decoding complexity*, defined as the number of bit operations (additions, shifts) required to recover the erased columns from the surviving ones. Unless noted otherwise, p will refer to a general prime number p.

3.2.2 Random/Clustered erasure correction

To describe column erasure combinations whose only restriction is the number of erased columns, it is customary to use the somewhat misleading term *Random* [LC83] erasures.

Definition 3.1 *An array is said to recover from ρ* ***Random erasures*** *if it can correct all combinations of ρ erased columns.*

The Random erasure model is most natural when storage nodes are known to, or more commonly, assumed to behave uniformly and independent of each other. Indeed, almost all array-code constructions discussed in the literature aim at correcting Random erasures. Refinement of the erasure model is possible by adding restrictions on the relative locations of the erased columns. This chapter considers *Clustered* erasures where the ρ erasures fall into a limited ($< \rho$) number of clusters. We now turn to some definitions related to the Clustered erasure model. In words, a *cluster* is a contiguous block of columns. More precisely,

Definition 3.2 *In an array code with columns numbered $\{0, 1, 2, \dots, n-1\}$, a* ***cluster*** *is a set of σ columns such that the difference between the highest numbered column and the* *lowest numbered one is exactly $\sigma - 1$.*

For example, $\{2, 3, 4, 5\}$ is a cluster with $\sigma = 4$. Now given a set of columns, the number of clusters that it occupies is the partition of that set to a minimal number of subsets, each of which is a cluster according to the definition above. Now we include another definition that will be useful later.

Definition 3.3 *A set of ρ columns is called* ***Clustered*** *if the number of clusters it occupies is* strictly *less than ρ.*

Random erasures have no restriction on their respective numbers of clusters and therefore they include both Clustered and non-Clustered erasures. The other extreme is the *column burst* model where all erased columns need to fall into a single cluster. These two well-studied extreme cases open our presentation and later the RC codes are shown to be very effective for all intermediate cases of Clustered erasures. An illustration of the column-clustering definitions is given in Figure 3.2.

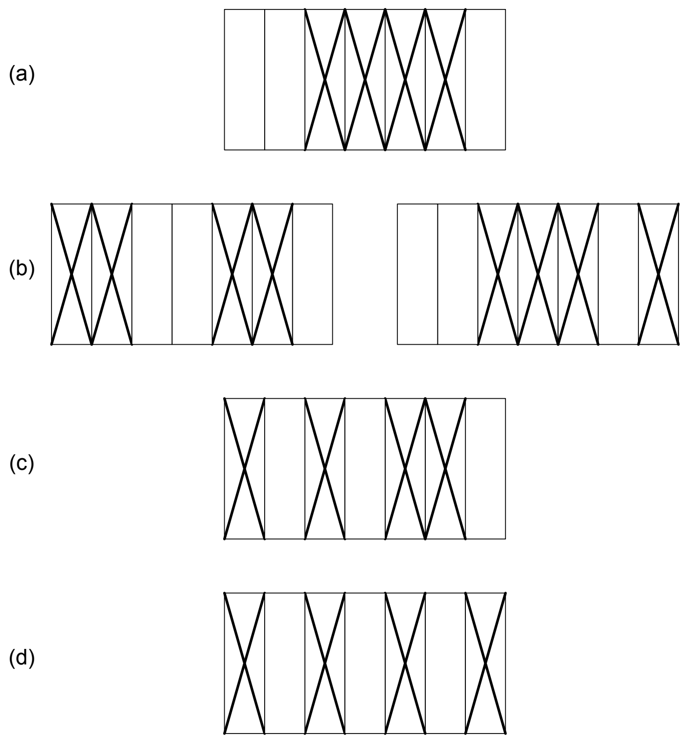


Figure 3.2: Classification of column combinations by their respective numbers of clusters. Four columns (marked with X) that fall into (a) One cluster (b) Two clusters (c) Three clusters (d) Four clusters (non-Clustered)

3.3 Preliminaries and Relevant Known Results

The purpose of this section is to discuss relevant known results in sufficient detail to prepare for the presentation of the new RC codes in the next section. Some algebraic tools used later for RC codes have been used before for other codes, so this section also serves to elucidate those tools prior to their use by the actual code construction.

3.3.1 Codes for erasures in a single cluster

Assume that our design goal is a disk array that will sustain any erasure of ρ columns in a single cluster, without requiring any Random erasure correction capability. One option is to take a code that corrects any ρ *Random* column erasures that, in particular, corrects any Clustered ρ erasures. However, as can be expected, this may not be the best approach since correcting all Random erasures is a far stronger requirement that excludes much simpler and more efficient constructs. As this section shows, a simple and well known technique called *interleaving* can achieve that task optimally both with respect to the required redundancy and in terms of the code update complexity.

Let $\mathcal{CP}$ be an array code with n' columns, out of which $k' = n' - 1$ are information columns. The remaining column holds the bit-wise parity of the k' information columns. Define the code $\mathcal{CP}_\rho$ as the length $n = \rho n'$ code that is obtained by the interleaving of ρ codewords of $\mathcal{CP}$. In other words, if $\mathsf{C}^{(1)}, \mathsf{C}^{(2)}, \ldots, \mathsf{C}^{(\rho)}$ are ρ codewords of $\mathcal{CP}$, then the corresponding code word of $\mathcal{CP}_\rho$ will be

$\mathsf{C}_1^{(1)}$	$\mathsf{C}_1^{(2)}$	$\cdots$	$\mathsf{C}_1^{(\rho)}$	$\mathsf{C}_2^{(1)}$	$\mathsf{C}_2^{(2)}$	$\cdots$	$\mathsf{C}_2^{(\rho)}$	$\mathsf{C}_3^{(1)}$	$\cdots$

Proposition 3.1 *The code $\mathcal{CP}_\rho$ corrects any ρ erasures in a single cluster.*

Proof: Any erasure that is confined to at most ρ consecutive columns erases at most one column of each constituent $\mathcal{CP}$ code. These single erasures are correctable by the individual $\mathcal{CP}$ codes. $\square$

It is clear that the code $\mathcal{CP}_\rho$ has optimal redundancy since it satisfies $\rho = r$ and ρ is a well known and obvious lower bound on the redundancy r. For any ρ, the code $\mathcal{CP}_\rho$ has update complexity (both small-write and full-column) of 1, which is optimal since a lower update complexity would imply at least one code bit that is independent of all other bits, and erasure of that bit would not be correctable.

3.3.2 Codes for Random erasures

As mentioned in sub-section 3.3.1, array codes that correct any ρ Random erasures also correct any ρ Clustered erasures. In this section we seek to survey a family of Random erasure correcting codes: the EVENODD [BBBM95],[BBV96] codes. These codes enjoy several properties that make them most appealing for implementation in disk arrays. The purpose of presenting the codes here is twofold. First, in the absence of prior codes that combine Clustered and Random correction capabilities, EVENODD will be used as the current state-of-the-art for comparison with our construction. Second, the analysis of the new RC codes herein is made simpler by building on properties previously shown for EVENODD.

3.3.2.1 EVENODD for correcting 2 Random disk erasures

An EVENODD code [BBBM95] takes p data columns, each of size $p-1$ and adds to them 2 parity columns of the same size. The encoded array is therefore of size $(p-1)\times(p+2)$. EVENODD can correct any 2 column erasures so it is clearly optimal in terms of added redundancy (MDS). Other properties of EVENODD are that it is strongly systematic, it has low small-write update-complexity that is approximately one above optimal, and optimal full-column update complexity. In addition, it can be encoded and decoded using simple XOR and shift operations. The simplest way to define the EVENODD code is through its encoding rules. Given a $(p-1)\times p$ information bit array, parity column P and parity column Q are filled using the rules shown in Figure 3.3, for the case $p=5$. An imaginary row, shown unshaded, is added to the array for the sake of presenting the structure of the encoding function. Parity column P is simply the bit-wise *even* parity of the information columns (4 parity groups are marked using the 4 different icons in Figure 3.3a). Parity column Q is the slope-1 diagonal parities of the information bits (whose groups are marked by icons in Figure 3.3b). Whether the parity of these diagonal groups is set to be even or odd is decided by the parity of the information bits that lie on the diagonal that was left blank in Figure 3.3b.

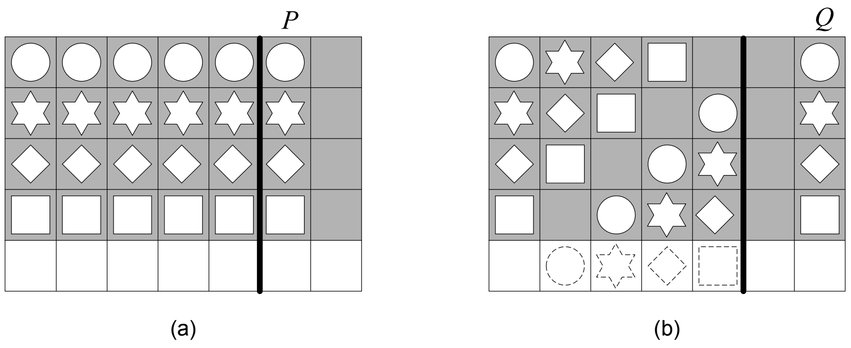


Figure 3.3: Encoding of the EVENODD code. Each array of icons specifies the encoding rules for one parity column. Each parity bit is calculated from all the information bits that carry the same icon shape. (a) Horizontal parity P (b) Diagonal parity Q

3.3.2.2 Algebraic description of EVENODD

In the previous sub-section, EVENODD codes were defined using their simple encoding functions. We now include the algebraic description of EVENODD codes from [BBBM95] that will be most useful for later proofs in this chapter. Columns of the $(p-1)\times(p+2)$ array are viewed as polynomials of degree $\leq p-2$ over $\mathbb{F}_2$ modulo the polynomial $M_p(x)$, where $M_p(x) = (x^p+1)/(x+1) = x^{p-1}+x^{p-2}+\cdots+x+1$ (recall that in $\mathbb{F}_2$ summation and subtraction are the same and both done using the boolean XOR function). According to that view, the polynomial for a column vector $\boldsymbol{c} = [c_0,\ldots,c_{p-2}]^T$ is denoted $c(\alpha) = c_{p-2}\alpha^{p-2}+\cdots+c_1\alpha+c_0$. Bit-wise addition modulo 2 of two columns is equivalent to summing the corresponding polynomials in the ring of polynomials modulo $M_p(x)$, denoted $\mathcal{R}_p$. Multiplying $c(\alpha)$ by α results in a downward shift of $\boldsymbol{c}$ if c_{p-2} is zero. In the case $c_{p-2} = 1$, reduction modulo $M_p(x)$ is needed and $\alpha c(\alpha)$ is obtained by first downward shifting $[c_0,\ldots,c_{p-3},0]^T$ and then inverting all the bits of the shifted vector. Then, it is not hard to see that the encoding rules depicted in Figure 3.3 induce the following parity check matrix over $\mathcal{R}_p$.

$$H = \left[\begin{array}{cccc|cc} 1 & 1 & \cdots & 1 & 1 & 0 \\ 1 & \alpha & \cdots & \alpha^{p-1} & 0 & 1 \end{array}\right]$$

The top row of H represents the horizontal parity constraints of P, where all columns have the same alignment. The bottom row represents the diagonal parity constraints of Q, where a column is shifted one location upwards relative to its left neighbor. The structure of the ring $\mathcal{R}_p$ provides for the even/odd adjustment of the Q parities, as a function of the parity of the bits in the blank cells. The importance of this algebraic definition of the code is due to the fact that proving correctability of an erasure combination reduces to showing that the determinant of a sub-matrix of H has an inverse in the ring $\mathcal{R}_p$. Subsequent sections will discuss that in more detail.

3.3.2.3 EVENODD for correcting 3 and more Random disk erasures

In [BBV96], EVENODD was generalized to codes that correct $r > 2$ Random erasures. The main idea in the generalization is to add more parity columns that constrain the code bits across diagonals with different slopes (recall that EVENODD uses slopes 0 and 1). Discussing EVENODD generalization in depth is beyond the scope of this chapter. We only mention the following facts that are relevant to our presentation.

- The asymptotic small-write update-complexity of the general EVENODD code family is $2r - 1 - o(1)$. $o(1)$ refers to terms that tend to zero as the code length goes to infinity. Their full-column update-complexity is r.

- For $r > 3$, generalized r Random erasure correcting EVENODD codes are only guaranteed to exist for p that belong to a subset of the primes: those that satisfy that 2 is a primitive element in the Galois field $\text{GF}(p)$.

- The best known way to decode general EVENODD codes is using the algorithm of [BR93] over $\mathcal{R}_p$, for which the decoding complexity is dominated by the term rkp.

3.3.3 The Ring $\mathcal{R}_p$

Properties of the ring $\mathcal{R}_p$ are used in subsequent sections to prove the correctability of erasure patterns by RC codes. This same ring was studied for code analysis in [BR93] and later in [BBV96]. Accordingly, the purpose of this section is to summarize useful properties of $\mathcal{R}_p$, following the necessary mathematical definitions. Recall that the elements of the ring $\mathcal{R}_p$ are polynomials with binary coefficients and degree $\leq p-2$. Addition is defined as the usual polynomial addition over $\mathbb{F}_2$ and multiplication is taken modulo $M_p(x) = \sum_{i=0}^{p-1} x^i$. Element $f(\alpha)$ is invertible in the ring if and only if $\gcd(f(x), M_p(x)) = 1$, where $\gcd(\cdot,\cdot)$ stands for the polynomial *greatest common divisor*. If $f(\alpha)$ is non-invertible ($\gcd(f(x), M_p(x)) \neq 1$), then there exists an element $g(\alpha) \in \mathcal{R}_p$ such that $f(\alpha)g(\alpha) = 0$. In that case we say that $f(\alpha), g(\alpha)$ are both divisors of 0. For convenience of notation the element $\sum_{i=0}^{p-2} \alpha^i$ is denoted α^{p-1}. Note also that $\alpha^p = 1$. The following is a useful lemma from [BR93].

Lemma 3.2 *For any prime p, all elements of the forms α^i and $\alpha^i + 1$ are invertible, for any $1 \leq i < p$.*

Next a fundamental closure property of rings is worth mentioning.

Lemma 3.3 *A product of invertible elements is invertible.*

Lemmas 3.2 and 3.3 together provide a family of ring elements that are invertible for any prime p, namely products of monomials and binomials. When the prime p has the property that 2 is a primitive element in GF(p), then $M_p(x)$ is irreducible, $\mathcal{R}_p$ becomes a *field* and consequently all non-zero elements of $\mathcal{R}_p$ are invertible. (See [LN86] for more details on finite fields). Hence, for such primes, the following Lemma (Lemma 2.7 of [BBV96]) provides additional invertible elements.

Lemma 3.4 *If a polynomial $g(x)$ has an odd number t of terms and $t < p$, then $g(\alpha)$ is invertible in $\mathcal{R}_p$, provided $M_p(x)$ is irreducible.*

3.4 Definition of the RC Codes

3.4.1 Geometric Description

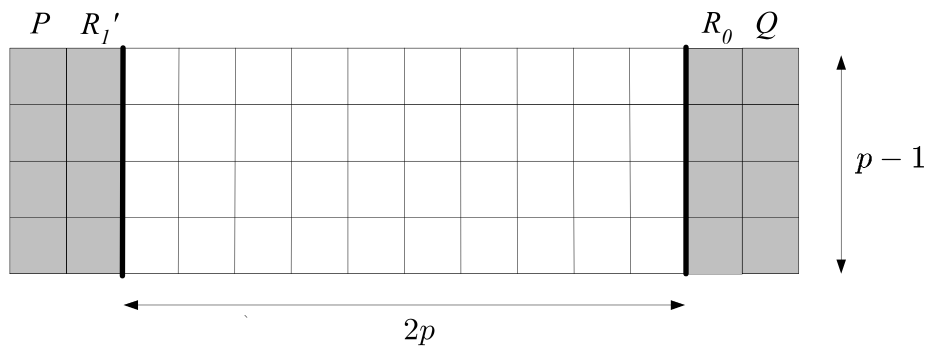


Figure 3.4: The RC-code array. RC codes have $2p$ information columns and 4 parity columns. The column size is $p-1$.

Referring to Figure 3.4, the RC code has $2p$ information columns (white) of $p-1$ bits each and 4 parity columns (shaded) with the same number of bits. The information columns are numbered in ascending order from left to right using the integers $\{0, 1, 2, \ldots, 2p-1\}$. Parity columns are not numbered and we use letter labels for them. The code is defined using its encoding rules shown in Figure 3.5, for the case $p = 5$. As before, an imaginary row is added to the array to show the structure of the encoding function. Each icon represents, similarly to the definition of EVENODD in 3.3.2.1, a group of bits that are constrained by the code to have even/odd parities. Parity column P, located in the left most column of the left parity section, is simply the bit-wise even parity of the $2p$ information columns. Parity column R_1', located second from left, is the slope -1 diagonal parity of the *odd* numbered information columns $\{1, 3, \ldots, 2p-1\}$. The bit groups of R_1' are set to have even parity if the bits marked EO have even parity, and odd parity otherwise. Parity column R_0, located in the left most column of the right parity section, is the slope 2 diagonal parity of the *even* numbered information columns $\{0, 2, \ldots, 2p-2\}$. Parity column Q, located in the right most column of the right parity section, is the XOR of the slope 1 diagonal parities of both the even numbered columns and the odd numbered columns. The parity groups of Q and R_0, similarly to those of R_1', are set to be even/odd, based on the parity of the corresponding EO groups. Note that parity columns P and Q

can be decomposed into $P = P0 \oplus P1$ and $Q = Q0 \oplus Q1$, where $P0, Q0$ depend only on even information columns and $P1, Q1$ only on odd ones.

For a formal definition of the code we write the encoding functions explicitly. Denote by $c_{i,j}$ the bit in location i in information column j. For an integer l, define $\langle l \rangle$ to be $l \pmod{p}$. Now we write the explicit expression of the parity bits.

$$P_i = \bigoplus_{j=0}^{2p-1} c_{i,j}$$

$$R'_{1_i} = S_1 \oplus \bigoplus_{j=0}^{p-1} c_{\langle i+j \rangle, 2j+1} \, ,$$

$$\text{where} \qquad S_1 = \bigoplus_{j=0}^{p-1} c_{\langle p-1+j \rangle, 2j+1}$$

$$R_{0_i} = S_0 \oplus \bigoplus_{j=0}^{p-1} c_{\langle i-2j \rangle, 2j} \, ,$$

$$\text{where} \qquad S_0 = \bigoplus_{j=0}^{p-1} c_{\langle p-1-2j \rangle, 2j}$$

$$Q_i = S_Q \oplus (\bigoplus_{j=0}^{p-1} c_{\langle i-j \rangle, 2j}) \oplus (\bigoplus_{j=0}^{p-1} c_{\langle i-j \rangle, 2j+1}) \, ,$$

$$\text{where} \qquad S_Q = (\bigoplus_{j=0}^{p-1} c_{\langle p-1-j \rangle, 2j}) \oplus (\bigoplus_{j=0}^{p-1} c_{\langle p-1-j \rangle, 2j+1})$$

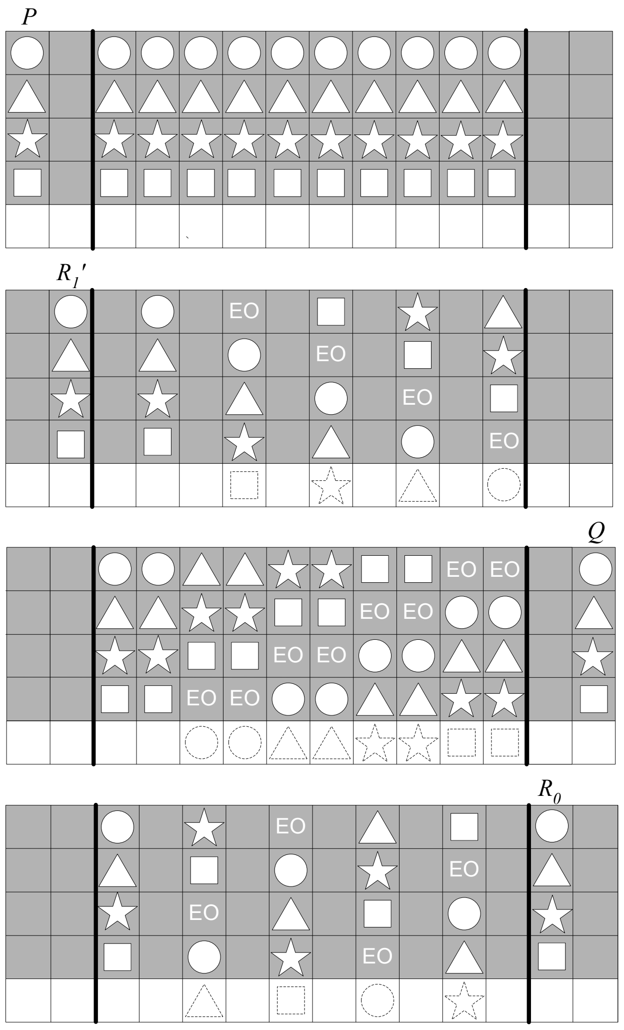


Figure 3.5: Encoding of the RC code. From top to bottom: the parity groups of parity columns P (slope 0), R_1' (slope -1), Q (slope 1) and R_0 (slope 2). Parity columns R_0 and R_1' each depends on only half of the columns, contributing to the low implementation complexity of RC codes.

3.4.2 Algebraic Description

Using the ring $\mathcal{R}_p$, the parity check matrix H of the RC code for $p = 5$ is given by

$$\left[\begin{array}{cc|cccccccccc|cc}
1 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & \alpha^4 & 0 & \alpha^3 & 0 & \alpha^2 & 0 & \alpha & 0 & 0 \\
0 & 0 & 1 & 0 & \alpha^2 & 0 & \alpha^4 & 0 & \alpha & 0 & \alpha^3 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 & \alpha & \alpha & \alpha^2 & \alpha^2 & \alpha^3 & \alpha^3 & \alpha^4 & \alpha^4 & 0 & 1
\end{array}\right]$$

The correspondence between the encoding function in Figure 3.5 and the parity check matrix above is straight forward. The columns of the parity check matrix correspond to columns of the code array. The two left most columns are for parity columns P and R'_1 and the two right most columns are for R_0 and Q. Columns in between correspond to information columns in the array. In the parity check matrix, row 1 represents the constraints enforced by parity column P, rows $2, 3, 4$ similarly represent the parity constraints of R'_1, R_0, Q, respectively. In any row j, the difference of exponents of α in two different columns is exactly the relative vertical shift of the two columns in the icon layout of the appropriate parity in Figure 3.5. For example, in the top row, all information columns have the same element, $1 (= \alpha^0)$, to account for the identical vertical alignment of the icons in the encoding rule of parity P. For general p the parity check matrix H has the following form.

$$H = \left[\begin{array}{cc|cccccccc|cc} 1 & 0 & 1 & 1 & \cdots & 1 & 1 & 1 & 1 & \cdots & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & \cdots & 0 & \alpha^{-i} & 0 & \alpha^{-(i+1)} & \cdots & \alpha & 0 & 0 \\ 0 & 0 & 1 & 0 & \cdots & \alpha^{2i} & 0 & \alpha^{2(i+1)} & 0 & \cdots & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & \cdots & \alpha^{i} & \alpha^{i} & \alpha^{i+1} & \alpha^{i+1} & \cdots & \alpha^{p-1} & 0 & 1 \end{array}\right] \tag{3.1}$$

After presenting the RC code family, we proceed in the next section to prove its Random and Clustered erasure correction capabilities.

3.5 Erasure Correctability of RC Codes

In this section we prove that essentially all Clustered combinations of 4 erasures are correctable by RC codes. Moreover, considering Random erasure correctability, we prove that a $7/8$ portion of *all* combinations of 4 erasures are correctable by RC codes.

3.5.1 Clustered erasure correctability

We first prove RC codes' excellent correction capability of Clustered erasures. This result is established in Theorems 3.8 and 3.9 below that follow a sequence of lemmas. Recall that the $2p + 4$ columns of the RC codes are labeled $\{P, R'_1, 0, 1, \ldots, 2p-2, 2p-1, R_0, Q\}$.

Lemma 3.5 *For any prime p, for a combination of 4 erasures, if 3 columns are either even numbered information columns or parity columns in $\{R_0, P, Q\}$, and 1 column is an odd numbered information column or the parity column R_1', then it is a correctable 4-erasure. The complement case, 3 odd (or R_1' or P or Q) and 1 even (or R_0), is correctable as well. (in particular, any 3-erasure is correctable).*

Proof: The RC code can correct the erasure patterns under consideration using a two-step procedure. The first step is to recover the erased odd information column. Since only one odd column is erased, parity column R_1' can be used to easily recover all of its bits. Then, when all odd columns are available, $P1$ and $Q1$ are computed, and used to find $P0$ and $Q0$ from P and Q (if not erased) by

$$P0 = P1 \oplus P \quad , \quad Q0 = Q1 \oplus Q$$

After that step, between even information columns, R_0, $P0$ and $Q0$, only 3 columns are missing. Since even columns, R_0, $P0$ and $Q0$ constitute an EVENODD code with $r = 3$, the 3 missing columns can be recovered. The complement case of 3 odd and 1 even column erasures is settled by observing that odd columns, R_1', $P1$ and $Q1$ constitute an $r = 3$ MDS code [HX05]. □

The next Lemma presents the key property that gives RC codes their favorable Random and Clustered erasure correctability.

Lemma 3.6 *For any prime p such that $p > 5$ and 2 is primitive in $GF(p)$, for a combination of 4 erasures, if 2 columns are even numbered information columns and 2 columns are odd numbered information columns, then it is a correctable 4-erasure.*

Proof: For the case of 2 even and 2 odd information column erasures we consider two erasure patterns. All possible erasure combinations of that kind are either covered by these patterns directly or are equivalent to them in a way discussed below. The discussion of each erasure pattern will commence with its representing diagram. In these diagrams, a column marked 0 represents an even column and a column marked 1 represents an odd column. Between each pair of columns, an expression for the number of columns that separate them is given .

a) Erasures of the form

0	$\leftarrow 2j \rightarrow$	1	$\leftarrow 2(k-1) \rightarrow$	0	$\leftarrow 2l \rightarrow$	1	

The variables j, k, l satisfy the following conditions: $1 \leq k$, $1 \leq j + k + l \leq p - 1$.

The location of the first even erased column, together with j, k, l determine the locations of the 4 erasures. Any even cyclic shift of the diagram above does not change the correctability of the erasure pattern since this shift gives the same sub-matrix of the parity-check matrix, up to a non-zero multiplicative constant. Hence, we can assume, without loss of generality, that the first even erased column is located in the leftmost information column. To prove the correctability of this erasure pattern we examine the determinant (over $\mathcal{R}_p$) of the square sub-matrix of H, that corresponds to the erased columns. This determinant is itself an element of $\mathcal{R}_p$ and if it is provably *invertible* in $\mathcal{R}_p$, for every p that satisfies the conditions of the lemma, then the erasure combination is correctable by the RC code.

The sub-matrix that corresponds to the erasure pattern above is

$$\mathcal{M}_a^{(j,k,l)} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & \alpha^{-j} & 0 & \alpha^{-j-k-l} \\ 1 & 0 & \alpha^{2(j+k)} & 0 \\ 1 & \alpha^{j} & \alpha^{j+k} & \alpha^{j+k+l} \end{bmatrix} .$$

Evaluating the determinant of this matrix gives

$$\begin{aligned}\left|\mathcal{M}_a^{(j,k,l)}\right| &= \alpha^{2j+3k+l}+\alpha^{2j+k-l}+\alpha^{j+2k}+\alpha^{j+k-l}+\\ &\quad+\alpha^{k+l}+\alpha^{k}+\alpha^{-l}+\alpha^{-k-l}\\ &= \alpha^{-l}(\alpha^{j+k}+1)(\alpha^{k+l}+1)\cdot\\ &\quad\cdot(\alpha^{j+k+l}+\alpha^{j}+\alpha^{l}+1+\alpha^{-k})\end{aligned}$$

The first three factors in the product are invertible for any legal j,k,l and any prime p by Lemma 3.2. The last factor is invertible for all j,k,l and any $p>5$ such that 2 is primitive in GF(p), by Lemma 3.4. Furthermore, for $p=5$, checking all possible assignments of j,k,l and verifying that the last factor does not equal $M_5(x)$, we conclude that this pattern is correctable for $p=5$ as well.

b) Erasures of the form

0	$\leftarrow 2j-1 \rightarrow$	0	$\leftarrow 2k \rightarrow$	1	$\leftarrow 2l-1 \rightarrow$	1	

The variables j,k,l satisfy the following conditions: $1\le j$, $1\le l$, $1\le j+k+l\le p-1$.

Here, like in the previous pattern, we assume, without loss of generality, that the first even erased column is located in the leftmost information column.

The sub-matrix that corresponds to the erasure pattern above is

$$\mathcal{M}_b^{(j,k,l)}=\begin{bmatrix}1 & 1 & 1 & 1\\ 0 & 0 & \alpha^{-j-k} & \alpha^{-j-k-l}\\ 1 & \alpha^{2j} & 0 & 0\\ 1 & \alpha^{j} & \alpha^{j+k} & \alpha^{j+k+l}\end{bmatrix}.$$

Evaluating the determinant of this matrix gives

$$\begin{aligned}\left|\mathcal{M}_b^{(j,k,l)}\right| &= \alpha^{2j+l}+\alpha^{2j-l}+\alpha^{j-k}+\alpha^{j-k-l}+\\ &\quad +\alpha^{l}+\alpha^{-l}+\alpha^{-k}+\alpha^{-k-l}\\ &= (\alpha^j+1)(\alpha^l+1)(\alpha^j+\alpha^{j-l}+1+\alpha^{-l}+\alpha^{-k-l})\end{aligned}$$

The first two factors in the product are invertible for any legal j,k,l and any prime p by Lemma 3.2. The last factor is invertible for all j,k,l and any $p>5$ such that 2 is primitive in GF(p), by Lemma 3.4. □

The next Lemma treats additional erasure combinations that include parity columns and that are not covered by Lemma 3.5.

Lemma 3.7 *For any prime p such that $p>3$ and 2 is primitive in GF(p), the following 4-erasure combinations are correctable:*

1. R_1', 1 *odd information column and* 2 *even information columns*
2. R_0, 1 *even information column and* 2 *odd information columns*
3. R_0,R_1', 1 *even information column and* 1 *odd information column, except pairs of information columns numbered* $2i,2i+1$, *respectively, for* $1\le i\le p$.

Proof: The sub-matrix that corresponds to case 1 is, up to a multiplicative non-zero constant,

$$\mathcal{M}_1^{(j,k)} = \begin{bmatrix} 1 & 1 & 1 & 0\\ 0 & \alpha^{-j} & 0 & 1\\ 1 & 0 & \alpha^{2(j+k)} & 0\\ 1 & \alpha^{j} & \alpha^{j+k} & 0 \end{bmatrix}.$$

The variables j,k satisfy the following conditions: $1\le k$, $1\le j+k\le p-1$. Evaluating the determinant of this matrix gives

$$\left|\mathcal{M}_1^{(j,k)}\right| = \alpha^j(\alpha^{j+k}+1)(\alpha^{j+k}+\alpha^k+1),$$

an invertible element if $p > 3$.

The sub-matrix that corresponds to case 2 is, up to a multiplicative non-zero constant,

$$\mathcal{M}_2^{(j,k)} = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & \alpha^{-j} & \alpha^{-j-k} & 0 \\ 1 & 0 & 0 & 1 \\ 1 & \alpha^{j} & \alpha^{j+k} & 0 \end{bmatrix}.$$

The variables j, k satisfy the following conditions: $1 \leq k$, $1 \leq j+k \leq p-1$. The determinant now equals

$$\left|\mathcal{M}_2^{(j,k)}\right| = \alpha^{-j-k}(\alpha^k+1)(\alpha^{j+k}+\alpha^j+1),$$

an invertible element if $p > 3$.

The sub-matrix that corresponds to case 3 is, up to a multiplicative non-zero constant,

$$\mathcal{M}_3^{(j)} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & \alpha^{-j} & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & \alpha^{j} & 0 & 0 \end{bmatrix},$$

whose determinant equals

$$\left|\mathcal{M}_3^{(j)}\right| = \alpha^j + 1,$$

an invertible element if $p > 3$ and if $j > 0$. The latter condition is equivalent to requiring that the even and the odd information columns are not numbered $2i, 2i+1$, respectively, for $1 \leq i \leq p$. $\square$

Finally, we are ready to prove the main result of this sub-section. RC codes are next shown to correct all 4-erasures in up to two clusters, and almost all 4-erasures in three clusters. Given the Lemmas above, establishing these results is rather straightforward.

Theorem 3.8 *For any prime p such that $p > 3$ and 2 is primitive in $GF(p)$, RC codes correct all 4-erasures that fall into at most two clusters.*

Proof: If a 4-erasure falls into two or less clusters, then it either has 2 even and 2 odd columns or 3 even and 1 odd columns (or the complement). If P or Q is erased, then the remaining 3 columns cannot be all odd or all even. Lemmas 3.5, 3.6 and 3.7 address all such combinations, except the two special cases $\{R'_1, 2, 3, R_0\}$ and $\{R'_1, 2p, 2p+1, R_0\}$. These combinations can be addressed by internal reordering of even information columns in a way that does not affect any of the other proved properties of the code. Also note that here we only required $p > 3$, compared to $p > 5$ in Lemma 3.6, since the only 4-erasure that gives a non-invertible determinant of $\mathcal{M}_b$ for $p = 5$ falls into three clusters (see the proof of Lemma 3.6). □

Theorem 3.9 *For any prime p such that $p > 5$ and 2 is primitive in $GF(p)$, the ratio between the number of RC-correctable 4-erasures that fall into three clusters and the total number of 4-erasures with three clusters is greater than 0.9696. As p goes to infinity, this ratio tends to 1.*

Proof: A 4-erasure with three clusters has two clusters of size 1 and one cluster of size 2. If a 4-erasure falls into three clusters, then it either has 2 even and 2 odd columns or 3 even and 1 odd columns (or the complement). Lemmas 3.5, 3.6 and 3.7 address all such combinations, except the following special cases. $\{R'_1, 2i, 2i+1, R_0\}$ cannot be corrected as it is not covered by case 3) of Lemma 3.7. Also, $\{P, R'_1, 2i+1, 2j+1\}$ and $\{2i, 2j, R_0, Q\}$ cannot be corrected as they are excluded from Lemma 3.5.

Hence the number of non-correctable 4-erasures with three clusters is

$$p + 2\binom{p}{2}$$

The total number of 4-erasures with three clusters is

$$3\binom{2p-1}{3}$$

(in general this equals $3\binom{n-3}{3}$ for length n arrays since by taking any choice of 3 points on a length $n-3$ line, we can uniquely obtain an erasure combination with three clusters, following the procedure below. We first choose 3 points from the $n-3$ line to be the cluster locations. Then the point that represents the cluster with size 2 is selected from these 3 points (for that we have the factor 3). Given these choices, the 3 clusters are obtained by augmenting the size 2 cluster with an additional point to its right and in addition augmenting each of the two left points with a point to its right as a cluster spacer.)

Thus the ratio between the number of correctable such 4-erasures and the total number of such 4-erasures equals

$$\begin{aligned}\frac{3\binom{2p-1}{3} - p - 2\binom{p}{2}}{3\binom{2p-1}{3}} &= 1 - \frac{p^2}{4p^3 - 12p^2 + 11p - 3} \\ &= 1 - \frac{9}{8p-12} - \frac{1}{8p-4} + \frac{1}{p-1}.\end{aligned}$$

For $p = 11$, the ratio attains its minimal value of 0.9696. Moreover, it is readily seen that this ratio equals $1 - o(1)$, while $o(1)$ are terms that tend to zero as p goes to infinity. □

3.5.2 Random erasure correctability

RC codes are next shown to correct a $7/8$ portion of all combinations of 4 erasures.

Theorem 3.10 *For any prime p such that $p > 5$ and 2 is primitive in GF(p), the ratio between the number of RC-correctable 4-erasures and the total number of 4-erasures is greater than 0.865. As p goes to infinity, this ratio tends to $7/8 = 0.875$.*

Proof: Building on Lemmas 3.5, 3.6 and 3.7, the number of correctable 4-erasures equals

$$\begin{aligned}&\overbrace{2\binom{p+3}{3}(p+1) - (p+1)^2}^{(1)} + \overbrace{\binom{p}{2}\binom{p}{2}}^{(2)} \\ &\qquad + \underbrace{2p\binom{p}{2}}_{(3)} + \underbrace{p(p-1)}_{(4)}\end{aligned}$$

(1), obtained by Lemma 3.5, is the number of ways to select 3 even information columns (or R_0 or P or Q) and 1 odd information column (or R'_1), multiplied by 2 to include the complement case, and subtracting the doubly counted combinations with both P and Q.

(2), obtained by Lemma 3.6, is the number of ways to select 2 even and 2 odd information columns.

(3), obtained by 1) and 2) of Lemma 3.7, is the number of ways to select 2 even information columns and 1 odd information column, multiplied by 2 to include the complement case.

(4), obtained by 3) of Lemma 3.7, is the number of ways to select an even information column $2i$ and an odd information column which is not $2i+1$.

The total number of 4-erasure combinations is

$$\binom{2p+4}{4}$$

Taking the ratio of the two we obtain

$$\frac{7p^4 + 34p^3 + 59p^2 + 32p + 12}{8p^4 + 40p^3 + 70p^2 + 50p + 12}$$

For $p = 11$, the ratio attains its minimal value of 0.865. Moreover, it is readily seen that this ratio equals $7/8 - o(1)$, while $o(1)$ are terms that tend to zero as p goes to infinity. $\square$

3.6 Efficient Decoding of Erasures

In the previous section, the decodability of Clustered and Random erasures was proved by algebraic reasoning. In this section we take a more constructive path and study simple and efficient ways to decode Random and Clustered erasures. The purpose of this analysis is to prove that decoding the RC code can be done using $3kp + o(p^2)$ bit operations, while the best known algorithm for a 4-erasure correcting MDS code is $4kp + o(p^2)$ [BR93]. Since k is taken to be in the order of p, saving about kp bit operations gives a quadratic (in p) savings in computations that is very significant in practice for large p.

For the sake of the analysis, we only consider erasure of 4 *information* columns since these are the most common and most challenging cases. We moreover only consider erasures of two even columns and two odd columns, since for RC codes, the three even and one odd case (or three odd and one even), reduces to a correction of three even (or odd) erasures, preceded by a simple parity completion for the single odd (or even) column erasure. A very simple and efficient decoder for three odd-column erasures can be obtained by using the decoder of the STAR code, given in [HX05], and a same-complexity modification of that decoder can be used for the case of three even-column erasures. Throughout the section we will assume that one of the erased columns is the leftmost even information column, as all other cases are cyclically equivalent.

3.6.1 Description of 4-erasure decoding algorithm

A general 4-erasure can be decoded using a straightforward procedure over $\mathcal{R}_p$. Ways to perform the steps of that procedure in an efficient way are the topic of the next subsection. The erased symbols, which represent the content of the erased array columns, are denoted by $\{e_1, o_1, e_2, o_2\}$. e_1, e_2 have even locations and o_1, o_2 have odd locations. First the syndrome vector $\boldsymbol{s}$ of the array is calculated by taking the product

$$\boldsymbol{s} = H\boldsymbol{r}$$

where $\boldsymbol{r}$ is the length $2p+4$ column vector over $\mathcal{R}_p$ that represents the array contents, with erased columns set to the zero element. Then the erased columns can be directly recovered by

$$\begin{bmatrix} e_1 \\ o_1 \\ e_2 \\ o_2 \end{bmatrix} = E^{-1}\boldsymbol{s} \tag{3.2}$$

where E denotes the 4×4 sub-matrix of H that corresponds to the 4 erased columns' locations:

$$E = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & \alpha_1^{-1} & 0 & \alpha_3^{-1} \\ 1 & 0 & \alpha_2^2 & 0 \\ 1 & \alpha_1 & \alpha_2 & \alpha_3 \end{bmatrix}.$$

Recall from (3.1) that each α_i is an element in $\mathcal{R}_p$ of the form α^{l_i}, for some $0 < l_i < p$. Therefore, E can be written as

$$E = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & \alpha^{-u} & 0 & \alpha^{-w} \\ 1 & 0 & \alpha^{2v} & 0 \\ 1 & \alpha^{u} & \alpha^{v} & \alpha^{w} \end{bmatrix}.$$

The inverse of E, which is used in (3.2) to decode erasures, is now given in a closed form

$$E^{-1} = \left(\alpha^u + \alpha^v + \alpha^w + \alpha^{u+v} + \alpha^{v+w}\right)^{-1} \cdot \begin{bmatrix} 1+\alpha^v & 0 & 0 & 0 \\ 0 & \alpha^u + \alpha^w & 0 & 0 \\ 0 & 0 & 1+\alpha^v & 0 \\ 0 & 0 & 0 & \alpha^u + \alpha^w \end{bmatrix}^{-1} \cdot$$

$$\cdot \begin{bmatrix} \alpha^{2v}(\alpha^u + \alpha^w) & \alpha^{u+2v+w} & \alpha^u + \alpha^v + \alpha^w & \alpha^{2v} \\ \alpha^{u+v} & \alpha^{u+w}(\alpha^v + \alpha^w + \alpha^{v+w}) & \alpha^u & \alpha^u(1+\alpha^v) \\ \alpha^u + \alpha^w & \alpha^{u+w} & 1+\alpha^u+\alpha^w & 1 \\ \alpha^{v+w} & \alpha^{u+w}(\alpha^u + \alpha^v + \alpha^{u+v}) & \alpha^w & \alpha^w(1+\alpha^v) \end{bmatrix}$$

From (3.2) and the closed-form expression above, the erased symbol e_1 can be recovered by the following product

$$e_1 = \left[\left(\alpha^u + \alpha^v + \alpha^w + \alpha^{u+v} + \alpha^{v+w}\right) \cdot (1+\alpha^v)\right]^{-1} \cdot$$
$$\cdot \left[\ \alpha^{2v}(\alpha^u + \alpha^w),\ \ \alpha^{u+2v+w},\ \ \alpha^u + \alpha^v + \alpha^w,\ \ \alpha^{2v}\ \right] \cdot s$$

Once e_1 is known, e_2 can be recovered using a simple parity completion with the aid of parity column R_0. The bits of the odd columns are then recovered by a chain of XOR operations with the aid of parity columns P, Q, that can now be adjusted to $P1, Q1$ when all even columns are known.

Calculating e_1 then reduces to the following chain of calculations

1. Finding the inverse of $(\alpha^u + \alpha^v + \alpha^w + \alpha^{u+v} + \alpha^{v+w})(1+\alpha^v)$ over $\mathcal{R}_p$.

2. Multiplication of sparse $\mathcal{R}_p$ elements by dense $\mathcal{R}_p$ elements. The sparse elements are the four elements from the E matrix (that have a small (≤ 3) constant number of non-zero monomials, for any p) and the dense elements are the four syndrome elements.

3. Multiplication of two dense $\mathcal{R}_p$ elements resulting from the previous steps.

3.6.2 Analysis of 4-erasure decoding algorithm

We now analyze the number of bit operations required for each decoding task.

1. **Finding inverses of $\mathcal{R}_p$ elements**:
   The inverse of an element $f(\alpha) \in \mathcal{R}_p$ is the element $\tilde{f}(\alpha)$ that satisfies $\tilde{f}(x)f(x) + a(x)M_p(x) = 1$, for some polynomial $a(x)$. When $f(\alpha)$ is invertible, the polynomial $\tilde{f}(x)$ can be found by the Extended Euclid Algorithm for finding the greatest common divisor of the polynomials $f(x)$ and $M_p(x)$. An efficient algorithm for polynomial greatest common divisors is given in [AHU74, Ch.8] that requires $O(p\log^4 p)$ bit operations ($O(\log p)$ polynomial multiplications, each taking $O(p\log^3 p)$ bit operations, as shown in item 3 below).

2. **Multiplication of a sparse $\mathcal{R}_p$ element by a dense $\mathcal{R}_p$ element** requires $O(p)$ bit operations. Since the number of non-zero monomials in the sparse polynomial is constant in p, the trivial polynomial multiplication algorithm requires $O(p)$ shifts and additions modulo 2.

3. **Multiplication of two dense $\mathcal{R}_p$ elements** can be done in $O(p \log^3 p)$ bit operations using Fourier domain polynomial multiplication. We describe this procedure for the special case of polynomial coefficients over GF(2). Let $N \geq 2p - 2$ be the smallest such integer of the form $N = 2^\ell - 1$. Let ω be a principal Nth root of unity in the finite field GF(2^ℓ). Then ω defines a Discrete Fourier Transform on length N vectors over GF(2^ℓ). The product of two polynomials of degree $p - 2$ or less can be obtained by element-wise multiplication of their individual Discrete Fourier Transforms, and then applying the Inverse Discrete Fourier Transform to the resulting length N vector. Using the FFT algorithm, each transformation requires $O(N \log N)$ operations over GF(2^ℓ), or $O(N \log^3 N)$ bit operations. The element-wise multiplication requires N multiplications over GF(2^ℓ), or $O(N \log^2 N)$ bit operations. Since $N < 4p$, the total number of bit operations needed for multiplying two dense $\mathcal{R}_p$ elements is $O(p \log^3 p)$.

3.7 Reliability Analysis of RC-code Protected Disk Arrays

The main motivation for the construction of array codes in general, and RC codes in particular, is to provide efficient protection for disk arrays against device failures. The benefit of deploying an array code in a practical storage system obviously lies in the trade-off it achieves between erasure correction capability and implementation complexity. To this end, the correction capability characterization of RC codes, and their benefits in implementation complexity were derived in concrete terms that can be clearly specified to the storage-system operator. However, to achieve an effective data protection using those codes, one needs to instill these specifications into a statistical framework for analyzing the *reliability* of the stored data. For a time instance t, the reliability of a disk array is defined as the probability that no data has been lost at time t [Gib92]. Finding the full reliability distribution for all times t is hard except for very simple protection structures. Therefore, the *expected* time before data loss, denoted $MTTDL$ (Mean Time To Data Loss), is used in practice as a quantitative indicator for the system reliability. Ultimately, this section will detail a procedure to find the $MTTDL$ of RC-code protected disk arrays in the presence of

Random and Clustered device failures. This will be done after first presenting the general method of $MTTDL$ calculation as applied in the literature to MDS codes under Random failures.

3.7.1 $MTTDL$ calculation for MDS codes under Random failures

Using the method presented in [Gib92, Ch.5] for single-erasure-correcting arrays under Random failures (termed *Independent disk lifetimes* therein), we calculate the $MTTDL$ of all-4-erasure-correcting arrays as an example that will be later used for comparison with RC codes. The direct calculation of the $MTTDL$ becomes a simpler task if disk failures and repairs follow a Markov process and can thus be described by Markov state diagram. To allow that, the following assumptions are made.

- Disk lifetimes follow an exponential distribution with equal mean[1] $MTTF_{\text{disk}} = 1/\lambda$.
- Repair times are also exponential with mean $MTTR_{\text{disk}} = 1/\mu$.
- The number of disks is large compared to the number of tolerable failures so the transition probabilities between states do not depend on the instantaneous number of failed disks.

When those assumptions are met, the reliability of a disk array can be described by the state diagram shown in Figure 3.6. The label of each state represents the number of failed

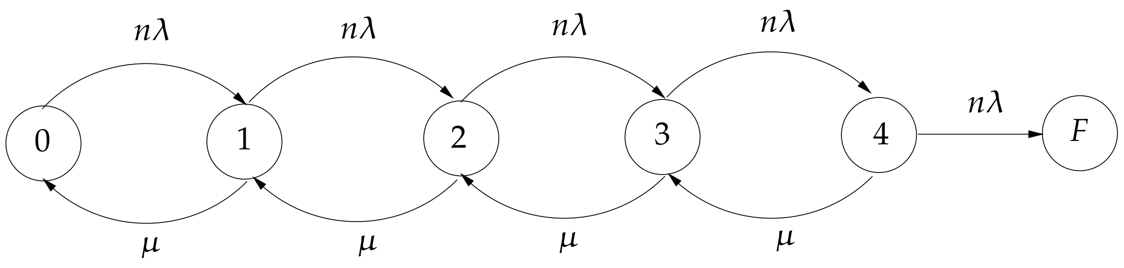

$$\begin{tikzcd}
0 \arrow[r, bend left, "n\lambda"] & 1 \arrow[l, bend left, "\mu"] \arrow[r, bend left, "n\lambda"] & 2 \arrow[l, bend left, "\mu"] \arrow[r, bend left, "n\lambda"] & 3 \arrow[l, bend left, "\mu"] \arrow[r, bend left, "n\lambda"] & 4 \arrow[l, bend left, "\mu"] \arrow[r, "n\lambda"] & F
\end{tikzcd}$$

Figure 3.6: State diagram description of all-4-erasure correcting arrays under Random failures. The failure process with rate $n\lambda$ moves to a higher failure state. The repair process with rate μ moves to a lower failure state.

[1]MTTF stands for Mean Time To Failure while MTTR stands for Mean Time To Repair

disks. State F (Fail) represents permanent data loss resulting from a failure count that is above the array tolerance. The exponential distributions allow specifying the transitions between states in terms of *rates*. The transition rate from lower to higher states is the inverse $MTTF_{\text{disk}}$ of individual disks, times the number of disks in the array. The reverse transitions that represent repairs have rates that are the inverse $MTTR_{\text{disk}}$ assumed in the system. Using the state diagram, the $MTTDL$ is the expected time beginning in state 0 and ending on the transition into state F.

$$MTTDL \triangleq E[0 \to F]$$

The Markov property of the process permits the decomposition

$$E[0 \to F] = E[\text{time stays in } 0] + E[1 \to F] = \frac{1}{n\lambda} + E[1 \to F]$$

Linear relationships between $E[i \to F]$ and $E[j \to F]$ are induced whenever state i and state j are connected. The $MTTDL$ is then obtained as the solution (for $E[0 \to F]$) of the following linear system.

$$\begin{bmatrix} 1 & -1 & 0 & 0 & 0 \\ -\frac{\mu}{\mu+n\lambda} & 1 & -\frac{n\lambda}{\mu+n\lambda} & 0 & 0 \\ 0 & -\frac{\mu}{\mu+n\lambda} & 1 & -\frac{n\lambda}{\mu+n\lambda} & 0 \\ 0 & 0 & -\frac{\mu}{\mu+n\lambda} & 1 & -\frac{n\lambda}{\mu+n\lambda} \\ 0 & 0 & 0 & -\frac{\mu}{\mu+n\lambda} & 1 \end{bmatrix} \begin{bmatrix} E[0 \to F] \\ E[1 \to F] \\ E[2 \to F] \\ E[3 \to F] \\ E[4 \to F] \end{bmatrix} = \begin{bmatrix} \frac{1}{n\lambda} \\ \frac{1}{\mu+n\lambda} \\ \frac{1}{\mu+n\lambda} \\ \frac{1}{\mu+n\lambda} \\ \frac{1}{\mu+n\lambda} \end{bmatrix}$$

that is found to be

$$MTTDL_{\text{MDS4}} = \frac{1}{\Lambda^5}(5\Lambda^4 + 4\mu\Lambda^3 + 3\mu^2\Lambda^2 + 2\mu^3\Lambda + \mu^4)$$

where $\Lambda \triangleq n\lambda$ was used for notational convenience.

3.7.2 $MTTDL$ calculation for RC codes under Random and Clustered failures

For the model of Random failures, the $MTTDL$ of RC codes can be calculated by a straightforward application of the method in the previous sub-section – executed on the transition diagram of Figure 3.7.

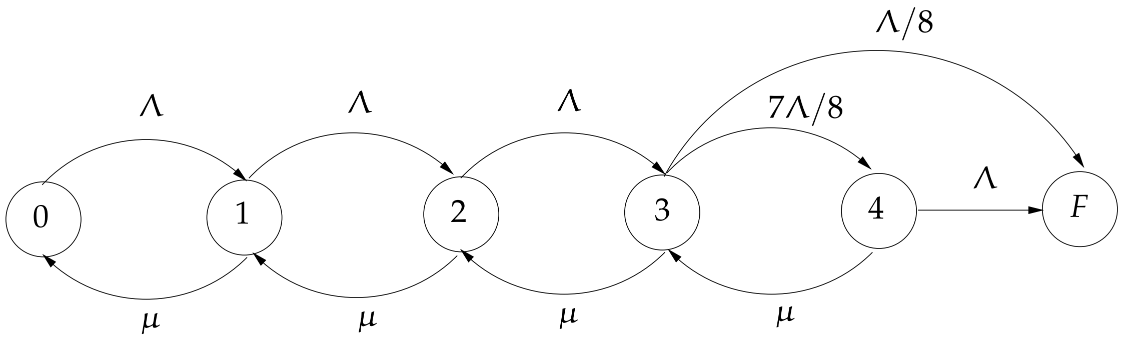


Figure 3.7: State diagram description of RC-coded arrays under Random failures. Since an RC code corrects only a $7/8$ ratio of 4-erasures, the failure rate out of state 3 is split to two rates with different terminal states.

The corresponding linear system of equations on the 5 active states $0, 1, 2, 3, 4$ is

$$\begin{bmatrix} 1 & -1 & 0 & 0 & 0 \\ -\frac{\mu}{\mu+\Lambda} & 1 & -\frac{\Lambda}{\mu+\Lambda} & 0 & 0 \\ 0 & -\frac{\mu}{\mu+\Lambda} & 1 & -\frac{\Lambda}{\mu+\Lambda} & 0 \\ 0 & 0 & -\frac{\mu}{\mu+\Lambda} & 1 & -\frac{7}{8}\cdot\frac{\Lambda}{\mu+\Lambda} \\ 0 & 0 & 0 & -\frac{\mu}{\mu+\Lambda} & 1 \end{bmatrix} \begin{bmatrix} E[0 \to F] \\ E[1 \to F] \\ E[2 \to F] \\ E[3 \to F] \\ E[4 \to F] \end{bmatrix} = \begin{bmatrix} \frac{1}{\Lambda} \\ \frac{1}{\mu+\Lambda} \\ \frac{1}{\mu+\Lambda} \\ \frac{1}{\mu+\Lambda} \\ \frac{1}{\mu+\Lambda} \end{bmatrix}$$

The solution of that system gives

$$MTTDL_{\text{RC,random}} = \frac{39\Lambda^4 + 35\mu\Lambda^3 + 26\mu^2\Lambda^2 + 17\mu^3\Lambda + 8\mu^4}{8\Lambda^5 + \mu\Lambda^4}$$

The exact $MTTDL$ calculations are now used to compare the reliability of RC codes to the reliabilities of all-4-erasure and all-3-erasure correcting codes. For the comparison, Λ is fixed to be $100/8760$[1/hr], which applies e.g. to an array with 100 disks and $MTTF_{\text{disk}} =$ 1[Year]. The $MTTDL$ in hours ([hr]) is then calculated for repair rates μ between 0.01[1/hr] and $10_{[1/\text{hr}]}$. The graph shows that RC codes outperform 3-Random failure codes by an order

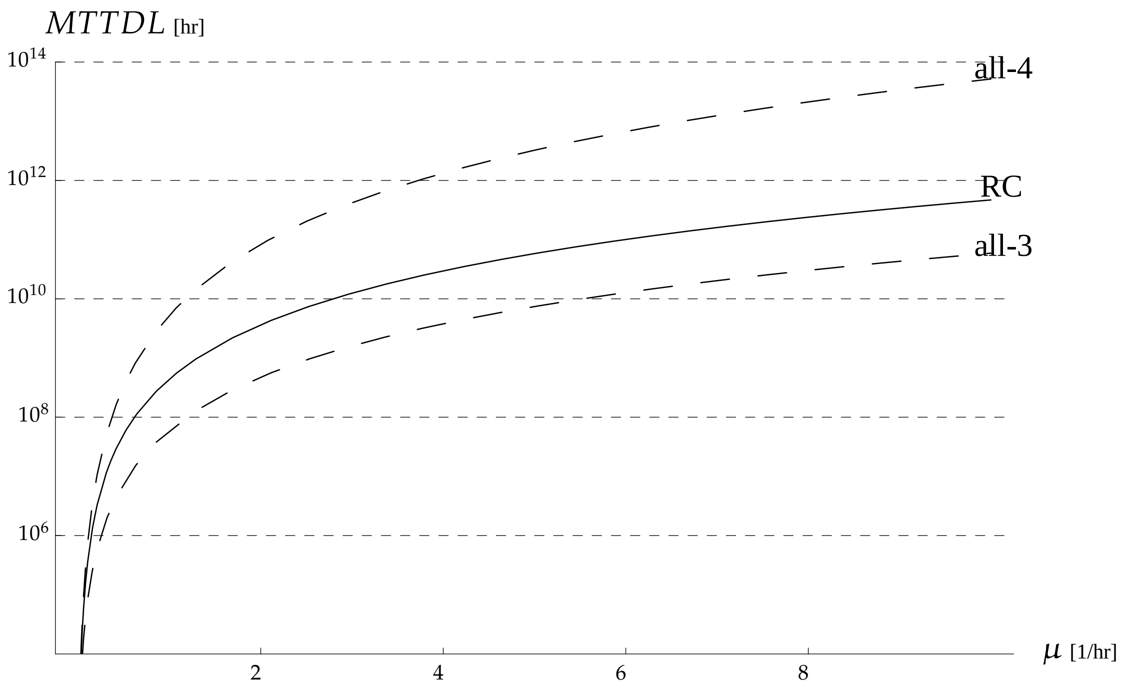


Figure 3.8: $MTTDL$ curves under Random failures for RC codes, all-3-erasure and all-4-erasure correcting codes. Under Random failures, RC codes are order of magnitude better than all-3-erasure correcting codes, and two orders of magnitude inferior to all-4-erasure correcting codes.

of magnitude, despite having the same encoding complexity, the same update complexity and asymptotically the same decoding complexity. However, when Random failures only is assumed, RC codes are still two orders of magnitude below 4-Random failure-correcting codes.

To compare RC codes and 4-Random failure codes in the presence of both Random and Clustered codes, the state diagram of RC codes in Figure 3.7 needs to be modified to comprise additional states that represent Clustered failures. The state diagram of 4-Random failure codes in Figure 3.6 remains the same since this code is oblivious to the distinction between Random and Clustered failures. To take Clustered failures into account in the Markov failure model, we add the following assumptions to those of the previous sub-section.

- Times to Clustered failures (failures that are adjacent to an unrepaired previous failure) are exponentially distributed with mean $1/\chi$.

- The exponentially distributed repair process eliminates isolated failures before Clustered ones.

With these assumptions, the state diagram of RC-code-protected arrays with Random and Clustered failures is given in Figure 3.9. States $2'$,$3'$ and $4'$ in the upper branch represent

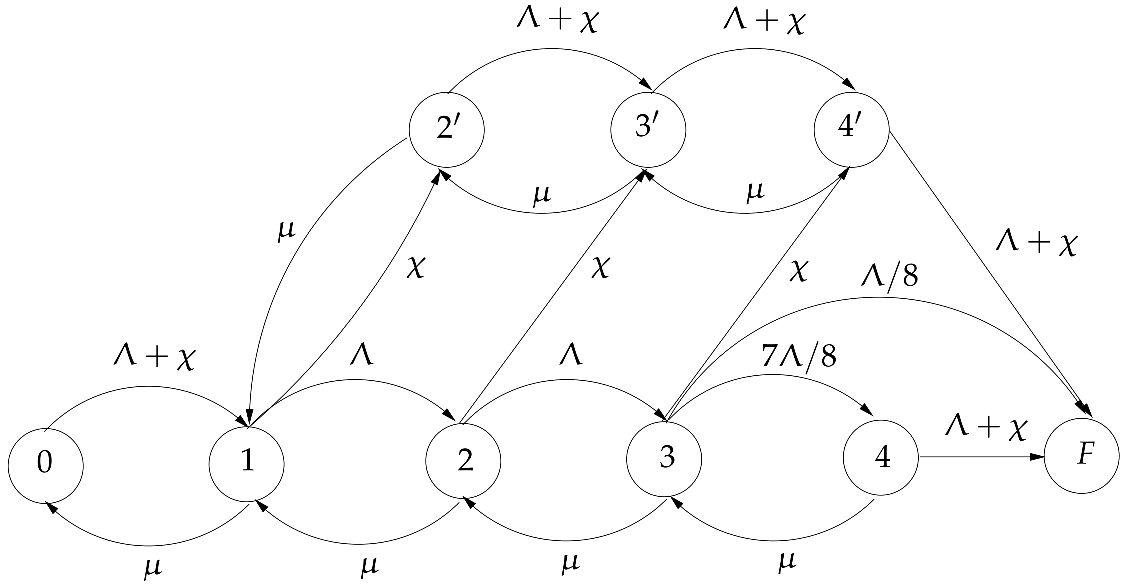


Figure 3.9: State diagram description of RC-coded arrays under Random and Clustered failures. A new failure process with rate χ introduces Clustered failures.

2, 3 and 4 Clustered (not all-isolated) failures, respectively. The transitions marked with χ represent moving from all-isolated failures to a Clustered failure combination. At the upper branch, both Random and additional Clustered failures result in Clustered failure combinations – and that accounts for the transitions marked $\Lambda + \chi$. From state 0 a Clustered failure is not well defined, but the rate χ is added to the transition to maintain consistency with respect to the total failure rate ($\Lambda + \chi$) outgoing from all other states.

Solving the 8×8 linear system for the diagram in Figure 3.9, the $MTTDL$ can be calculated in closed form for all combinations of χ, Λ, μ. This ability to have a closed form expression for the $MTTDL$ of RC codes, under both Random and Clustered failures, is crucial for a system operator to predict the reliability of the storage array under more realistic failure assumptions. The resulting $MTTDL$ curves for RC codes under three different χ values are plotted in Figure 3.10, and compared to the $MTTDL$ of a 4-Random failure code under the same conditions (4-Random codes give the same $MTTDL$ independent of the ratio between χ and Λ, as long as their sum is fixed). Not surprisingly, the curves of Figure 3.10 prove that as Clustered failures become more dominant, the reliability of RC codes is approaching the reliability of a 4-Random failure-correcting code.

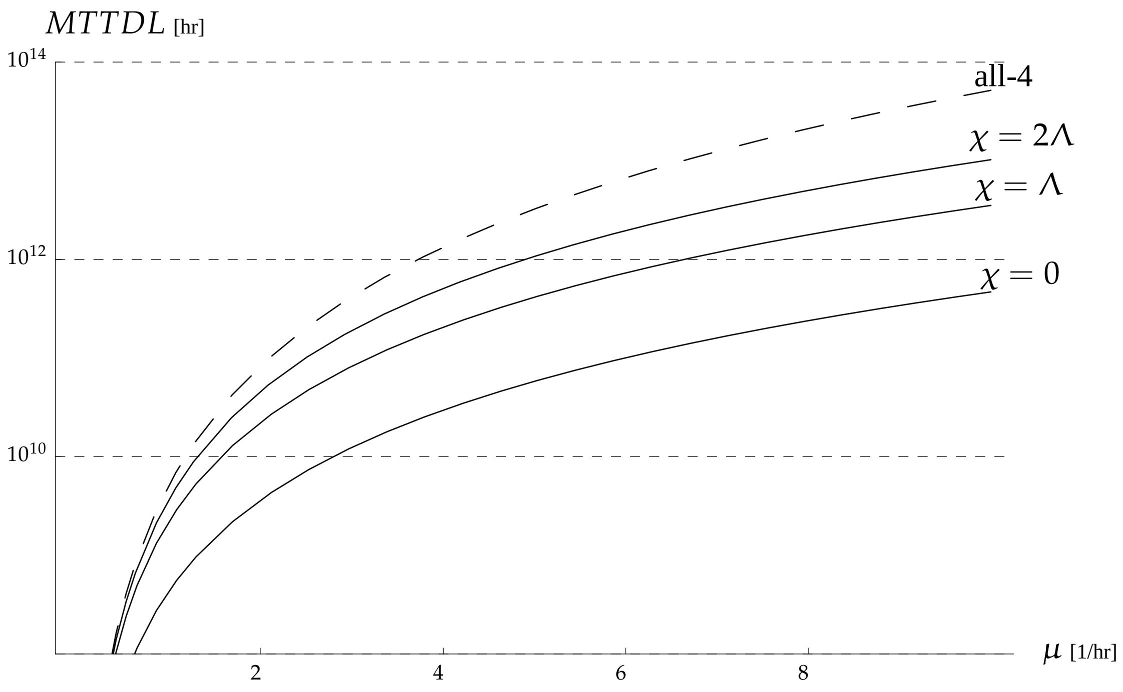


Figure 3.10: $MTTDL$ curves under Random and Clustered failures for RC codes and all-4-erasure correcting code. For three values of χ, the $MTTDL$ of RC codes is shown by the solid curves. The $MTTDL$ of an all-4-erasure correcting code is the same for all values of χ.

3.8 Code Evaluation and Comparison with Existing Schemes

We compare RC codes to EVENODD ($r = 4$) codes using various performance criteria. The failure-correction properties in Table 3.1 apply for any prime p such that 2 is primitive in GF(p).

The redundancy r is 4 for both codes. RC codes can support up to $2p$ information columns while EVENODD can only have up to p. Since parity columns R_0 and R'_1 each depends on half of the information columns, the encoding complexity of RC codes is $3kp$, compared to $4kp$ in EVENODD. In both cases, when k is of the same order of p, the decoding complexity is dominated by syndrome calculations (for RC codes this has been shown in Section 3.6). Therefore, similarly to the encoding case, RC codes need about $3kp$

	RC Codes	4-EVENODD
Code Length (up to)	$2p$	p
Redundancy	4	4
Encoding Complexity	$3kp$	$4kp$
Decoding Complexity	$3kp$	$4kp$
Update Complexity	5	7
Clustered Failures	$\sim$ All	All
Random Failures	$7/8$	All

Table 3.1: Comparison of RC Codes and EVENODD Codes

bit operations to decode, compared to $4kp$ for EVENODD. As for the update-complexity, RC codes are significantly more efficient. Their small-write update complexity is 5. Each of the $2p(p-1)$ updated information bits needs 3 parity updates, P, Q, R_0 for bits in even columns and P, Q, R_1' for bits in odd columns. The $4(p-1)$ bits that belong to EO diagonals ($2(p-1)$ in Q and $p-1$ in each of R_0, R_1') require additional $p-1$ parity-bit updates each for adjusting even/odd parities. The small-write update-complexity of RC is then obtained by averaging

$$\frac{6p(p-1)+4(p-1)^2}{2p(p-1)} = 5 - o(1)$$

Recall that EVENODD has small-write update-complexity of $2r-1-o(1) = 7-o(1)$. The full-column update-complexity of RC is 3 while EVENODD's is 4. Thus RC offers a 28.57% improvement in the average number of small-writes and 25% improvement in the number of full-column updates. The fraction of Clustered erasures correctable by RC codes is $1-o(1)$, essentially the same as EVENODD's 1 fraction. Only in Random erasure-correction capability are RC codes slightly inferior to EVENODD codes, the fraction of correctable Random erasures is $7/8 - o(1)$ compared to 1 for EVENODD.

3.9 Discussion

The key idea in the construction of the family of RC codes, is to find a "good" "cooperating interleaving" of two codes. By "cooperating interleaving" we mean that some of the code parity bits are computed from only one constituent code, but other parity bits are computed from both codes. By “good” we mean that all 4-erasure combinations, except those that fall exclusively on one constituent code, will be correctable by the code. For the particular case addressed by RC codes, the challenge was to simultaneously correct both combinations of (3 even/odd,1 odd/even) column failures *and* combinations of (2 even,2 odd) column failures. Both are needed to cover all cases of Clustered failures. In that respect, Pyramid codes [HCL07] use “cooperating interleaving” in their construction. Nevertheless, these interleavings are not “good” in the sense that there are many more uncorrectable erasures beyond what allowed by the definition of “good” above.

A central contribution of this chapter is the classification of column sets by the number of clusters they occupy, and the use of that classification to analyze the correctability of Clustered 4-erasures. Admittedly, that classification is much more general than its context here. From a coding theoretic perspective, a rich variety of error models can be defined based on that abstract classification. It is an interesting open problem, one with great practical promise, whether a general theory of Clustered error correction can be found, that includes both general code constructions and tight upper bounds.

Chapter 4

Cyclic Lowest-Density MDS Array Codes

It is when practical motivations meet mathematical beauty that a research area becomes attractive and vibrant. Many areas of Coding Theory have flourished thanks to their intriguing links to frontiers of deep mathematics. When questions about code properties translate to the most fundamental combinatorial or algebraic problems, the code designer is humbled by the increased load that is adjoined to his attempts. In the area of array codes, the problem with the clearest reflections in mathematics, and with simultaneously a great practical appeal, is the construction of *lowest-density* MDS codes. This chapter adds new results, to the handful of previously known ones, by constructing codes that are lowest-density, MDS, and also *cyclic*, thus offering better codes in the practical sense and improved understanding of the underlying combinatorial entities. The main contributions of the chapter are summarized below.

- Definition of a new class of array codes: *systematically-cyclic* array codes.
- Construction of three new families of lowest-density, systematically-cyclic, MDS array codes.
- Description of the complexity benefits systematically-cyclic codes offer to practical storage systems.

The results of this chapter appear in [CB06] and [CB07].

4.1 Introduction

MDS (Maximum Distance Separable) codes over large symbol alphabets are ubiquitous in data storage applications. Being MDS, they offer the maximum protection against device failures for a given amount of redundancy. Array codes, as mentioned in the previous chapter, are one type of such codes that are very useful in dynamic high-speed storage systems, as they enjoy low-complexity decoding algorithms, as well as low update complexity when small changes are applied to the stored content. That is in contrast to the family of Reed-Solomon codes [MS77, Ch.10] that in general has none of these favorable properties.

A particular array-code sub-class of interest is *lowest density* array codes, those that have the smallest possible update complexity for their parameters. Since the update complexity dictates the access time to the storage array, even in the absence of failures, this parameter of the code is the primary limiting factor of the code implementation in enterprise storage systems. Examples of constructions that yield lowest-density array codes can be found in [ZZS81],[XBBW99],[XB99],[LR06],[BR99]. In this chapter we propose lowest-density codes that are also *cyclic* or *quasi-cyclic*. Adding regularity in the form of cyclic symmetry to lowest-density MDS array codes makes their implementation simpler and potentially less costly. The benefit of the cyclic symmetry becomes especially significant when the code is implemented in a distributed way on distinct network nodes. In that case, the use of cyclic codes allows a uniform design of the storage nodes and the interfaces between nodes. The code constructions additionally offer a theoretical value by unveiling more of the rich structure of lowest-density MDS array codes.

As an example, we examine the following code defined on a 2×6 array. The $+$ signs represent the binary Exclusive-OR operation. This code has 6 information bits $a_0, \ldots, a_5$,

a_0	a_1	a_2	a_3	a_4	a_5
$a_2+a_3+a_4$	$a_3+a_4+a_5$	$a_4+a_5+a_0$	$a_5+a_0+a_1$	$a_0+a_1+a_2$	$a_1+a_2+a_3$

and 6 parity bits $a_2 + a_3 + a_4,\ a_3 + a_4 + a_5,\ a_4 + a_5 + a_0,\ a_5 + a_0 + a_1,\ a_0 + a_1 + a_2,\ a_1 + a_2 + a_3$. It is easy to see that all 6 information bits can be recovered from *any* set of 3 columns. For example, if we want to recover a_3, a_4, a_5 from the bits of the 3 left columns, we can proceed by $a_3 = (a_3 + a_4 + a_5) + (a_4 + a_5 + a_0) + a_0$, then $a_4 = a_2 + (a_2 + a_3 + a_4) + a_3$, and finally $a_5 = (a_3 + a_4 + a_5) + a_3 + a_4$. Since 3 columns have 6 bits in total, the code is Maximum Distance Separable (MDS). Additionally, the code has lowest-density, since updating an information bit requires 3 parity updates – a trivial lower bound for a code that recovers from any 3 erasures. However, the focus of this chapter is a different property of this sample code: its cyclicity. To convince oneself that the code is cyclic, we observe that all the indices in a column can be obtained by adding one (modulo 6) to the indices in the column to its (cyclic) left. Thus any shift of the information bits row results in an identical shift in the parity bits row (and hence the code is closed under cyclic shifts of its columns).

The sample code above, as well as all the codes constructed in the chapter, belong to a sub-class of cyclic array codes: *systematically-cyclic array codes*. Section 4.3 contains characterizations of cyclic array codes in general and systematically-cyclic codes (first defined here), in particular. Codes in the systematically-cyclic sub-class enjoy greater implementation benefits relative to the general class of cyclic codes. Properties of cyclic and systematically-cyclic array codes that imply simpler implementation are provided in section 4.6. In particular, these properties manifest simpler updates and encoding, and more efficient erasure and error decoding.

	array dimensions	r	notes
$\kappa_{1\circ}$	$(p-1)/2 \times (p-1)$	2	
$\kappa_{2\circ}$	$(p-1)/r \times (p-1)$	3,4	2 primitive in GF(p)
$\kappa_{3\circ}$	$(p-1) \times 2(p-1)$	2	2-quasi-cyclic

Table 4.1: Summary of cyclic code constructions

In sections 4.4 and 4.5, three families of lowest-density, systematically-cyclic (or systematically-quasi-cyclic) MDS array codes are constructed. The families are named $\kappa_{1\circ}$, $\kappa_{2\circ}$ and $\kappa_{3\circ}$, respectively (the $\circ$ qualifier designates a cyclic or quasi-cyclic code), and their properties are summarized in Table 4.1 above. For all primes p, $\kappa_{1\circ}$ provides systematically-cyclic codes on arrays with dimensions $(p-1)/2 \times (p-1)$ and redundancy $r = 2$, over any Abelian group. For all primes p, such that $r|p-1$ and 2 is primitive in GF(p), $\kappa_{2\circ}$, which is a generalization of $\kappa_{1\circ}$, provides systematically-cyclic codes on arrays with dimensions

$(p-1)/r \times (p-1)$ and redundancy $r = 3, 4$, over fields of characteristic 2. $\kappa_{2\circ}$ is the first known family of cyclic lowest density MDS array codes with $r > 2$. Finally, for all primes p, $\kappa_{3\circ}$ provides systematically-quasi-cyclic codes on arrays with dimensions $(p-1) \times 2(p-1)$, over any Abelian group. A specific instance of the family $\kappa_{i\circ}$ is denoted $\kappa_{i\circ}(p)$, for some prime p. Cyclic codes with the same parameters as $\kappa_{1\circ}$ were proposed in [ZZS81], but these are not systematically-cyclic and therefore enjoy only part of the properties $\kappa_{1\circ}$ have. Non-cyclic codes with the same parameters as $\kappa_{2\circ}$ are given in [LR06]. In addition, the existence of codes with the same parameters as $\kappa_{1\circ}$ and $\kappa_{3\circ}$ was shown in [XBBW99]. However, using the suggested combinatorial construction tools of [XBBW99] gives non-cyclic codes.

The construction technique we use is first constructing non-cyclic lowest-density MDS codes, and then explicitly providing a transformation to their parity check matrices that results in new, non-equivalent, cyclic codes with the same minimum distance and density. For easier reading, a construction of a sample code precedes the general construction method in section 4.4 while the construction of section 4.5 works an example after each step.

4.2 Definitions

A *linear array code* $\mathcal{C}$ of dimensions $b \times n$ over a field $F = \mathrm{GF}(q)$ is a linear subspace of the vector space F^{nb}. The dual code $\mathcal{C}^{\perp}$ is the null-space of $\mathcal{C}$ over F. To define the minimum distance of an array code we regard it as a code over the alphabet F^b, where F^b denotes length b vectors over F. Then the minimum distance is simply the minimum Hamming distance of the length n code over F^b. Note that though the code symbols can be regarded as elements in the finite field $\mathrm{GF}(q^b)$, we do not assume linearity over this field.

$\mathcal{C}$ can be specified by either its parity-check matrix H of size $N_p \times nb$ or its generator matrix G of size $(nb - N_p) \times nb$, both over F. An array S of size $b \times n$ is a codeword of $\mathcal{C}$ if the length nb column vector $\boldsymbol{\sigma}$, obtained by taking the bits of S column after column, satisfies $H\boldsymbol{\sigma} = \mathbf{0}$, where $\mathbf{0}$ is the length N_p all-zero column vector. From practical considerations, array codes are required to be *systematic*, namely to have a parity-check (or generator) matrix that is systematic, as now defined.

Definition 4.1 *A parity-check (or generator) matrix is called [weakly]* ***systematic*** *if it has N_p (or $nb - N_p$), not necessarily adjacent, columns that when stacked together form the identity matrix of order N_p (or $nb - N_p$), respectively.*

Given a systematic H matrix or G matrix (one can be easily obtained from the other), the nb symbols of the $b \times n$ array can be partitioned into N_p parity symbols and $nb - N_p$ information symbols. Define the *density* of the code as the average number of non-zeros in a row of G: $\frac{N(G)}{nb-N_p}$, where $N(M)$ is the number of non-zeros in a matrix M. When H is systematic, an alternative expression for the density is $1 + \frac{N(H)-N_p}{nb-N_p}$. The codes proposed in this chapter, all have the lowest possible density, as defined below.

Definition 4.2 *A code $\mathcal{C}$ is called* ***lowest density*** *if its density equals its minimum distance.*

(the minimum distance is an obvious lower bound on the density [BR99]). If $b|N_p$ and the minimum distance d equals $\frac{N_p}{b} + 1$, then the code is called Maximum Distance Separable (MDS) with redundancy $r = \frac{N_p}{b}$.

Throughout the chapter $[s, t]$ denotes the set $\{x \in \mathbb{Z} : \; s \leq x \leq t\}$. To simplify the presentation of the constructions in the chapter, we introduce another structure that defines a code when, as is the situation here, the parity check matrix has elements in $\{0, 1\}$.

Definition 4.3 *Given a parity check matrix H of a code $\mathcal{C}$, define the* ***index array*** $\mathsf{A}_{\mathcal{C}}$ *to be a $b \times n$ array of subsets of $[0, N_p - 1]$. The set in location i, j of $\mathsf{A}_{\mathcal{C}}$ contains the elements $\{x : \; h_{i+bj}(x) = 1\}$, where h_l denotes the l^{th} column of H and $h_l(x)$ denotes the x^{th} element of h_l, $x \in [0, N_p - 1]$*

Each set in $\mathsf{A}_{\mathcal{C}}$ represents a column of H. If H is systematic, $\mathsf{A}_{\mathcal{C}}$ has N_p sets of size 1, called singletons. Note that $\mathsf{A}_{\mathcal{C}}$ has the same dimensions as the code array. As an example we take a ($n = 6, b = 3, N_p = 6$) systematic code and provide in Figure 4.1 a generator matrix G and a parity check matrix H with its index array $\mathsf{A}_{\mathcal{C}}$.

$$G = \left[\begin{array}{ccc|ccc|ccc|ccc|ccc|ccc}
0&1&0&0&0&0&0&0&0&0&0&0&1&0&0&1&0&0\\
0&0&1&1&0&0&0&0&0&1&0&0&0&0&0&0&0&0\\
1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&1&0&0\\
0&0&0&0&0&1&1&0&0&0&0&0&1&0&0&0&0&0\\
1&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&1&0&0&0&0&0&1&0&0\\
0&0&0&1&0&0&1&0&0&0&1&0&0&0&0&0&0&0\\
1&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&0&0\\
0&0&0&0&0&0&1&0&0&1&0&0&0&1&0&0&0&0\\
0&0&0&1&0&0&0&0&0&0&0&0&0&0&1&1&0&0\\
0&0&0&0&0&0&0&0&0&1&0&0&1&0&0&0&1&0\\
1&0&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&1
\end{array}\right] \updownarrow nb - N_p$$

$$\xleftrightarrow{\qquad\qquad nb \qquad\qquad}$$

$$H = \left[\begin{array}{ccc|ccc|ccc|ccc|ccc|ccc}
1&0&0&0&1&0&0&1&0&0&0&1&0&0&0&0&0&1\\
0&0&1&1&0&0&0&1&0&0&1&0&0&0&1&0&0&0\\
0&0&0&0&0&1&1&0&0&0&1&0&0&1&0&0&0&1\\
0&0&1&0&0&0&0&0&1&1&0&0&0&1&0&0&1&0\\
0&1&0&0&0&1&0&0&0&0&0&1&1&0&0&0&1&0\\
0&1&0&0&1&0&0&0&1&0&0&0&0&0&1&1&0&0
\end{array}\right] \updownarrow N_p$$

$$\mathsf{A}_{\mathcal{C}} = \begin{array}{|c|c|c|c|c|c|}
\hline
0 & 1 & 2 & 3 & 4 & 5\\
\hline
4,5 & 5,0 & 0,1 & 1,2 & 2,3 & 3,4\\
\hline
1,3 & 2,4 & 3,5 & 4,0 & 5,1 & 0,2\\
\hline
\end{array} \updownarrow b$$

$$\xleftrightarrow{\qquad\qquad n \qquad\qquad}$$

Figure 4.1: G,H and the index array $\mathsf{A}_{\mathcal{C}}$ for a sample $n = 6, b = 3, N_p = 6$ code $\mathcal{C}$. Each set of $\mathsf{A}_{\mathcal{C}}$ specifies the locations of the ones in a single column of H.

4.3 Cyclic Array Codes

The codes constructed in this chapter are codes of length n over F^b which are cyclic but *not* linear (though they *are* usually linear over F). In this section we wish to discuss such codes in general, providing conditions for a code to be cyclic. One way to characterize cyclic array codes is as cyclic group codes over the direct-product group of the additive group of F. Another is to view them as length nb linear *b-quasi-cyclic* codes. For the most part, the latter view will prove more useful since the constructions in the chapter are not explicit group theoretic ones. In fact, the description of array codes using index arrays we chose here was used in [TW67] to describe quasi-cyclic code constructions. We start off with the basic definition of cyclic codes.

Definition 4.4 *The code $\mathcal{C}$ over F^b is **cyclic** if*

$$s = (s_0, s_1, \ldots, s_{n-2}, s_{n-1}) \in \mathcal{C}$$
$$\Rightarrow s' = (s_1, s_2, \ldots, s_{n-1}, s_0) \in \mathcal{C}$$

and $s_i \in F^b$.

Cyclic codes over F^b are related to quasi-cyclic codes over F in the following manner.

Proposition 4.1 *An array code $\mathcal{C}$ of length n over F^b is cyclic if and only if the code $\mathcal{C}_{1D}$ of length nb over F, that has the same parity check matrix, is quasi-cyclic with basic block length b.*

This equivalence allows us to use the characterization of quasi-cyclic codes from [PW72, pp.257], to determine the cyclicity of an array code.

Theorem 4.2 *A code $\mathcal{C}$ on $b \times n$ arrays and $N_p = \rho n$, ρ an integer, is cyclic if it has a parity check matrix of the form*

$$H = \left[\begin{array}{c|c|c|c} \mathbf{Q}_0 & \mathbf{Q}_1 & \cdots & \mathbf{Q}_{n-1} \\ \mathbf{Q}_{n-1} & \mathbf{Q}_0 & \cdots & \mathbf{Q}_{n-2} \\ \vdots & \vdots & & \vdots \\ \mathbf{Q}_1 & \mathbf{Q}_2 & \cdots & \mathbf{Q}_0 \end{array}\right]$$

where $\mathbf{Q}_i$ are arbitrary matrices of size $\rho \times b$.

Note that if H is *not* required to have full rank of ρn, then Theorem 4.2 captures the most general cyclic array codes (the *if* statement can be replaced with an *if and only if* one.). However, there exist cyclic array codes that do not have full rank matrices H, of the form given above. For example, $H = \left[\begin{array}{cc|cc} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{array}\right]$ has the following words as codewords

$$\left\{ \begin{array}{|c|c|} \hline 0 & 0 \\ \hline 0 & 0 \\ \hline \end{array}, \begin{array}{|c|c|} \hline 1 & 1 \\ \hline 0 & 0 \\ \hline \end{array}, \begin{array}{|c|c|} \hline 0 & 0 \\ \hline 1 & 1 \\ \hline \end{array}, \begin{array}{|c|c|} \hline 1 & 1 \\ \hline 1 & 1 \\ \hline \end{array} \right\}$$

and hence it is cyclic. However, there is no 2×4 parity check matrix for this code that admits the structure of $\left[\begin{array}{c|c} \mathbf{Q}_0 & \mathbf{Q}_1 \\ \mathbf{Q}_1 & \mathbf{Q}_0 \end{array}\right]$.

A sub-class of the cyclic codes characterized above, *systematically-cyclic array codes*, is next defined. These are cyclic array codes in which each column has ρ parity symbols, at the same locations for all columns.

Definition 4.5 *A code $\mathcal{C}$ on $b \times n$ arrays and $N_p = \rho n$, ρ an integer, is* ***systematically-cyclic*** *if it has a parity check matrix of the form*

$$H = \left[\begin{array}{c|c|c|c} \mathbf{IP}_0 & \mathbf{OP}_1 & \dots & \mathbf{OP}_{n-1} \\ \mathbf{OP}_{n-1} & \mathbf{IP}_0 & \dots & \mathbf{OP}_{n-2} \\ \vdots & \vdots & & \vdots \\ \mathbf{OP}_1 & \mathbf{OP}_2 & \dots & \mathbf{IP}_0 \end{array}\right]$$

where **I** *and* **O** *represent, respectively, the identity and all-zero matrices of order ρ.* $\mathbf{P}_i$ *are arbitrary matrices of size $\rho \times (b - \rho)$.*

An equivalent characterization can be obtained using the index array $\mathsf{A}_{\mathcal{C}}$ of the code $\mathcal{C}$. Corollary 4.3 to Theorem 4.2 and Definition 4.6 provide this characterization.

Corollary 4.3 *A code $\mathcal{C}$ on $b \times n$ arrays and $N_p = \rho n$, ρ an integer, is cyclic if it has an index array representation $\mathsf{A}_{\mathcal{C}}$, in which adding ρ to all set elements modulo ρn results in a cyclic shift of $\mathsf{A}_{\mathcal{C}}$.*

Definition 4.6 *A code $\mathcal{C}$ on $b \times n$ arrays and $N_p = \rho n$, ρ an integer, is systematically-cyclic if it has an index array representation $\mathsf{A}_{\mathcal{C}}$, in which N_p of the sets are singletons and adding ρ to all set elements modulo ρn results in a cyclic shift of $\mathsf{A}_{\mathcal{C}}$.*

4.4 $\kappa_{1\circ}, \kappa_{2\circ}$: Cyclic Lowest-Density MDS codes with $n = p - 1, b = \frac{p-1}{r}$

The constructions of the code families in this chapter specify the index arrays of codes with dimensions parametrized by a prime p. For two of the code families – $\kappa_{1\circ}, \kappa_{2\circ}$, the construction uses abstract properties of Finite Fields to obtain index-array sets that guarantee cyclic lowest-density MDS codes for all code dimensions. To better understand the construction method of $\kappa_{1\circ}, \kappa_{2\circ}$, the general construction is preceded by the construction of one particular instance of the family: $\kappa_{1\circ}(7)$.

$\kappa_{1\circ}(7)$ is a cyclic MDS array code with dimensions $b = 3, n = 6$ and redundancy $r = 2$. In the finite field with 7 elements GF(7),[1] pick $\alpha = 6$, an element of multiplicative order $r = 2$. Pick $\beta = 3$, an element with multiplicative order $p - 1 = 6$. Using α and β, GF(7) is partitioned into the following sets C_i.

$$C_{-1} = \{0\} \quad , \quad C_0 = \{\beta^0, \beta^0\alpha\} = \{1, 6\},$$
$$C_1 = \{\beta^1, \beta^1\alpha\} = \{3, 4\} \quad , \quad C_2 = \{\beta^2, \beta^2\alpha\} = \{2, 5\}$$

The elements of the sets C_{-1}, C_1, C_2 (C_0 is discarded since it contains the element $p - 1 = 6$) are permuted by the permutation $[0, 1, 2, 3, 4, 5] \xrightarrow{\bar{\psi}} [0, 2, 1, 4, 5, 3]$ and the corresponding sets D_j now follow.

$$D_0 = \bar{\psi}(C_{-1}) = \{0\} \quad ,$$
$$D_1 = \bar{\psi}(C_1) = \{4, 5\} \quad , \quad D_2 = \bar{\psi}(C_2) = \{1, 3\}$$

The sets D_0, D_1, D_2 define the first column of the index array of $\kappa_{1\circ}(7)$. Each of the other 5 columns is obtained by adding 1 modulo 6 to the elements of the sets in the column to its

[1]GF(p) used for the code construction should not be confused with F, the code alphabet

left. The final index array of the code $\kappa_{1\circ}(7)$ is now given.

$$\mathsf{A}_{\kappa_{1\circ}(7)} = \begin{array}{|c|c|c|c|c|c|} \hline 0 & 1 & 2 & 3 & 4 & 5 \\ \hline 4,5 & 5,0 & 0,1 & 1,2 & 2,3 & 3,4 \\ \hline 1,3 & 2,4 & 3,5 & 4,0 & 5,1 & 0,2 \\ \hline \end{array} .$$

It is left as an exercise to verify that $\kappa_{1\circ}(7)$ is cyclic, lowest-density and MDS.

We now provide the general construction of the code families $\kappa_{1\circ}, \kappa_{2\circ}$.
Let r be a divisor of $p-1$, and p an odd prime. Let α be an element in GF(p) of order r and β be an element in GF(p) of order $p-1$. The order of an element x in GF(p) is defined as the smallest non-zero integer i such that $x^i = 1 \pmod{p}$. α and β define a partition of GF(p) to $\frac{p-1}{r}+1$ sets. These sets are the $\frac{p-1}{r}$ cosets of the multiplicative subgroup of order r of GF(p), plus a set that contains only the zero element. Except for the zero set, all sets are of cardinality r.

$$C_{-1} = \{0\} \qquad C_i = \{\beta^i, \beta^i\alpha, \ldots, \beta^i\alpha^{r-1}\} \tag{4.1}$$

where $0 \le i < \frac{p-1}{r}$. The sets C_i are used in [LR06] to construct (non-cyclic) lowest density MDS codes with redundancy $r = 3, 4$. The same construction, only with $r = 2$, provides (non-cyclic) lowest density MDS codes by applying the perfect 1-factorization of complete graphs with $p+1$ vertices by Anderson [And73], to the construction of [XBBW99]. Shortened versions of the non-cyclic constructions of [XBBW99] and [LR06] are used in the proofs of the constructions of this chapter, and are denoted κ_1 and κ_2, respectively. As shown by [LR06], κ_2 provides lowest density MDS codes for a wide range of parameters. When F has characteristic 2, MDS codes are obtained for $r = 3$ and $r = 4$, whenever 2 is primitive in GF(p). For larger characteristics, codes with additional r values were shown to be MDS. For $r = 2$, κ_1 provides MDS codes over any Abelian group [XBBW99].

Since $\kappa_{1\circ}, \kappa_{2\circ}$ follow the same construction (only with different r), in the forthcoming discussion we treat them as one family (denoted $\kappa_{1\circ,2\circ}$). Following the presentation of the $\kappa_{1\circ,2\circ}$ construction, we explicitly present the construction for the non-cyclic MDS codes $\kappa_{1,2}$. This is done for the benefit of proving the MDS property of $\kappa_{1\circ,2\circ}$ - through a minimum-distance preserving transformation from the parity-check matrix of $\kappa_{1,2}$ to that of $\kappa_{1^\circ,2^\circ}$.

Better readability in mind and with a slight abuse of notation, operations on sets denote element-wise operations on the elements of the sets. Specifically, if $\langle x + l\rangle_z$ is used to denote $x + l \pmod z$, then $\langle S + l\rangle_z$ denotes the set that is obtained by adding l to the elements of S modulo z. Similarly, permutations and arithmetic operations on sets represent the corresponding operations on their elements.

We now turn to show how the sets C_i of equation (4.1) are used to construct the cyclic lowest-density MDS codes $\kappa_{1^\circ}, \kappa_{2^\circ}$. Define $I_0 = \{i : \forall x \in C_i\ ,\ 0 \le x < p-1\}$. I_0 is the set of all indices i, except for the unique index i' for which $C_{i'}$ contains the element $p-1$. Clearly $|I_0| = \frac{p-1}{r}$. Denote the j^{th} element of I_0 by $I_0(j)$, $j \in [0, \frac{p-1}{r} - 1]$, where indices in I_0 are ordered lexicographically. The permutation $\psi : [0, p-2] \to [0, p-2]$ is defined to be $\psi(x) = \beta^x - 1 \pmod p$. We also define the inverse of ψ, $\bar{\psi}(y) = \log_\beta(y+1)$. The constructing sets D_j are now defined using C_i and the permutation ψ.

$$D_j = \bar{\psi}(C_{I_0(j)}),\ \text{for } j \in [0, \frac{p-1}{r} - 1].$$

The construction of $\kappa_{1^\circ,2^\circ}$ is now provided by specification of the index array $\mathsf{A}_{\kappa_{1^\circ,2^\circ}}$.

In $\mathsf{A}_{\kappa_{1^\circ,2^\circ}}$, the set at location

$$(j, l) \in [0, \frac{p-1}{r} - 1] \times [0, p-2]$$

is

$$\langle D_j + l\rangle_{p-1}.$$

The codes $\kappa_{1^\circ,2^\circ}$ are systematically-cyclic by Definition 4.6 since the top row ($j = 0$) contains sets of size 1, and for every l, translations of the same sets D_j are taken.

As for the codes $\kappa_{1,2}$, for every $0 \le m < p-1$ define $I_m = \{i : \forall x \in \langle C_i + m\rangle_p\ ,\ 0 \le x < p-1\}$ (I_m is the set of all indices i, except for the unique index i' for which $\langle C_i + m\rangle_p$ contains the element $p-1$). It is obvious that for every m, $|I_m| = \frac{p-1}{r}$ since for every translation m of the sets C_i, only one set contains the element $p-1$. Denote the j^{th} element

of I_m by $I_m(j)$, $j \in [0, \frac{p-1}{r} - 1]$, where indices in I_m are ordered lexicographically. The code $\kappa_{1,2}$ is defined via an index array $\mathsf{A}_{\kappa_{1,2}}$.
In $\mathsf{A}_{\kappa_{1,2}}$, the set at location

$$(j, m) \in [0, \frac{p-1}{r} - 1] \times [0, p-2]$$

is

$$\langle C_i + m \rangle_p \,, \quad i = I_m(j).$$

Note that because of the restriction $i \in I_m$, $\kappa_{1,2}$ provides *non*-cyclic codes.

The known MDS property of $\kappa_{1,2}$ is next used to prove the MDS property of $\kappa_{1_\circ,2_\circ}$.

Theorem 4.4 *$\kappa_{1_\circ,2_\circ}$ and $\kappa_{1,2}$ have the same redundancy, minimum distance and density.*

Proof: We explicitly show an invertible transformation from $\mathsf{A}_{\kappa_{1_\circ,2_\circ}}$ to $\mathsf{A}_{\kappa_{1,2}}$ that preserves the code redundancy, density, and minimum distance. To refer to an element x in the set at location (j, l) in an index array $\mathsf{A}_{\mathcal{C}}$, we use the tuple $(x, j, l, \mathcal{C})$. The aforementioned transformation is given by showing that $\mathsf{A}_{\kappa_{1,2}}$ is obtained from $\mathsf{A}_{\kappa_{1_\circ,2_\circ}}$ by a mapping $(x, j, l, \kappa_{1_\circ,2_\circ}) \leftrightarrow (\psi(x), j', m, \kappa_{1,2})$. The mapping $x \leftrightarrow \psi(x)$ represents permuting the rows of the parity check matrix and the mapping $(j, l) \leftrightarrow (j', m)$ represents permuting columns of the parity check matrix (which for array codes, in general, does not preserve the minimum distance). As will soon be proved, the mapping $(j, l) \leftrightarrow (j', m)$ has a special property that it only reorders columns of the index array and reorders sets *within* its columns (m is a function of l, independent of j, and j' is a function of both j, l.). Hence, all operations preserve the redundancy of the code, its minimum distance and its density. More concretely, we need to show that for every $l \in [0, p-2]$ there exists an $m \in [0, p-2]$ such that every j has a corresponding $t = I_m(j')$ that together satisfy

$$\psi[\langle D_j + l \rangle_{p-1}] = \langle C_t + m \rangle_p$$

Since $\langle D_0 + l \rangle_{p-1}$ consists of the single element l and $\langle C_{-1} + m \rangle_p$ consists of the single element m, the integers l and m have to satisfy $m = \psi(l)$. Then, for the remainder of the sets ($j > 0$), we rewrite the above condition as

$$\psi[\langle D_j + l\rangle_{p-1}] = \langle C_t + \psi(l)\rangle_p$$

Define $i = I_0(j)$, we can now prove the above statement

$$\psi[\langle D_j + l\rangle_{p-1}] = \psi[\langle \bar{\psi}[C_i] + l\rangle_{p-1}] =$$

$$\langle \beta^{\log_\beta(\langle C_i+1\rangle_{p-1})+l} - 1\rangle_p = \langle \beta^l C_i + \beta^l - 1\rangle_p =$$

$$\langle C_{\langle i+l\rangle_{\frac{p-1}{r}}} + \psi(l)\rangle_p$$

and the required transformation is
$(x, j, l, \kappa_{1\circ,2\circ}) \leftrightarrow (\psi(x), j', \psi(l), \kappa_{1,2})$, where j' satisfies $I_{\psi(l)}(j') = \langle I_0(j) + l\rangle_{(p-1)/r}$ for $j > 0$, and $j' = j = 0$ for $j = 0$.

$\square$

4.4.1 Example: $\kappa_{1\circ}(7)$ revisited – the transformation from $\kappa_1(7)$

To construct $\kappa_1(7)$, the sets

$$C_{-1} = \{0\},\ C_0 = \{1,6\},\ C_1 = \{3,4\},\ C_2 = \{2,5\}$$

are used by taking the sets $\langle C_i + m\rangle_7$ to be the sets of $\mathsf{A}_{\kappa_1(7)}$ in column m, leaving out the particular set in that column that contains the element 6.

$\mathsf{A}_{\kappa_1(7)} =$

0	1	2	3	4	5
$3,4$	$2,0$	$3,1$	$4,2$	$5,3$	$1,2$
$2,5$	$4,5$	$4,0$	$5,1$	$0,1$	$0,3$

The permutations ψ and $\bar{\psi}$ written explicitly are $[0,1,2,3,4,5] \xrightarrow{\psi} [0,2,1,5,3,4]$ and $[0,1,2,3,4,5] \xrightarrow{\bar{\psi}} [0,2,1,4,5,3]$. $\bar{\psi}$ acting on the array $\mathsf{A}_{\kappa_1(7)}$ yields

$\bar{\psi}(\mathsf{A}_{\kappa_1(7)}) =$

0	2	1	4	5	3
4,5	1,0	4,2	5,1	3,4	2,1
1,3	5,3	5,0	3,2	0,2	0,4

which after reordering of columns and sets within columns results in the systematically-cyclic code $\kappa_{1\circ}(7)$.

$\mathsf{A}_{\kappa_{1\circ}(7)} =$

0	1	2	3	4	5
4,5	5,0	0,1	1,2	2,3	3,4
1,3	2,4	3,5	4,0	5,1	0,2

4.5 $\kappa_{3\circ}$: Quasi-Cyclic Lowest-Density MDS Codes with $n = 2(p-1), b = p-1, r = 2$

Before constructing the 2-quasi-cyclic code $\kappa_{3\circ}$, we discuss quasi-cyclic array codes in general. The definitions and characterizations provided for cyclic array codes in section 4.3 can be generalized to quasi-cyclic array codes.

Definition 4.7 *The code $\mathcal{C}$ over F^b is **T-quasi-cyclic** if*

$$s = (s_0, s_1, \ldots, s_{n-2}, s_{n-1}) \in \mathcal{C}$$
$$\Rightarrow s' = (s_T, s_{T+1}, \ldots, s_{n-1}, s_0, \ldots, s_{T-1}) \in \mathcal{C}$$

and $s_i \in F^b$.

A generalization of Theorem 4.2 to quasi-cyclic array codes is now provided.

Theorem 4.5 *A code $\mathcal{C}$ on $b \times n$ arrays and $N_p = \rho n$, ρ an integer, is T-quasi-cyclic*

($n = \lambda T$) if it has a parity check matrix of the form

$$H = \begin{bmatrix} \mathbf{Q}_0 & \mathbf{Q}_1 & \cdots & \mathbf{Q}_{\lambda-1} \\ \mathbf{Q}_{\lambda-1} & \mathbf{Q}_0 & \cdots & \mathbf{Q}_{\lambda-2} \\ \vdots & \vdots & & \vdots \\ \mathbf{Q}_1 & \mathbf{Q}_2 & \cdots & \mathbf{Q}_0 \end{bmatrix}$$

where $\mathbf{Q}_i$ are arbitrary matrices of size $T\rho \times Tb$.

Systematically-quasi-cyclic codes are now defined through their index arrays as a generalization of systematically-cyclic codes defined in Definition 4.6.

Definition 4.8 *A code $\mathcal{C}$ on $b \times n$ arrays and $N_p = \rho n$, ρ an integer, is **systematically-T-quasi-cyclic** if it has an index array representation $\mathsf{A}_{\mathcal{C}}$, in which N_p of the sets are singletons and adding $T\rho$ to all set elements modulo ρn, results in a T-cyclic shift of $\mathsf{A}_{\mathcal{C}}$.*

4.5.1 Construction of the $\kappa_{3\circ}$ codes

The code $\kappa_{3\circ}$ is defined over arrays of size $(p-1) \times 2(p-1)$. Since it is a systematically quasi-cyclic code ($T = 2$), we denote the $N_p = 2(p-1)$ parity constraints in the index array $\mathsf{A}_{\kappa_{3\circ}}$ by $a_0, b_0, a_1, b_1, \ldots, a_{p-2}, b_{p-2}$. The $n = 2(p-1)$ columns of the array will be marked by the same labels. The construction to follow, specifies the contents of "a columns" (a_l) and "b columns" (b_l) of $\mathsf{A}_{\kappa_{3\circ}}$ separately.

Let p be an odd prime and β be a primitive element in GF(p). The permutation ψ : $[0, p-2] \to [0, p-2]$ is defined, as in section 4.4, to be $\psi(x) = \beta^x - 1 \pmod p$. The inverse permutation $\bar{\psi}$ is then $\bar{\psi}(y) = \log_\beta(y+1)$. For any permutation ϕ, we use $\phi(a_i)$ and $\phi(b_i)$ to denote, respectively, $a_{\phi(i)}$ and $b_{\phi(i)}$. Also $a_i + l$,$b_i + l$ are used for a_{i+l},b_{i+l}, respectively, and $[a_s, a_t]$,$[b_s, b_t]$ are used for $\{a_s, a_{s+1}, \ldots, a_t\}$ and $\{b_s, b_{s+1}, \ldots, b_t\}$, respectively.

4.5.1.1 *a* Columns

Define the sets Γ_i, $i \in [0, p-2]$ to be

$$\Gamma_i = \left\{ a_i, b_{\langle i-1 \rangle_p} \right\} \tag{4.2}$$

Define the sets Δ_j, $j \in [1, p-2]$ to be

$$\Delta_j = \left\{ \bar{\psi}(a_j), \bar{\psi}(b_{\langle j-1 \rangle_p}) \right\} \tag{4.3}$$

The a columns of $\mathsf{A}_{\kappa_{3\circ}}$ are now defined. The set in location $(0, a_l)$, $a_l \in [a_0, a_{p-2}]$ is $\{a_l\}$ and the set in location $(j, a_l) \in [1, p-2] \times [a_0, a_{p-2}]$ is $\langle \Delta_j + l \rangle_{p-1}$.

As an example we write the a columns of $\mathsf{A}_{\kappa_{3\circ}}(5)$. For $p = 5$ the sets Γ_i are

$$\Gamma_0 = \{a_0, b_4\},\ \Gamma_1 = \{a_1, b_0\},\ \Gamma_2 = \{a_2, b_1\},\ \Gamma_3 = \{a_3, b_2\}$$

For $\beta = 2$, the permutation $\bar{\psi}$ is $[0,1,2,3] \xrightarrow{\bar{\psi}} [0,1,3,2]$. The sets Δ_j, defined through the permutation $\bar{\psi}$, are

$$\Delta_1 = \{a_1, b_0\},\ \Delta_2 = \{a_3, b_1\},\ \Delta_3 = \{a_2, b_3\}$$

Finally, the a columns of $\mathsf{A}_{\kappa_{3\circ}(5)}$ are provided.

a_0		a_1		a_2		a_3	
a_1, b_0		a_2, b_1		a_3, b_2		a_0, b_3	
a_3, b_1		a_0, b_2		a_1, b_3		a_2, b_0	
a_2, b_3		a_3, b_0		a_0, b_1		a_1, b_2	

4.5.1.2 *b* Columns

Define the following p sets

$$\begin{aligned}\{b_0, b_{p-1}\}, \{b_1, b_{p-2}\}, \ldots, \{b_{(p-3)/2}, b_{(p+1)/2}\}\\ \{a_0, a_{p-1}\}, \{a_1, a_{p-2}\}, \ldots, \{a_{(p-3)/2}, a_{(p+1)/2}\}\\ , \{a_{(p-1)/2}, b_{(p-1)/2}\}.\end{aligned}$$

The indices of every set sum to $p-1$. From the sets above define the following $p-1$ sets

$$\begin{aligned}\{\quad b_0 \quad\}, \{b_1, b_{p-2}\}, \ldots, \{b_{(p-3)/2}, b_{(p+1)/2}\}\\ , \{a_1, a_{p-2}\}, \ldots, \{a_{(p-3)/2}, a_{(p+1)/2}\}\\ , \{a_{(p-1)/2}, b_{(p-1)/2}\}.\end{aligned}$$

The element b_{p-1} was removed from the set $\{b_0, b_{p-1}\}$ and the set $\{a_0, a_{p-1}\}$ was removed altogether. After modifying the sets listed above, the resulting sets contain distinct elements from the sets $[a_0, a_{p-2}]$ and $[b_0, b_{p-2}]$. The sets $\nabla_0, \ldots, \nabla_{p-2}$ are obtained by permuting the sets above using $\bar{\psi}$,

$$\begin{aligned}\{\bar{\psi}(b_0)\}, \{\bar{\psi}(b_1), \bar{\psi}(b_{p-2})\}, \ldots, \{\bar{\psi}(b_{(p-3)/2}), \bar{\psi}(b_{(p+1)/2})\}\\ , \{\bar{\psi}(a_1), \bar{\psi}(a_{p-2})\}, \ldots, \{\bar{\psi}(a_{(p-3)/2}), \bar{\psi}(a_{(p+1)/2})\}\\ , \{\bar{\psi}(a_{(p-1)/2}), \bar{\psi}(b_{(p-1)/2})\}.\end{aligned}$$

The b columns of $\mathsf{A}_{\kappa_{3\circ}}$ are now defined. The set in location $(j, b_l) \in [0, p-2] \times [b_0, b_{p-2}]$ is $\langle \nabla_j + l \rangle_{p-1}$.

As an example we write the b columns of $\mathsf{A}_{\kappa_{3\circ}}(5)$. For $p = 5$, the $p-1$ sets, before operating the $\bar{\psi}$ permutation are

$$\begin{array}{rl} \{ \quad b_0 \quad \}, & \{b_1, b_3\} \\ , & \{a_1, a_3\} \\ , & \{a_2, b_2\}. \end{array}$$

After applying the $\bar{\psi}$ permutation, the sets $\nabla_0, \nabla_1, \nabla_2, \nabla_3$ are obtained

$$\begin{array}{rl} \{ \quad b_0 \quad \}, & \{b_1, b_2\} \\ , & \{a_1, a_2\} \\ , & \{a_3, b_3\}. \end{array}$$

Finally, the b columns of $\mathsf{A}_{\kappa_{3\circ}(5)}$ are provided.

	b_0		b_1		b_2		b_3
	b_1, b_2		b_2, b_3		b_3, b_0		b_0, b_1
	a_1, a_2		a_2, a_3		a_3, a_0		a_0, a_1
	a_3, b_3		a_0, b_0		a_1, b_1		a_2, b_2

By mapping the indices $(a_0, b_0, \ldots, a_{p-2}, b_{p-2})$ to the integer indices $(0, 1, \ldots, 2p - 3)$, the code $\kappa_{3\circ}$ clearly satisfies the requirements of Definition 4.8, hence

Proposition 4.6 *The code $\kappa_{3\circ}$ is systematically 2-quasi-cyclic.*

The rest of this section is devoted to proving that $\kappa_{3\circ}$ is an MDS code.

4.5.2 Proof of the MDS property

To prove the MDS property of the codes $\kappa_{3\circ}$, a two step proof will be carried out. First we define a different, non-quasi-cyclic code κ_3, and show that it is MDS. Then we show a distance preserving mapping from the rows and columns of the parity-check matrix of κ_3 to those of $\kappa_{3\circ}$. κ_3 is now defined. The definition only specifies the sets of each column of

A_{κ_3}, without specifying the set locations within a column. This definition suffices for the MDS proof and for the mapping provided later. The array dimensions and code parameters of κ_3 are identical to those of $\kappa_{3_\circ}$.

Definition 4.9 *The columns $a_0, b_0, a_1, b_1, \ldots, a_{p-2}, b_{p-2}$ of the code κ_3 are defined as follows.*
1) An a column $a_l \in [a_0, a_{p-2}]$ of A_{κ_3} contains the set $\{a_l\}$ and all sets $\{a_m, b_{m'}\}$ such that $m - m' = l + 1 \pmod p$. Only the $p-2$ such sets with $(m, m') \in [0, p-2] \times [0, p-2]$ are taken.
2) A b column $b_l \in [b_0, b_{p-2}]$ of A_{κ_3} contains the set $\{b_l\}$, the set $\{a_{(l-1)/2}, b_{(l-1)/2}\}$, and all sets $\{a_m, a_{m'}\}$ and $\{b_m, b_{m'}\}$ such that $m + m' = l - 1 \pmod p$. Here too, only the $p-3$ sets with $(m, m') \in [0, p-2] \times [0, p-2]$ are taken.

To prove the MDS property of κ_3, we define and use a graphical interpretation of index arrays. This interpretation can be applied when the index array $\mathsf{A}_{\mathcal{C}}$, of a binary parity-check matrix, has only sets of sizes two or less. Given an index array whose union of sets is $\{0, 1, \ldots, \mathrm{R}-1\}$, denote by $\mathsf{K}_{\mathrm{R}+1}$ the complete graph on the $\mathrm{R}+1$ vertices labeled $\{0, 1, \ldots, \mathrm{R}-1, \infty\}$. Each set of size two, $\{x, y\}$, defines a subgraph of $\mathsf{K}_{\mathrm{R}+1}$, called set-subgraph, that has the vertices x, y and an edge connecting them. Each set of size one, $\{x\}$, defines a set-subgraph of $\mathsf{K}_{\mathrm{R}+1}$ that has the vertices x, ∞ and an edge connecting them. A bit [2] assignment to an array corresponds to the union of set-subgraphs in locations with non-zero entries. The following is a simple but useful observation.

Proposition 4.7 *A bit assignment to an array is a codeword of $\mathcal{C}$ if and only if all vertices have even degrees in its $\mathsf{A}_{\mathcal{C}}$ set-subgraph union (the subgraph is a cycle or a union of edge-disjoint cycles, with possibly some isolated vertices).*

The above graphical interpretation is now explained with an example.

[2] A similar interpretation works for array symbols from any Abelian group

Example 4.1 *Let the array code $\mathcal{C}$ be defined by the following index array.*

$A_{\mathcal{C}} =$

0	1	2	3	4	5
4,5	5,0	0,1	1,2	2,3	3,4
1,3	2,4	3,5	4,0	5,1	0,2

The word

$V_1 =$

1	0	1	0	0	0
0	1	0	0	0	0
0	1	0	0	0	0

has the set-subgraph union in Figure 4.2*(a). Vertices* $4, 5$ *have odd degrees of* 1*, and thus the word* V_1 *is not a codeword of* $\mathcal{C}$*. On the other hand, the word*

$V_2 =$

0	1	0	1	0	0
0	1	1	0	0	0
1	0	0	0	1	0

has the set-subgraph union in Figure 4.2*(b). All vertices have even degrees and thus* V_2 *is a codeword of* $\mathcal{C}$*.*

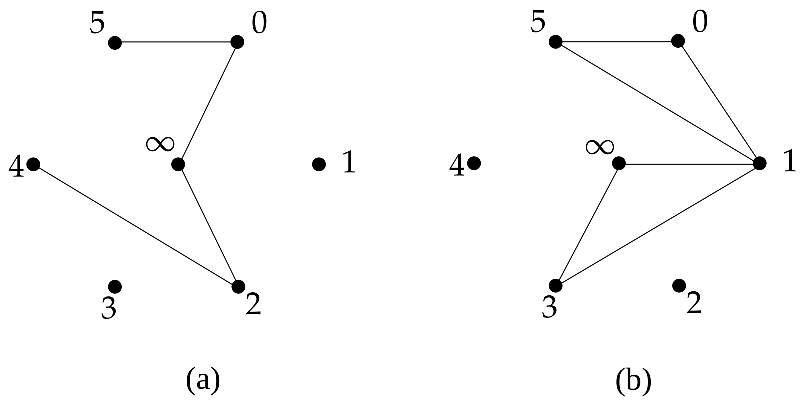


Figure 4.2: Set-subgraph unions of the code $\mathcal{C}$. (a) For the word V_1. (b) For the codeword V_2.

The next Lemma establishes the MDS property of κ_3 by showing that there are no codewords of column weight smaller than 3.

Lemma 4.8 *For any two columns from $\{a_0, b_0, a_1, b_1, \ldots, a_{p-2}, b_{p-2}\}$, there are no non-zero codewords of κ_3 that are all zero outside these two columns.*

Proof: For each pair of columns, the proof will show that no subgraph of the set subgraph corresponding to these two columns, can contain a cycle. Hence there are no non-zero codewords with column weight 2 or less. We distinguish between three cases. A similar proof, but for a different combinatorial construct (which does not yield quasi-cyclic codes) appears in [And73].

Case 1: Two a columns contain all non-zero locations.

For columns a_l and a_{l+v} such that $0 \le l < l+v \le p-2$, the set-subgraph is given in Figure 4.3. A solid edge comes from a set in column a_l and a dashed edge comes from a set in column a_{l+v}. Note that the edges satisfy the constraints of 1 in Definition 4.9. To

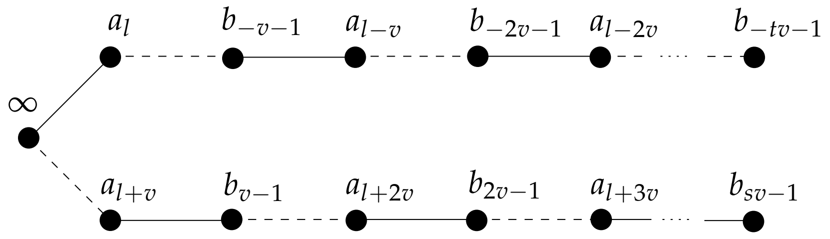


Figure 4.3: Set-subgraph of columns a_l,a_{l+v}. Solid edges represent sets from column a_l whose indices have difference $l+1$. Dashed edges represent sets from column a_{l+v} whose indices have difference $l+v+1$. Having a cycle as a subgraph implies either $l - tv \equiv l + sv \pmod{p}$ (an a vertex shared by top and bottom branches) or $-tv - 1 \equiv sv - 1 \pmod{p}$ (a b vertex shared by top and bottom branches). Each results in a contradiction.

have a cycle as a subgraph, there must exist two integers s, t such that $s + t < p$ and either $l - tv \equiv l + sv \pmod{p}$ or $-tv - 1 \equiv sv - 1 \pmod{p}$. The first condition refers to the case when an index of a from the upper chain is identical to an index of a from the lower chain (and thus a cycle is created). The second condition refers to the case when an index of b from the upper chain is identical to an index of b from the lower chain. Each of the conditions requires $(s+t)v \equiv 0 \pmod{p}$, which is a contradiction for a prime p.

Case 2: Two b columns contain all non-zero locations.

For columns b_l and b_{l+v} such that $0 \le l < l+v \le p-2$, the set-subgraph is given in Figure 4.4. The edges satisfy the constraints of 2 in Definition 4.9. Cycles with an odd number of edges are not possible since elements appear at most once in every column (any

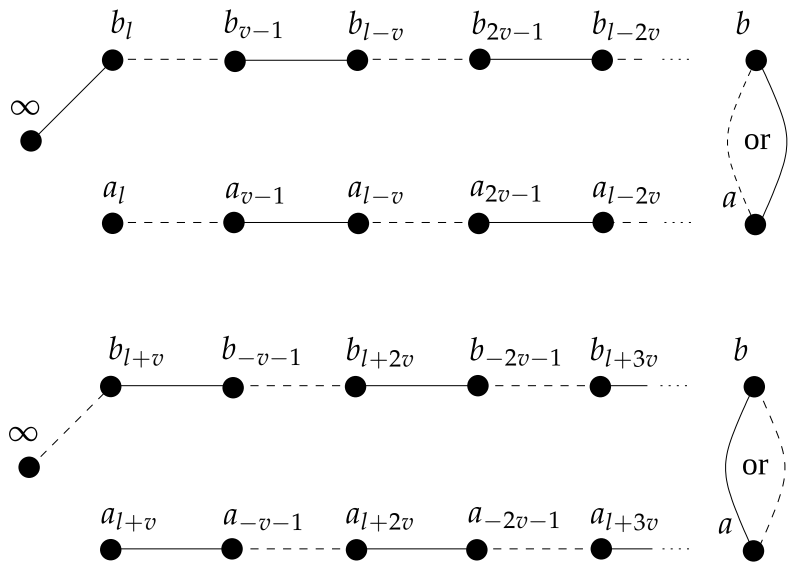


Figure 4.4: Set-subgraph of columns b_l,b_{l+v}. Solid edges represent sets from column b_l whose indices sum to $l-1$. Dashed edges represent sets from column b_{l+v} whose indices sum to $l+v-1$. Having a cycle as a subgraph implies either $l-tv \equiv l+sv \pmod{p}$ (an a vertex shared by top and bottom chains) or $-tv-1 \equiv sv-1 \pmod{p}$ (a b vertex shared by top and bottom chains). Each results in a contradiction.

vertex has one solid edge and one dashed edge incident on it). To have a cycle with an even number of edges, the same contradictory conditions of Case 1 apply.

Case 3: One a column and one b column contain all non-zero locations.

Denote the non-zero columns by a_l and b_ℓ. A solid edge comes from a set in column a_l and a dashed edge comes from a set in column b_ℓ. Assume first that the cycle does not contain the edge that corresponds to the special set $\{a_{(\ell-1)/2}, b_{(\ell-1)/2}\}$. Then the number of edges in the cycle is a multiple of 4 (because of the $a \dashrightarrow a \rightarrow b \dashrightarrow b \rightarrow a$ structure), and it has the structure of Figure 4.5. For each path of length 4 of the pattern $a \dashrightarrow a \rightarrow b \dashrightarrow b \rightarrow a$, the index of the final a vertex is greater by $2l+2$ modulo p than the index of the initial a vertex. Therefore, as seen at the top vertex in Figure 4.5, the existence of such a cycle depends on the condition that $i \equiv i + 2s(l+1) \pmod{p}$, for some $s < (p-1)/2$. This is a contradiction for a prime p and $l < p-1$. Now assume that there exists a cycle that does contain the edge $\{a_{(\ell-1)/2}, b_{(\ell-1)/2}\}$. In that case there exists a path from a_ℓ to b_{-l-2} (the only two vertices with degree 1) with the structure of Figure 4.6. For each path of length 4 of the pattern $b \rightarrow a \dashrightarrow a \rightarrow b \dashrightarrow b$, the index of the final b vertex is greater by $2l+2$ modulo p than the index of the initial

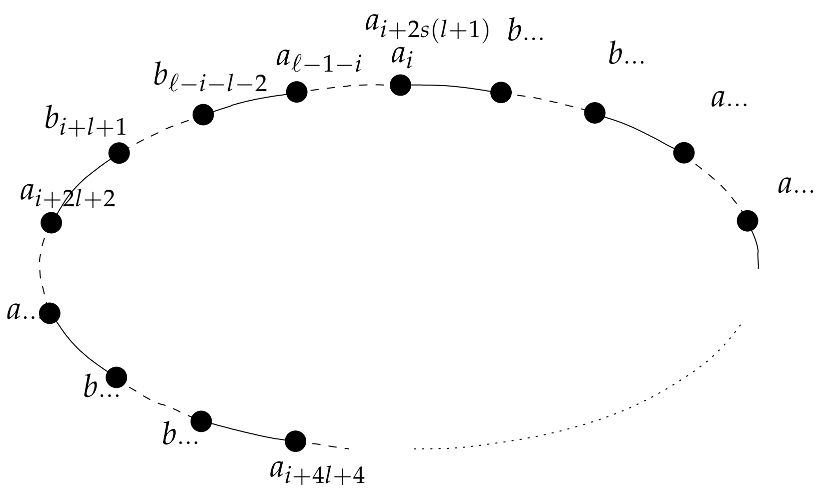


Figure 4.5: A cycle from columns a_l,b_ℓ. Solid edges represent sets from column a_l whose indices have difference $l+1$. Dashed edges represent sets from column b_ℓ whose indices sum to $\ell-1$. The two indices assigned to the top vertex imply that $i = i + 2s(l+1) \pmod{p}$, which results in a contradiction.

b vertex. Therefore, as seen at the top right vertex in Figure 4.6, the existence of such a path depends on the condition that $-l-2 \equiv l + 2s(l+1) \pmod{p}$, or equivalently $2(s+1)(l+1) \equiv 0 \pmod{p}$, for some $s < (p-1)/2$. This is again a contradiction for a prime p and $l < p-1$. □

Lemma 4.9 *$\mathsf{A}_{\kappa_{3\circ}}$ can be obtained from A_{κ_3} by a minimum-distance preserving transformation.*

Proof: We show that by permuting the indices of $\mathsf{A}_{\kappa_{3\circ}}$, its columns and sets within its columns, A_{κ_3} can be obtained. All these operations preserve the redundancy, minimum distance of the code and its density. We provide the transformation and prove its aforementioned property for a and b columns separately.

***a* Columns:**

Recall that the set in location $(j, a_l) \in [1, p-2] \times [a_0, a_{p-2}]$ of $\mathsf{A}_{\kappa_{3\circ}}$ is

$$\{\langle \bar{\psi}(a_j) + l \rangle_{p-1} \, , \, \langle \bar{\psi}(b_{j-1}) + l \rangle_{p-1}\}.$$

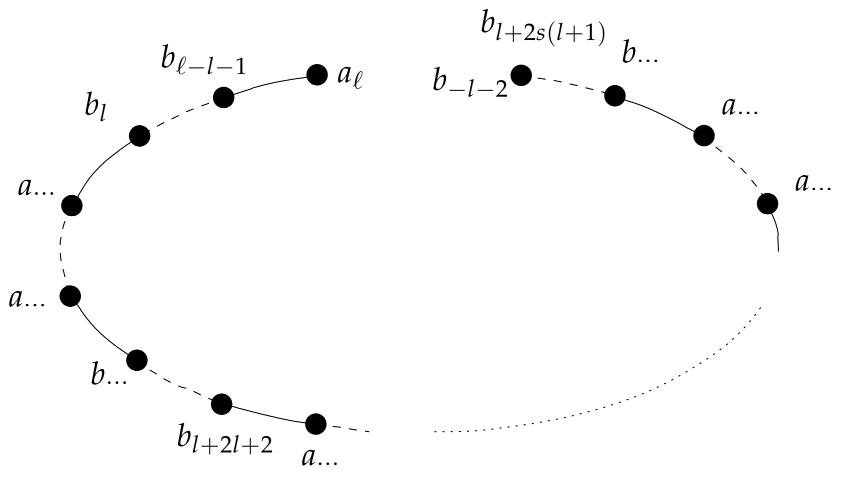


Figure 4.6: A path from columns a_l,b_ℓ. If there exists a path from vertex a_ℓ to vertex b_{-l-2} then the two indices assigned to the right top vertex imply that $-l-2 = l+2s(l+1) \pmod{p}$, which results in a contradiction.

To show the transformation we look at the difference between the a index and the b index above

$$\langle\bar{\psi}(j)+l\rangle_{p-1} - \langle\bar{\psi}(j-1)+l\rangle_{p-1},$$

and permute each summand using ψ to get

$$\psi\left[\langle\bar{\psi}(j)+l\rangle_{p-1}\right] - \psi\left[\langle\bar{\psi}(j-1)+l\rangle_{p-1}\right] =$$

substituting the permutations $\psi, \bar{\psi}$ we write

$$= \beta^{\log_\beta(j+1)+l} - 1 - \beta^{\log_\beta(j)+l} + 1 =$$

$$= \beta^l(j+1-j) = \beta^l - 1 + 1 = \psi(l) + 1.$$

In words, pairs of a, b indices of $\mathsf{A}_{\kappa_{3\circ}}$, after permutation, have the same relation as the pairs of indices of A_{κ_3} (as defined in 1 of Definition 4.9), with columns permuted by the same permutation. Since all elements in the sets of column l of $\mathsf{A}_{\kappa_{3\circ}}$ are distinct, permuting the indices and columns using ψ results in the same sets that form A_{κ_3}.

***b* Columns:**

We proceed similarly to the previous case but this time look at the sum

$$\psi\left[\langle\bar{\psi}(j)+l\rangle_{p-1}\right]+\psi\left[\langle\bar{\psi}(p-1-j)+l\rangle_{p-1}\right]=$$

and substitute $\psi, \bar{\psi}$ to get

$$=\beta^{\log_\beta(j+1)+l}-1+\beta^{\log_\beta(p-j)+l}-1=$$

$$=\beta^l(j+1-j)-2=\beta^l-1-1=\psi(l)-1.$$

For b columns too, permuting the indices and columns of $\mathsf{A}_{\kappa_{3\circ}}$ results in the sets of A_{κ_3} (as defined in 2 of Definition 4.9). □

Lemma 4.8 and Lemma 4.9 together prove the main theorem of the section.

Theorem 4.10 *For every prime p, $\kappa_{3\circ}(p)$ has minimum column distance 3, and thus it is an MDS code.*

4.6 Implementation Benefits of Cyclic and Quasi-Cyclic Array Codes

Cyclic and Quasi-Cyclic array codes possess a more regular structure relative to general array codes. Regular structures often simplify the realization of error-correcting codes in complexity-limited systems. Cyclic $b \times n$ array codes can be specified using only b sets, compared to the nb sets that are required to specify non-cyclic codes. That means that encoder/decoder designs are much simpler for cyclic codes and they require lower storage overhead for decoding tables. A pictorial illustration of this advantage is given in Figures 4.7 and 4.8. The graph that represent a 3×6 array code is given in Figure 4.7. The 18 nodes of the graph marked with ◯ represent the 18 array bits, partitioned to 6 groups, each represents an array column. The 6 nodes marked ⊞ , represent the parity constraints that the array bits must satisfy. To implement the code, 30 edges need to be specified, resulting in a complex realization of encoders and decoders. However, if the code is cyclic, then the description of Figure 4.8, with only 5 edges, is sufficient and allows a simpler regular implementation of the code (all other bit groups have the same local connectivity, appropriately shifted cyclically). In particular, when the array code is implemented in

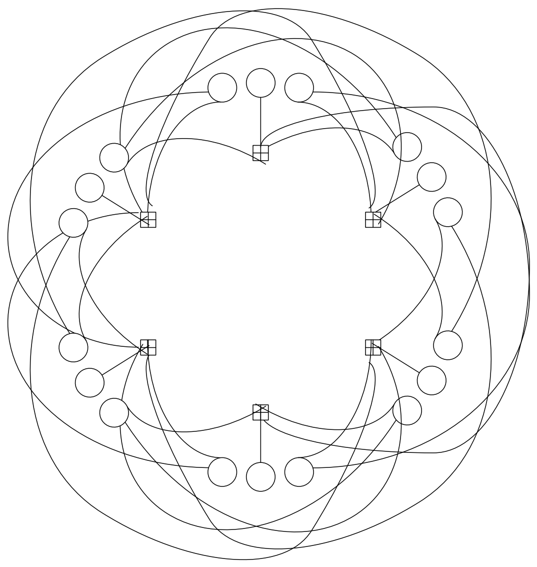


Figure 4.7: A full description of a sample 3×6 array code using 30 edges.

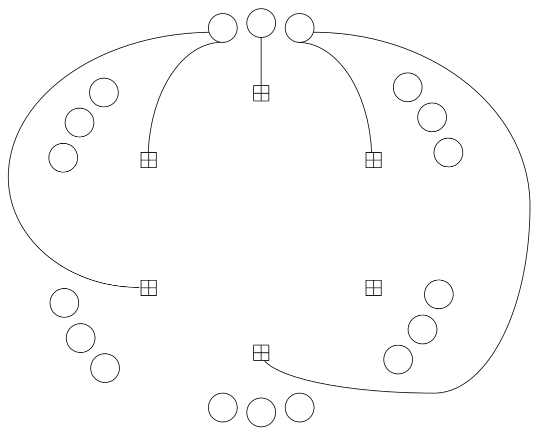


Figure 4.8: A compact description of a sample 3×6 *cyclic* array code using 5 edges.

a distributed fashion, as practiced in storage and network-storage applications, the cyclic symmetry of the codes allows using a single uniform design for all nodes, contrary to non-cyclic codes in which each node needs to perform different operations.

Though the exact advantage of cyclic codes depends on the qualities and constraints of particular implementations, we next attempt to motivate their use in general, by illustrating some of their properties. The properties are given for cyclic codes only, but quasi-cyclic codes enjoy similar properties with a slightly reduced symmetry.

4.6.1 Encoding and Updates

Property 4.1 *In a systematically-cyclic array code (see Definition 4.5), if updating an information symbol at array location (j, l) requires updating parity symbols at array locations $\{(j_1, l_1), \ldots, (j_r, l_r)\}$, then updating an information symbol at array location $(j, l + s)$ requires the same parity updates at array locations $\{(j_1, l_1 + s), \ldots, (j_r, l_r + s)\}$, where all $+$ operations are modulo n.*

This property, established directly from the parity-check matrix structure of systematically-cyclic array codes, simplifies the circuitry needed for bit updates, an operation that is invoked at a very high rate in a typical dynamic storage application. In cylindrical storage arrays, it also allows to update a group of array symbols without absolute angular synchronization. Cyclic codes that are not systematically cyclic do not enjoy the same property, in general.

4.6.2 Syndrome Calculation

The syndrome $\boldsymbol{s}$ of a word R with dimensions $b \times n$ is obtained by first converting it, by column stacking its elements, to a length nb column vector $\boldsymbol{r}$. Then it is defined as $\boldsymbol{s} = H\boldsymbol{r}$. Computing the syndrome is a first step in error and erasure decoding of array codes. A more economic calculation of syndrome symbols is achieved for cyclic array codes thanks to the following property.

Property 4.2 *In a cyclic array code, if symbol i of the syndrome is a function f of the symbols in the following array locations $f[(j_1, l_1), (j_2, l_2), \ldots]$, then symbol $i + s$ of the syndrome is the function $f[(j_1, l_1 + s), (j_2, l_2 + s), \ldots]$, indices taken modulo n.*

4.6.3 Erasure and Error Decoding

Property 4.3 *If in a cyclic array code, a set of erased columns $\Lambda = \{i_1, \dots, i_t\}$ is recovered by a matrix vector product $H_\Lambda^{-1}\mathbf{s}$, where $\mathbf{s}$ is the syndrome of the codeword with missing symbols set to zero, then the set of erased columns $\Lambda_s = \{i_1 + s, \dots, i_t + s\}$ (indices modulo n) is recovered by $H_\Lambda^{-1}U_s\mathbf{s}$, where U_s is the sparse matrix that cyclically shifts the syndrome ρs locations upward.*

This property relies on the fact that for cyclic codes, $H_{\Lambda_s} = D_s H_\Lambda$, where D_s is the sparse matrix that cyclically shifts the rows of H_Λ, ρs locations downward. Taking the inverse results in $H_{\Lambda_s}^{-1} = H_\Lambda^{-1}D_s^{-1} = H_\Lambda^{-1}U_s$. The benefit of that property is that many of the decoding matrices are cyclically equivalent, and therefore only a $1/n$ portion of decoding matrices needs to be stored, compared to non-cyclic array codes with the same parameters. A similar advantage exists for error decoding, where the cyclic equivalence of syndromes allows a simpler error location.

4.7 Conclusion

Beyond the practical benefit of the constructed cyclic codes, these codes and their relationship to known non-cyclic codes raise interesting theoretical questions. The indirect proof technique used for all three code families is a distinctive property of the code constructions. It is curious that a direct MDS proof of the more structured cyclic codes, seems hard to come by. Such a proof may reveal more about the structure of these codes and possibly allow finding new code families. This optimistic view is supported by computer searches that find cyclic lowest-density MDS codes with parameters that are not covered by the known families of non-cyclic codes.

Part II

Decoding Techniques

Chapter 5

Decoding Beyond Half the Minimum Distance

Instances of failed decoding are especially undesirable in data-storage systems, since they cost a permanent loss of user data. Stronger decoders that can correct more errors are therefore sought, to improve the system's reliability without introducing additional redundancy. This task of increasing the decoder's decoding radius entails two major challenges: high decoding complexity and increased miscorrection probability. Both issues are the subject of this chapter. The main contributions of this chapter can be summarized as follows.

- Analysis and optimization of Reed-Solomon list decoders based on the *instantaneous* number of errors.
- A new lower bound on the miscorrection probability of list decoders.
- The best known closed-form upper bound on the list size for codes over moderately large alphabets. The same bound also improves over the classical q-ary Johnson bound for constant-weight codes.

The majority of the results in this chapter have appeared in [CB05] and in [CB04].

5.1 Introduction

The core precept of Coding Theory is the trade-off between redundancy and correction capability. Countless constructions and bounds couple codes' correction capabilities with the corresponding redundancy that they carry. More often than not, correction capability is measured in worst-case terms, allowing the employment of combinatorial and algebraic analysis tools over the Hamming and other metric spaces. A far less studied framework, at least in the combinatorial/algebraic domain, is that of fixing the code redundancy, and analyzing the performance of *decoders* with increased correction capabilities. This area of study is called *list decoding*, debuted in the two articles by Elias [Eli57] and by Wozencraft [Woz58]. Despite the overwhelming theoretical horizons opened by these works, list decoding remained off the coding-theory mainstream, mainly because of the absence of algorithmic solutions to increasing the decoding radius. A major swing to the favor of list decoding ensued when Sudan introduced a polynomial-time list decoder for Reed-Solomon codes [Sud97], that had a strong impact on a multitude of active research areas in Theoretical Computer Science. In the Information Theory community, the advent of efficient list-decoding algorithms has also created a great interest, focused on understanding the issues of applying list-decoding schemes in communication and data-storage systems. This chapter mostly follows the latter, more practically oriented, research trajectory of list decoding. In particular, it adds insight and novelty in the following study fronts:

- **Algorithmic efficiency.** How can list decoding be made less complex? A finer-grain analysis and optimization of list-decoding algorithms, which is outside the scope of their study in the computer science domain.

- **Non-uniqueness of decoding.** What are the practical consequences of decoding beyond the unique-decoding bound? Study of possible decoding outcomes of list decoders, reasoning about their interpretation and impact on performance.

Discussing the algorithmic list-decoding problem in sections 5.2, 5.3, 5.4, and 5.5, we focus on Reed-Solomon codes for both hard-decision and soft-decision decoding. While the worst-case list-decoding complexity of Reed-Solomon codes is well understood as a function of the code parameters and the decoding radius, this analysis ignores the effect of the number of *instantaneous errors* on the decoding complexity. In systems that employ RS codes, the average number of instantaneous errors introduced by the channel, is typically much lower than the decoder's worst-case decoding radius. Hence optimizing the decoding complexity with respect to the number of instantaneous errors, improves the average decoding time and in turn the decoder's throughput. An average-case analysis of classical RS decoders is pursued in [BM85], where the running-time dependence on the error weight is obtained experimentally. The study of [GKKG06] seeks to improve the average case complexity of the RS algebraic soft decision decoder [KV03a], by using a layered decoder whose decode time depends on the instantaneous channel noise. In this chapter, the average-case analysis hinges on the dependence of the *interpolation cost* (the number of required interpolation coefficients) on the error weight. This dependence is studied and quantified in section 5.3 using analytical tools for hard-decision decoders. Then, in section 5.4 an interpolation algorithm is proposed whose running time favorably depends on the instantaneous interpolation cost. This algorithmic proposition achieves improved average-case running time for both hard-decision and soft-decision decoding. Finally, in section 5.5, a comparison of the instantaneous interpolation costs of hard-decision and soft-decision decoders is carried out using simulations of both decoders.

The second part of the chapter in sections 5.6 and 5.7, discusses decoder behaviors and code properties of general error-correcting codes, occasionally using Reed-Solomon codes only as examples. The effect of increased decoding radius on the *miscorrection probability* is especially interesting, and section 5.6 adds insight on that issue. In particular, it shows that miscorrections occur significantly more frequently, even for small increases in the decoding radius, questioning "popular belief" that a small average list-size implies that list decoders behave essentially the same as unique decoders. Section 5.7 presents a closed-form upper bound on the decoder's output-list size. This bound is an important tool to achieve the bounds on miscorrection in the preceding section, and a very interesting combinatorial result in its own. This bound joins many other attempts at bounding the size of the list, in both the Information Theory and Computer Science communities. For moderately large alphabets it is the best known closed-form bound, and its generality allows using bounds on binary constant-weight codes to further tighten it.

5.2 Review of Guruswami-Sudan Algebraic List Decoding

A codeword C from an $[n, k, d]$ Reed-Solomon (RS) code is the evaluation of a degree $k-1$ or less message polynomial $f(x)$ on n distinct points of GF(q), $\{\alpha_1, \ldots, \alpha_n\}$. Let E be an error vector of Hamming weight e over the same alphabet GF(q). The received word R is defined as $R = C + E$, over GF(q) arithmetic. Classical decoding algorithms of RS codes, e.g. the Berlekamp algorithm, the Massey algorithm, and their predecessor Peterson-Gorenstein-Zierler algorithm (see [Bla83] for description of the algorithms), all attempt to efficiently solve the following linear system of ν equations (ν = number of errors):

$$\begin{bmatrix} S_1 & S_2 & \cdots & S_{\nu-1} & S_\nu \\ S_2 & S_3 & \cdots & S_\nu & S_{\nu+1} \\ S_3 & S_4 & \cdots & S_{\nu+1} & S_{\nu+2} \\ \vdots & & & & \vdots \\ S_\nu & S_{\nu+1} & \cdots & S_{2\nu-2} & S_{2\nu-1} \end{bmatrix} \begin{bmatrix} \Lambda_\nu \\ \Lambda_{\nu-1} \\ \Lambda_{\nu-2} \\ \vdots \\ \Lambda_1 \end{bmatrix} = \begin{bmatrix} -S_{\nu+1} \\ -S_{\nu+2} \\ -S_{\nu+3} \\ \vdots \\ -S_{2\nu} \end{bmatrix}$$

where Λ_i are the coefficients of the unknown error-locator polynomial and S_j are the known syndromes. When $d \geq 2\nu + 1$, this system of equations has a unique solution and thus the algorithms mentioned above can decode errors up to half the minimum distance: the unique-decoding bound.

A completely different approach to decoding RS codes, that can correct more errors than classical algorithms, has been introduced by Sudan [Sud97], and improved by Guruswami and Sudan [GS99], using relatively simple but powerful algebraic-geometric ideas. In the Guruswami-Sudan (GS) algorithm [GS99], the received word is used to interpolate a bivariate polynomial $Q(x, y)$. To achieve a large correction radius, $Q(x, y)$ should be the minimal $(1, k-1)$-weighted degree[1] bivariate polynomial that satisfies the following $n\binom{m+1}{2}$ constraints: $D_{r,s}Q(\alpha_i, R_i) = 0$ for $i = \{1, \ldots, n\}$ and $\{(r, s) : r + s < m\}$. m is a decoder parameter called the *interpolation multiplicity*. $D_{r,s}Q(\alpha, \beta)$ is the Hasse

[1]the $(1, v)$-weighted degree of a bivariate polynomial is the maximum over all of its monomials $x^i y^j$ of the sum $i + vj$

derivative of x-order r and y-order s, evaluated on the point $x = \alpha, y = \beta$ (more on Hasse derivatives in sub-section 5.3.1 below and in [McE03a].) If those interpolation constraints are satisfied by $Q(x, y)$, it is guaranteed that codewords within the prescribed decoding radius of the decoder will be found by factorization of $Q(x, y)$. A block diagram of the GS list-decoding algorithm is given in Figure 5.1 below.

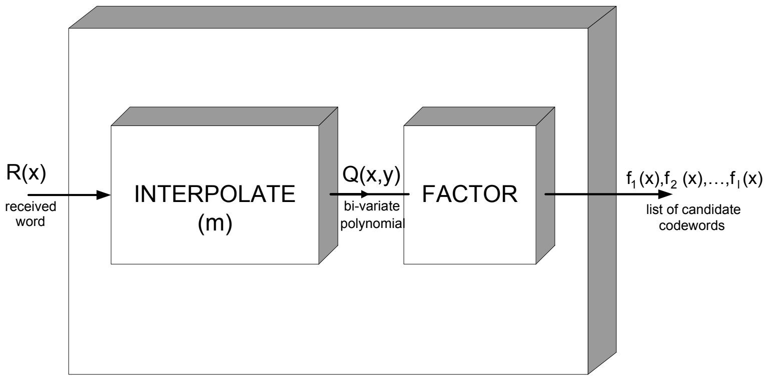


Figure 5.1: Block diagram of the Guruswami-Sudan list-decoding algorithm

By formulating the interpolation as a system of homogeneous linear equations it has been observed that $n\binom{m+1}{2} + 1$ coefficients are sufficient to make $Q(x, y)$ satisfy the above constraints. We denote by C_{wc} this worst case number of interpolation coefficients, so $\mathsf{C}_{wc} = n\binom{m+1}{2} + 1$. C_{wc} will be later called the worst case *interpolation cost* of the GS (n, k, m) decoder. The key yield from that decoding scheme is that a sufficient condition to correct t errors is $m(n - t) > d_{1,k-1}(\mathsf{C}_{wc})$, where $d_{1,k-1}(J)$ is the minimal $(1, k - 1)$-weighted degree of a bivariate polynomial with J coefficients. Since in general the number of correctable errors t is larger than $\lfloor (d - 1)/2 \rfloor$, half the minimum distance of the code, the decoder output is a *list* that possibly contains multiple codewords. Hence the qualifier *list-decoding* is used for the GS decoder, as well as for other decoders that correct beyond the unique decoding bound $\lfloor (d - 1)/2 \rfloor$. Throughout the chapter, we assume that the monomials of the interpolating polynomials are ordered by nondecreasing $(1, k - 1)$-weighted degrees, with reverse-lexicographic tie-breaking, i.e. $x^{(k-1)s}$ precedes $x^{(k-1)(s-1)}y$ (or in general, if two monomials have the same $(1, k - 1)$-weighted degree, then a monomial with lower y-degree precedes others with higher y-degrees). For a more detailed discussion of multivariate polynomials and monomial ordering please refer to [CLO97].

5.3 Interpolation Polynomials for Low-Weight Errors

In this section we develop tools for bounding the interpolation cost given an error weight. The bounds are achieved by providing classes of interpolating polynomials for received words resulting from an error of a given weight, and then analyzing the degrees of these polynomials to get upper bounds on interpolation costs. To this end we have introduced the worst case interpolation cost C_{wc}, which is determined by the decoder parameters n, k, m. For the sake of the forthcoming analysis, we define the error-weight dependent interpolation cost $\mathsf{C}_e(\epsilon)$ as the number of interpolation coefficients required given an error word ϵ of Hamming weight e. Note that $\mathsf{C}_e(\epsilon)$ is not a function of e alone; different interpolation costs are possible for different error words of a given weight e.

5.3.1 Hasse derivatives

For their central role in the interpolation procedure, Hasse derivatives and their properties are discussed in detail.

Definition 5.1 (The (r,s) Hasse derivative) *The (r,s) Hasse derivative of a polynomial $Q(x,y)$, denoted $D_{r,s}Q(x,y)$ is defined as*

$$D_{r,s}Q(x,y) = \sum_{i,j} \binom{i}{r}\binom{j}{s} a_{i,j} x^{i-r} y^{j-s}$$

where $a_{i,j}$ is the coefficient of $x^i y^j$ in $Q(x,y)$.

Hasse derivatives owe their use in RS list-decoding to the following fact

$$D_{r,s}Q(\alpha,\beta) = \text{coeff}_{x^r y^s} Q(x+\alpha, y+\beta)$$

In words, the coefficient of $x^r y^s$ in the polynomial $Q(x+\alpha, y+\beta)$ equals the (r,s) Hasse derivative of the polynomial $Q(x,y)$, evaluated at the point $x=\alpha, y=\beta$. We now turn to state (without proof) the well-known product rule for Hasse derivatives.

Lemma 5.1 (The Hasse Derivative product rule)

The Hasse derivative of a product of L polynomials

$$D_{r,s}\left[\prod_{i=1}^{L} Q_i\right] = \sum_{\substack{r_1+\cdots+r_L=r \\ s_1+\cdots+s_L=s}} \prod_{i=1}^{L} D_{r_i,s_i} Q_i$$

From Lemma 5.1 we get the following lemma.

Lemma 5.2 *If $Q(\alpha,\beta) = 0$, then for every $\{r,s : r+s < m\}$, $D_{r,s}[Q(\alpha,\beta)^m] = 0$, where $D_{r,s}[Q(\alpha,\beta)^m]$ is the r,s Hasse derivative of $Q(x,y)^m$, evaluated at (α,β).*

Proof: Lemma 5.1 states that

$$D_{r,s}[Q(x,y)^m] = \sum_{\substack{r_1+\cdots+r_m=r \\ s_1+\cdots+s_m=s}} \prod_{i=1}^{m} D_{r_i,s_i} Q(x,y)$$

since $r+s = \sum_{i=1}^{m}(r_i+s_i) < m$, for every assignment to $r_1, s_1, \ldots, r_m, s_m$ at least one of the pairs (r_i, s_i) equals $(0,0)$. That means every product in the sum contains at least one factor $D_{0,0}Q(x,y) = Q(x,y)$. Substituting $(x,y) = (\alpha,\beta)$, the right hand side evaluates to zero. □

5.3.2 Closed form upper bound on interpolation cost

Theorem 5.3 *Let E be an error vector of weight e and let $\{j_1, j_2, \ldots, j_e\}$ be the error locations. Then there exists an interpolation polynomial whose last monomial (according to the $(1,k-1)$-weighted degree order with reverse-lexicographic tie breaking) has (x,y)-degree of (em,m). This polynomial can be explicitly written as*

$$Q(x,y) = \left[(y-f(x))(x-\alpha_{j_1})(x-\alpha_{j_2})\cdots(x-\alpha_{j_e})\right]^m \tag{5.1}$$

Proof:

$$D_{r,s}Q(x,y) = D_{r,s}\left[(y-f(x))^m(x-\alpha_{j_1})^m \cdots (x-\alpha_{j_e})^m\right] =$$

$$= \sum_{\substack{r_0+r_1+\cdots+r_e=r \\ s_0+s_1+\cdots+s_e=s}} D_{r_0,s_0}(y-f(x))^m \prod_{i=1}^{e} D_{r_i,s_i}(x-\alpha_{j_i})^m =$$

$$= \sum_{r_0+r_1+\cdots+r_e=r} D_{r_0,s}(y-f(x))^m \prod_{i=1}^{e} D_{r_i,0}(x-\alpha_{j_i})^m$$

If (r,s) satisfy $r+s<m$, then obviously $r_0+s<m$ and $r_i<m$ for $i=\{1,\ldots,e\}$. Therefore by Lemma 5.2, any product in the sum will have both a factor of $y-f(x)$ and factors of $x-\alpha_{j_i}$, for all $i=\{1,\ldots,e\}$. This establishes that $D_{r,s}Q(\alpha_i,R_i)=0$ for both the correct symbols and the corrupted symbols. □

The strength of the arguments used in the proof above is that they allow to predict the *form* of interpolating polynomials for any error weight, even without constructively interpolating particular received words.

Taking the polynomial structure of (5.1) with some straightforward monomial counting we get a bound on $\mathtt{C}_e(\epsilon)$ in the following corollary.

Corollary 5.4 *Let* $\Delta = m(e+k-1)$ *and* $r = \Delta \bmod (k-1)$. *For any error of weight e we have the following bound*

$$\mathtt{C}_e(\epsilon) \le \frac{\Delta^2}{2(k-1)} + \frac{\Delta}{2} + \frac{r(k-r-1)}{2(k-1)} + m + 1 \tag{5.2}$$

Proof: Theorem 5.3 proves that there exists an interpolating polynomial with $(1,k-1)$-weighted degree of $em+(k-1)m=\Delta$, whose last monomial (according to the monomial order) is $x^{em}y^m$. The expression in the right hand side of (5.2) is the straightforward calculation of the position of $x^{em}y^m$ in the monomial order, or equivalently the interpolation cost. The inequality comes from the fact that there *may be* other interpolating polynomials, besides the one of Theorem 5.3, with lower interpolation cost. □

5.3.3 The no errors case

Theorem 5.5 *Let $\rho = k/n$ be the rate of the RS code. When the received word R is a codeword, the interpolation cost $\mathsf{C}_0(\epsilon)$ satisfies*

$$\mathsf{C}_0(\epsilon) \leq \lceil \rho \mathsf{C}_{wc} \rceil$$

Proof: Define $v = k - 1$. When R is a codeword, R can be interpolated by the bivariate polynomial $Q(x, y) = (y - f(x))^m$. The last monomial of $(y - f(x))^m$ in the monomial order is y^m, whose $(1, v)$-degree is mv.

Lemma 5.6 *The location of y^m in the monomial order is $v\binom{m+1}{2} + m + 1$.*

Proof: Because of the reverse lexicographic ordering, y^m is the last monomial whose $(1, v)$-degree is mv. Hence a polynomial whose last monomial is y^m has C non-zero coefficients and C is given below.

$$\mathsf{C} = |(i, j) : i + vj \leq mv| = v\binom{m+1}{2} + m + 1$$

□

Now using Lemma 5.6 we get

$$\mathsf{C} = v\binom{m+1}{2} + m + 1 = k\binom{m+1}{2} - \frac{1}{2}(m+1)(m-2)$$

Substituting $k\binom{m+1}{2} = \rho\mathsf{C}_{wc} - \rho$:

$$\mathsf{C} = \rho\mathsf{C}_{wc} - \rho - \frac{1}{2}(m+1)(m-2)$$

$\frac{1}{2}(m+1)(m-2) \geq -1$ and so

$$\mathsf{C} \leq \rho\mathsf{C}_{wc} - \rho + 1 \leq \lceil \rho\mathsf{C}_{wc} \rceil$$

Since $\mathsf{C}_\epsilon(0) \leq \mathsf{C}$ the theorem follows. □

5.3.4 Tighter bounds for higher weight errors

When e is large, bounds on the interpolation cost can still be obtained, though using (5.2) may not be the best choice. For such cases we can use the following theorem.

Theorem 5.7 *Let E be an error vector of weight e and let $\{j_1, j_2, \ldots, j_e\}$ be the error locations. A polynomial of the form*

$$Q(x, y) = (y - f(x))^{m'} P(x, y)$$

is an interpolating polynomial when $P(x, y)$ satisfies $e\binom{m+1}{2} + (n-e)\binom{m-m'+1}{2}$ interpolation constraints.

Proof: We first find a minimal $(1, k-1)$-degree polynomial $P(x, y)$ that satisfies the following constraints. For the e corrupted locations $\{j_1, j_2, \ldots, j_e\}$, we require the usual interpolation constraints $D_{r,s}P(\alpha_{j_i}, R_{j_i}) = 0$ for every $(r, s) : r + s < m$. For the $n - e$ uncorrupted locations we require fewer such constraints: $D_{r,s}P(\alpha_j, R_j) = 0$ for every $(r, s) : r + s < m - m'$. Since for the corrupted symbols $P(x, y)$ alone satisfies all interpolation requirements, $Q(x, y)$ obviously does so too. As for the uncorrupted symbols we write

$$D_{r,s}Q(x, y) = \sum_{\substack{r_1 + r_2 = r \\ s_1 + s_2 = s}} D_{r_1,s_1}(y - f(x))^{m'} D_{r_2,s_2}P(x, y)$$

Splitting the sum to two disjoint intervals

$$D_{r,s}Q(x, y) = \sum_{\substack{r_1 + r_2 = r \\ s_1 + s_2 = s\ : \\ r_1 + s_1 < m'}} D_{r_1,s_1}(y - f(x))^{m'} D_{r_2,s_2}P(x, y) +$$

$$+ \sum_{\substack{r_1 + r_2 = r \\ s_1 + s_2 = s\ : \\ r_1 + s_1 \geq m'}} D_{r_1,s_1}(y - f(x))^{m'} D_{r_2,s_2}P(x, y) =$$

$$\sum_{\substack{r_1+r_2=r\\ s_1+s_2=s\ :\\ r_1+s_1<m'}} D_{r_1,s_1}(y-f(x))^{m'}D_{r_2,s_2}P(x,y)+\sum_{\substack{r_1+r_2=r\\ s_1+s_2=s\ :\\ r_2+s_2\le r+s-m'}} D_{r_1,s_1}(y-f(x))^{m'}D_{r_2,s_2}P(x,y)$$

The left sum is zero by Lemma 5.2 and the right sum is zero since $r+s-m' < m-m'$ and the $(r_2,s_2) : r_2+s_2 < m-m'$ Hasse derivatives of $P(x,y)$ vanish on the uncorrupted locations by construction. □

Corollary 5.8 *For any $0 \le m' \le m$, let $x^{d_x}y^{d_y}$ be the monomial whose index in the monomial order is $e\binom{m+1}{2}+(n-e)\binom{m-m'+1}{2}$ and define $\Delta' = d_x+(k-1)(m'+d_y)$, $r' = \Delta' \bmod (k-1)$. Then the interpolation cost is bounded by*

$$\mathsf{C}_e(\epsilon) \le \frac{(\Delta')^2}{2(k-1)}+\frac{\Delta'}{2}+\frac{r'(k-r'-1)}{2(k-1)}+m'+d_y+1 \tag{5.3}$$

Proof: $x^{d_x}y^{d_y}$ is the last monomial of the polynomial $P(x,y)$ used in Theorem 5.7. The last monomial of $Q(x,y)$ is $x^{d_x}y^{m'+d_y}$, and its $(1,k-1)$-weighted degree is $d_x+(k-1)(m'+d_y)=\Delta'$. Now finding the interpolation cost of $Q(x,y)$ is a matter of calculating the index of $x^{d_x}y^{m'+d_y}$ in the monomial order, in the same way that has been done in Corollary 5.4. □

Notes:

(1) Theorem 5.3 is a special case of Theorem 5.7 with $m'=m$ and $P(x,y)$ univariate in x. In general, m' can be freely chosen to find the best bound on the interpolation cost $\mathsf{C}_e(\epsilon)$ for each error weight e.

(2) The more general bound of (5.3) is not given in closed form since calculating d_x and d_y in closed form as functions of the monomial index in the order is not possible. Closed form upper bounds on d_x and d_y can be used instead, but the tightness of the bound would be compromised in this case.

(3) The power of the composition of $Q(x,y)$ as a product of two polynomials seems to lie on the following fact. In the interesting cases $1 \le m' < m$, for each uncorrupted location the composition polynomial $(y-f(x))^{m'}P(x,y)$ satisfies more interpolation constraints relative to the sum of constraints satisfied by the individual components $(y-f(x))^{m'}$ and $P(x,y)$. $(y-f(x))^{m'}$ satisfies $\binom{m'+1}{2}$ constraints, $P(x,y)$ satisfies

$\binom{m-m'+1}{2}$ and as proved in Theorem 5.7, $Q(x, y)$ satisfies $\binom{m+1}{2}$. These numbers reflect a difference of $m'(m - m')$.

5.3.5 Interpolation costs for a sample RS code

In section 5.3 bounds are given for the error-weight dependent interpolation costs. Here we wish to explore the tightness of these bounds by interpolating received words induced by different error words and comparing the observed interpolation costs to the bounds above. For that task a GS decoder was implemented and run on a $[n, k] = [31, 15]$ RS code. The interpolation multiplicity chosen for the decoder is $m = 3$, which allows correcting 9 errors and has a worst case interpolation cost of $\mathsf{C}_{wc} = n\binom{m+1}{2} + 1 = 187$. The results are summarized in Table 5.1 below. Each row reflects a value of e and the columns compare observed results to the bounds. The columns tagged *observed* are the maximum, average and minimum interpolation costs used by the decoder. These numbers were generated using repeating runs ($\sim 10^5$ per e) with random errors. For $e \leq 6$, no interpolation costs smaller than the closed form bound of Corollary 5.4 were observed. For $e = 7$ the bound is attained in almost all instances, with few exceptions of up to a difference of 2. That is the case also for $e = 8$, only that Corollary 5.8 is used to find an improved bound over Corollary 5.4. For $e = 9$ the best upper bound for the interpolation cost is C_{wc}. The results of this experimental study are that the upper bounds on interpolation costs provided here are tight in the worst case (max values attain the bounds for all e), and close to tight even in the average case. Hence, at least for this sample decoder, the bounds provide a succinct and reliable characterization of the decoder behavior. Validating the upper bounds' tightness becomes a practical challenge for long codes with large interpolation multiplicities, and general analytical lower bounds seem hard to come by.

5.4 From Interpolation Cost to Decoding Complexity

In the preceding section it has been argued that in many cases the interpolation cost is significantly lower than the worst case C_{wc}. That immediately means factorization algorithms would run faster in low cost instances. However, the most computationally expensive part of the decoder is the interpolation algorithm. Unfortunately, a reduced interpolation cost

				observed		
#errors e	worst case C_{wc}	closed form (5.2)	improved (5.3)	max	average	min
0	187	88	-	88	88	88
1	187	100	-	100	100	100
2	187	112	-	112	112	112
3	187	124	-	124	124	124
4	187	136	-	136	136	136
5	187	149	-	149	149	149
6	187	164	-	164	164	164
7	187	179	-	179	178.95	177
8	187	194	183 , $m' = 1$	183	182.97	179
9	187	209	187 , $m' = 0$	187	186.93	184

Table 5.1: Interpolation costs for the $[31, 15]$ RS code with $m = 3$

does not automatically provide reduced running time of interpolation algorithms. Admittedly, we will see that accepted interpolation algorithms do not translate the savings in coefficients to savings in running time. That is true even in light of the fact that these algorithms do eventually output the lowest degree interpolation polynomials. This situation is unfortunate since the decoder fails to benefit from the worst-case/instantaneous-case gap that was pointed out earlier in the chapter. We examine such behaviors of two interpolation algorithms in the case of reduced interpolation cost. We subsequently suggest modifications to the interpolation algorithms to improve their average-case running time.

5.4.1 Gaussian elimination interpolation

By formulating the interpolation problem as a system of homogeneous linear equations, Gaussian elimination stands out as a natural straight forward algorithm to solve it. This interpolation method is not the most efficient that exists and we present it only to illustrate the connection between interpolation cost and running time. A naive way to use Gaussian elimination is to start with a $(\mathsf{C}_{wc} - 1) \times \mathsf{C}_{wc}$ matrix and perform full Gaussian elimination. The number of rows being the number of interpolation constraints and the number of columns is the worst case interpolation cost. Since that matrix is under-determined, at termination we are guaranteed to reveal linearly dependent columns which result in coefficients of an interpolating polynomial. To analyze the running time of the above procedure, we will approximate the dimensions of the matrix by $\mathsf{C}_{wc} \times \mathsf{C}_{wc}$. It is well known that the running time of Gaussian elimination on a $c \times c$ matrix approaches $\frac{2}{3}c^3$ finite field operations (plus lower order terms $o(c^3)$) [TB97, ch. IV]. This follows from

$$2 \sum_{k=1}^{c} \sum_{j=k+1}^{c} (c-k+1) \rightarrow \frac{2}{3}c^3$$

Thus using straightforward Gaussian elimination would consume $\frac{2}{3}\mathsf{C}_{wc}^3$ finite field operations, regardless of the actual interpolation cost of the decoding instance. By using a simple variation on that process we can save considerably in the total number of field operations. When a shorter interpolation polynomial exists, some of the columns in the matrix will not participate in the interpolation. Exploiting that, a row operation should be performed on a column index, only if the columns to its left are linearly independent. This replaces the row operation on the full row vector performed in Gaussian elimination. An even more obvious modification is stopping the process at the first time linearly dependent columns are revealed. If we denote $c = \mathsf{C}_{wc}$, $c' = \mathsf{C}_e(\epsilon)$ and $\gamma = \frac{c'}{c}$, then the running time of the modified Gaussian elimination will be

$$2 \sum_{k=1}^{c'} \sum_{j=k+1}^{c} (c'-k+1) \rightarrow \frac{2}{3}cc'^2 + \frac{1}{3}c'^2(c-c') =$$

$$= \frac{2}{3}\mathsf{C}_{wc}^3 \left(\frac{3}{2}\gamma^2 - \frac{1}{2}\gamma^3\right)$$

and that yields a $\frac{3}{2}\gamma^2 - \frac{1}{2}\gamma^3$ factor of saving.

5.4.2 The standard interpolation algorithm

Now we wish, for the same purpose of average-case analysis, to consider the standard, most efficient interpolation algorithm used in RS list decoding. This algorithm and its variants are intensively studied in the literature [Köt96],[NH00], [KV03b], [AKS04] and more. Its mathematical richness notwithstanding, only a rough sketch of the algorithm is presented here, to focus on the computational issue at hand. The key idea of the algorithm is to interpolate $L+1$ polynomials, each with a different y-degree, and upon termination select the one with the lowest interpolation cost. By fixing the y-degrees throughout the update process, a “greedy” update rule cumulatively satisfies all interpolation constraints, and is guaranteed to output minimal polynomials for each y-degree. The algorithm pseudo-code description given in Figure 5.2, refers to this greedy update rule that successively eliminates discrepancies with respect to all interpolation constraints. The *argmin* operator selects the index j of the polynomial Q_j whose highest monomial has the lowest index with respect to the standard monomial ordering. The non-zero discrepancy of that lowest degree polynomial is used to eliminate the discrepancies of other higher degree polynomials.

```
**Initialize**
$Q_j := y^j,\ \forall j \in \{0, \ldots, L\}$      // L is a bound on the $y$-degree of the output polynomial
**for** $i := 1$ **to** $\mathsf{C}_{wc} - 1$      // interpolation constraints
(1)  $\delta_j^{(i)}$ := discrepancy of $Q_j$ with respect to constraint $i$
     $j^* := \mathrm{argmin}(j : \delta_j^{(i)} \neq 0)$
     **forall** $j$ with non-zero $\delta_j^{(i)}$
(2)      for $j \neq j^*$ update $Q_j$ with no change in degree
         for $j^*$ update $Q_{j^*}$ with degree increment
**output** $Q_j$ with minimal degree
```

Figure 5.2: Standard interpolation algorithm (sketch)

Analyzing the complexity of the algorithm, it iterates on $\mathsf{C}_{wc} - 1$ constraints and in each iteration performs operations *(1)* and *(2)* on (at most) $L + 1$ polynomials, each with no more than C_{wc} coefficients. Therefore the worst case running time is $L\mathsf{C}_{wc}^2$ finite field operations.

We next observe that the running time will not be significantly better in cases when the final interpolation cost is small. The reason being that the computation load is dominated by operations on non-minimal polynomials. Even if a polynomial Q_{j^*} satisfies all interpolation constraints with low cost, the algorithm does not know the identity of that j^* in advance and has to successively update all polynomials Q_j that have higher costs. It also does not a priori know the final required cost and thus cannot exclude polynomials with higher costs during computation. Consequently, this fast interpolation algorithm will have an average case running time not better than that of the worst case.

To fix that undesirable behavior, we modify the algorithm in a way that discrepancy calculations and polynomial updates are performed only on polynomials whose coefficient counts are guaranteed to be at most the final interpolation cost. This can be done by modifying the algorithm iteration, with no increase in worst-case running time. Storage complexity is higher, as older versions of polynomials Q_j are needed for updates during runtime. One can think of the modified algorithm as a relayed version of the standard algorithm where each time the leading candidate (the minimal degree polynomial) is sequentially updated, until a better candidate is found. On its way, before it is updated with degree increase, the best candidate stores its coefficients and discrepancies to allow for future candidates to ”catch up” with their updates. In Figure 5.3 the modified algorithm is presented.

```
Initialize
$Q_j := y^j,\ \forall j \in \{0, \ldots, L\}$
$i_j := 0,\ \forall j \in \{0, \ldots, L\}$      // constraint pointer for each $j$
$j^* := 0$      // $j^* = \text{argmin}_j Q_j$
while $i_{j^*} < \mathsf{C}_{wc} - 1$   // while no $Q_j$ satisfies all constraints
  $i_{j^*}++$
  find $\delta_{j^*}^{(i_{j^*})}$
   if $\delta_{j^*}^{(i_{j^*})} = 0$ continue
(*) mem-lookup $(\Delta[j, i_{j^*}]\ ,\ \mathcal{Q}[j, i_{j^*}])$  // look for stored poly
   if (found) update $Q_{j^*}$ with no change in degree
   else
      store $(\Delta[j^*, i_{j^*}]\ ,\ \mathcal{Q}[j^*, i_{j^*}]) \leftarrow (\delta_{j^*}^{(i_{j^*})}\ ,\ Q_{j^*})$
      update $Q_{j^*}$ with degree increment
       $j^* := \text{argmin}_j Q_j$  // proceed with the best candidate

output $Q_{j^*}$
```

Figure 5.3: Interpolation algorithm with improved average running time

mem-lookup in *(*)* refers to the action of looking up a stored polynomial Q_j that had a non-zero discrepancy on i_{j^*}. The following facts facilitate the correctness of the algorithm and its complexity.

(1) Discrepancy calculations and polynomial updates are performed only on polynomials with degrees lower than or equal to the final interpolation polynomial.

(2) The first Q_j whose pointer i_j reaches an index i is the lowest degree polynomial

that satisfies constraints $1, \ldots, i-1$. Therefore, the stored polynomials will always be the lowest degree polynomials that satisfy $1, \ldots, i-1$ but not i.

(3) If mem-lookup fails for Q_j on constraint i_j, it is equivalent to Q_j being the lowest degree polynomial with non-zero discrepancy on i_j.

(4) The polynomial whose pointer i_j first reaches C_{wc} satisfies all constraints and is the minimal to achieve that.

For every constraint with non-zero discrepancy, at most one polynomial is stored and each of these has at most $\mathsf{C}_e(\epsilon)$ coefficients. Thus in this non-optimized formulation, the amount of memory required for coefficient storage is bounded by $(\gamma \mathsf{C}_{wc})^2$. The time complexity of the algorithm is $\gamma \mathsf{C}_{wc}^2$, since for each constraint, discrepancy calculation evaluates a polynomial with at most $\gamma \mathsf{C}_{wc}$ coefficients.

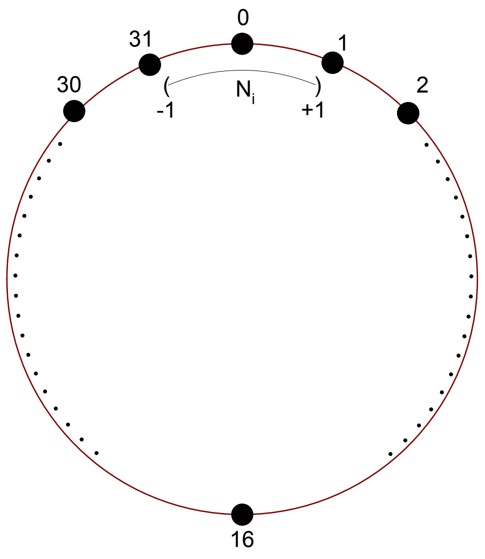


Figure 5.4: Channel model for soft-decision decoding. One of 32 symbols is transmitted and corrupted by a noise process N_i with bounded support $(-1, 1)$. The small support ensures a very simple case of soft-decision decoding where only two symbols are assigned non-zero interpolation multiplicities.

5.5 Interpolation Cost in Soft-Decision Decoders

The bounds presented thus far apply to GS decoders which have a fixed interpolation multiplicity m. They do not apply to the weighted interpolation used by Kötter and Vardy's

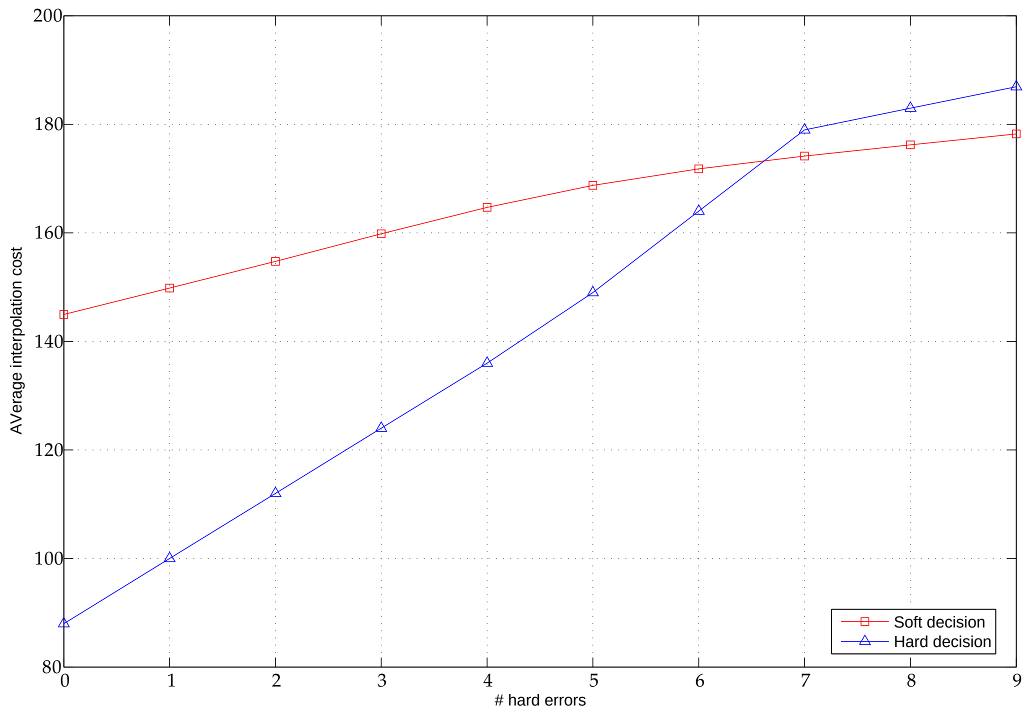


Figure 5.5: SD-HD comparison, average interpolation cost. Low error weights reduce the interpolation cost in hard-decision decoding and much less so in soft-decision decoding. For the same-worst case complexity the soft-decision decoder has higher average-case complexity.

soft decision decoder that was shown to correct more errors when soft inputs are available. In this section we examine another aspect of algebraic soft-decision list-decoders: their error-dependent interpolation costs. Since soft decision decoders surrender their fixed multiplicity property, none of the bounds above apply to them. Moreover, when the decoder inputs are soft symbols, different ways exist to define the instantaneous channel error upon which the interpolation cost may depend. The difficulty of analytic treatment of the soft decision case arises from the fact that the interpolation cost depends on the interpolation multiplicities which in turn depend on the channel error in a non-simple fashion. The bounds obtained for the hard decision case used the structure of the interpolation polynomial endowed by the fixed multiplicity m. It is therefore conjectured that the soft decision decoder will not enjoy as favorable interpolation cost behavior, and consequently will have higher average case decoding complexity, even if it is designed for the same worst case cost

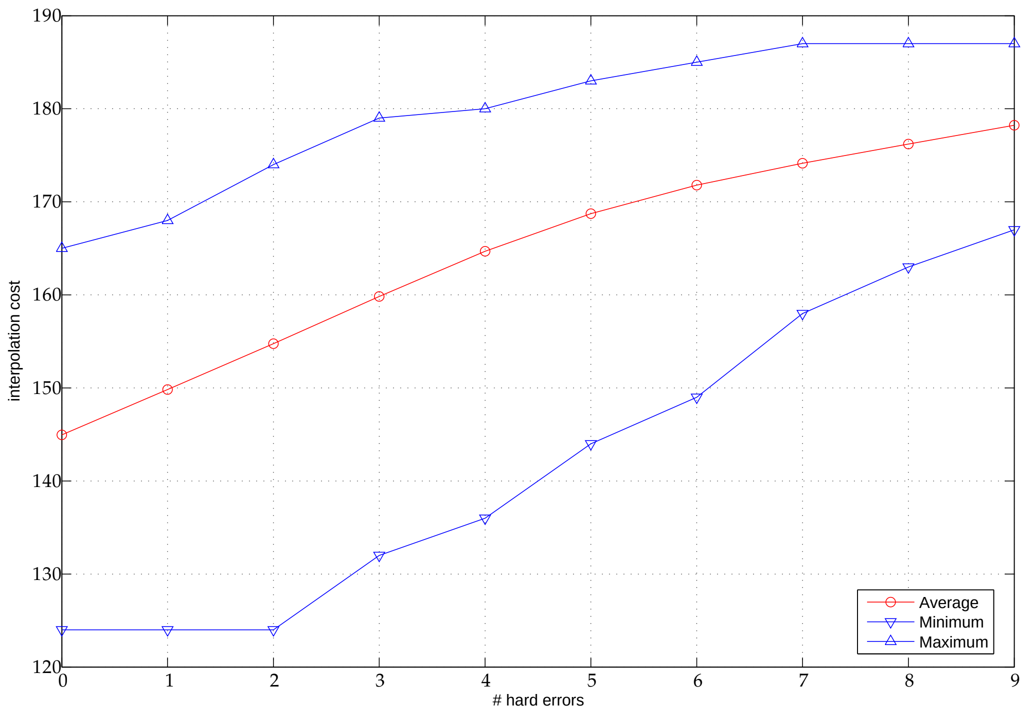


Figure 5.6: Interpolation cost of SD decoder, MAX, MIN and AVERAGE. Contrary to hard-decision decoding, soft-decision decoding exhibits high variability of the interpolation cost given an error weight.

as the hard decision decoder. To support that conjecture we veer to the experimental realm.

5.5.1 Simulation results for soft-decision decoding

For the $[31, 15]$ RS code of section 5.3.5, we simulated soft decoding over a channel whose description follows. We regard the 32 alphabet symbols as integers lying on a ring of circumference 32 (see Figure 5.4). The noise is taken to be an additive (modulo 32) i.i.d random process, denoted $N = \{N_1, \ldots, N_n\}$. For simplicity we take the probability density function of N_i to have a bounded support $(-1, 1)$. This property implies that at most two symbols will be assigned non-zero interpolation multiplicities by the Kötter-Vardy algorithm. The decoder we used has a worst case interpolation cost identical to that of the hard decision decoder we used in section 5.3.5: $\mathtt{C}_{wc} = 187$. It is thus interesting to compare the instantaneous interpolation costs exhibited by the soft-decision decoder to that of the hard-decision decoder. To have a ground for comparison, we plot the interpolation cost as a function of the number of ”hard” errors e caused by the channel. This number can be recovered by $e = |\{N_i : |N_i| > 0.5\}|$. In Figure 5.5 the hard decision (HD) and soft decision (SD) average interpolation costs are plotted as a function of the number of errors. Each point on the graphs was obtained from an order of 10^5 runs. We see that for low error weights the SD decoder requires higher interpolation costs compared to the HD decoder. For high weights SD is more efficient but only slightly. Another difference can be seen in Figure 5.6. While Table 5.1 shows the low variability of the cost for the HD decoder, Figure 5.6 shows that this is not the case for its SD counterpart. Both the relative flatness in Figure 5.5 and the variability across runs in Figure 5.6 indicate that in SD decoding, the dependence of the interpolation costs on the error weight is rather weak, contrary to the HD case. Once decoder running times depend on the instantaneous interpolation costs and not merely on the worst-case, the average decoding time of SD decoders may be higher than HD, even if they have identical worst-case running times.

5.6 Miscorrection Probability of List Decoders

When the number of symbol errors within a code block is large, decoding can go wrong in two different ways. The first, called *decoding failure*, is when the decoder cannot correct the errors and thus declares failure without providing any hypothesis on the transmitted codeword (detected error). The second, and more detrimental outcome, is called *miscorrection*, that happens when the decoder outputs a wrong codeword as its hypothesis on the transmitted codeword (undetected error). Three possible decoding outcomes are illustrated in Figure 5.7. The circle marked C_0 represents the transmitted codeword, the square E is the received word, and $\bar{C}$ is a different codeword (not the transmitted one) found in the decoding ball.

Miscorrections cannot happen when t, the decoding radius of the decoder (the radius of the Hamming ball around the received word to which decoder outputs are limited), and e, the Hamming weight of the error, satisfy $t + e < d$, where d is the minimum Hamming distance of the code. In the common case of decoders with maximal decoding radius under

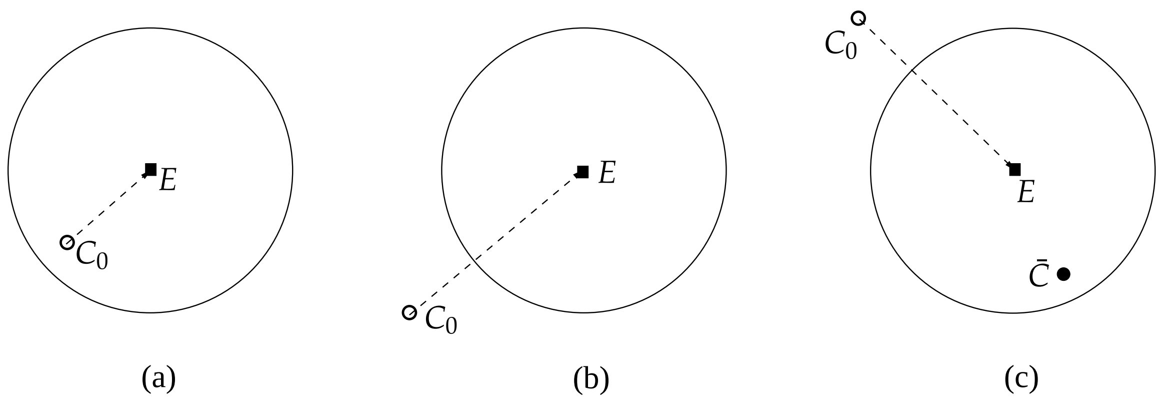


Figure 5.7: Possible decoding outcomes. (a) Successful decoding. (b) Decoding failure. (c) Miscorrection.

unique decoding ($2t + 1 = d$), any error of weight $e > t$ *may* cause miscorrection, and obviously any error of weight $e \leq t$ is successfully corrected. In the case of unique decoding, when the code is linear and the channel is symmetric, the probability of miscorrection can be calculated *exactly* if the weight distribution of the code is known [HM77]. This has independently been shown for the special case of linear MDS codes in [Che92].

For a given error weight e, all error words of that weight are partitioned into *decodable words*, error words that result in a miscorrection, and *non-decodable words*, error words that are either successfully decoded or result in decoding failure. Finding the miscorrection probability is thus reduced to counting the number of decodable error words of weight e and dividing this number by the total number of error words of the same weight. The notion of decodable words is best described graphically. In Figure 5.8, a bipartite graph is shown whose left nodes are all the codewords, excluding the all zero codeword, and right nodes are error words of some weight e. An edge connects a codeword and an error word if their Hamming distance is t (the decoding radius) or less. If the code is linear, one can assume that the all zero codeword was transmitted. In that case, it is readily seen that error words E on the right that have at least one incident edge, are decodable words that result in miscorrection. This is true since the decoder will output as hypotheses the non-zero codewords connected to the error word. When $2t + 1 \leq d$, an error word can have at most one incident edge, otherwise it would imply two codewords that are in distance less than d apart. In that case counting the *edges* of the graph is equivalent to counting decodable words; and indeed, it is observed that the counting methods of both [HM77] and [Che92] count the exact number of edges in that graph.

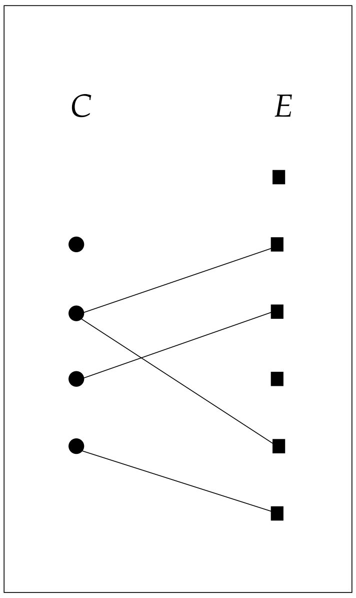


Figure 5.8: Decodability graph under unique decoding. A right error-word node is connected to a left codeword node if they are at Hamming distance t or less from each other. Unique decoding guarantees that the degree of any error-word node is at most one.

When the decoding radius is stretched beyond unique decoding ($2t \geq d$), error words may have multiple incident edges, and counting edges becomes only an upper bound on the number of decodable words (a similar observation was made in [McE03b]). A graphical description of this scenario is shown in Figure 5.9: two of the received words in the graph, marked with dashed circles, have multiple incident edges and are therefore multiply counted as decodable words, resulting in an overestimate on the miscorrection probability.

The main observation of this section, made simple by the chosen graphical description, is that a *lower* bound on the number of decodable words can be obtained using an *upper* bound on the number of edges incident on error-word nodes. If the number of decodable words is denoted $D_t(e)$, the number of edges in the graph denoted $|\mathcal{E}_e|$, and M is an upper bound on the degree of E nodes, then

$$\frac{|\mathcal{E}_e|}{M} \leq D_t(e) \leq |\mathcal{E}_e|$$

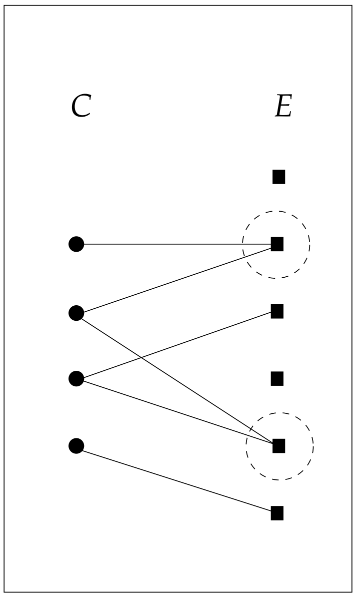


Figure 5.9: Decodability graph under list decoding. Some error-word nodes have degrees greater than one that leads to multiply-counted decodable error-words.

Good bounds M on the degree of E nodes, arguably useful in obtaining lower bounds on miscorrection, turn out to independently be a fundamental problem in the area of list decoding – that of bounding the size of the decoder's output list. For moderately large alphabets, the bound derived in the next section is the tightest known closed-form bound on the list size, and hence provides the best known lower bound on the miscorrection probability. Bounds on miscorrection probability of a sample list decoder are now compared. Results for a linear MDS code with parameters $n = 31, k = 15, d = 17, q = 32$ (q is the alphabet size of the code) are shown in Figure 5.10 for decoding radius of $t = 9$. The curves from top to bottom are: i) the upper bound $|\mathcal{E}_e|$ using the method from [HM77]. ii) improved lower bound $|\mathcal{E}_e|/M$, using M from the next section. iii) lower bound that counts the exact number of correctable words for decoder radius of $t_0 = (d-1)/2 = 8$. The true value of the miscorrection is proved to be between the two upper curves. The sample results reflect a 1.5 orders of magnitude improvement by the new bound compared to the (previously best known) bound that assumes decoding radius of an optimal unique decoder. Consequently, in spite of the constant factor gap between the new lower bound and the upper bound, the new lower bound does show that the miscorrection probability grows significantly when decoding beyond the unique decoding bound.

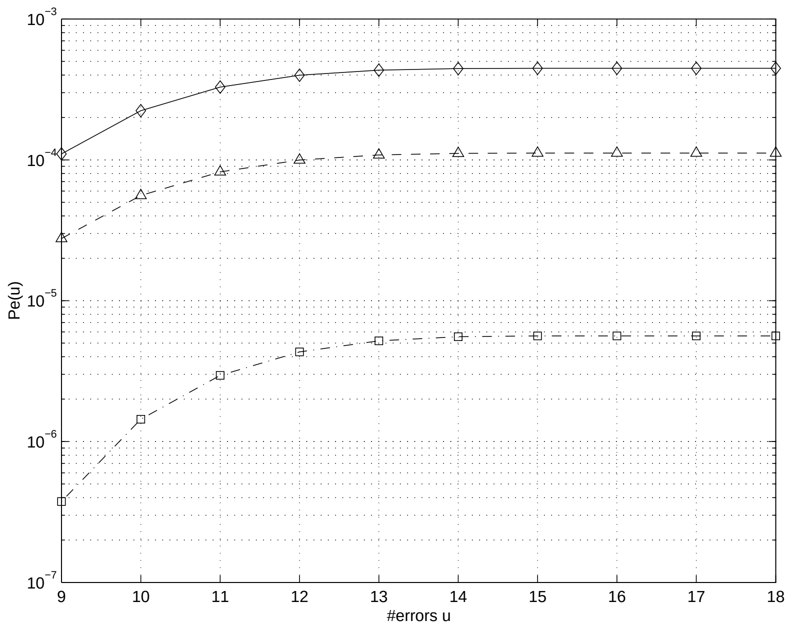


Figure 5.10: Bounds on the miscorrection probability for a [31,15] MDS code, decoded to radius 9. The solid curve is an upper bound by counting edges in the decodability graph. The lower dashed curve is a lower bound that counts decodable words only within the unique-decoding sphere. The upper dashed curve is the new lower bound.

5.7 A Combinatorial Bound on the Decoder's Output-List Size

For a decoding instance of a decoder whose radius is greater than half the code's minimum distance, unique decoding is not guaranteed and the decoder outputs, in general, a list of codewords that fall in the decoding Hamming ball. Bounding from above the size of this codeword list is a well studied problem with both theoretical and practical appeal. Codes that have short lists for relatively large decoding radii are termed *list-decodable*, and their design is of prime interest. In this section we focus on cases where the worst-case list-size is bounded by a constant number (independent of n), and try to find the smallest of such constant bounds. From a theoretical standpoint, a list size that is at most polynomial in the code length is a necessary condition for having a polynomial-time list-decoding algorithm. For practical usage of list decoders however, very small constant-size lists are desirable, to minimize the information uncertainty at the decoder output.

Though list-decodability of a code does not necessarily imply good minimum distance, the minimum distance does ensure a certain degree of list-decodability. The bound derived in this section, as those that predate it, uses the minimum distance of a code to bound the list size of a radius t decoder for that code. Hence it applies to *any* code with a given minimum distance. The reason we can bound the number of codewords in a radius t Hamming ball, just based on the minimum distance is rather obvious; packing many codewords in a small ball is impossible when every pair of codewords should be at least d apart. As mentioned in other works that deal with the worst case list size, most notably [Eli91] and [Gur01, Ch.3], this problem is closely related to the problem of bounding the maximal size of a constant-weight code. Accordingly, Johnson bound based arguments [MS77, pp.525], with necessary modifications, prove effective for the list-size problem. In [Eli91], the Johnson bound is shown to provide a valid list-size bound in the binary case. The q-ary case was addressed in [GRS00], though the main bound there can be extracted from the q-ary generalization of the Johnson bound, and a simple argument on its applicability to the list-size problem (see for example 2.3.1 of [HP03] for the q-ary Johnson bound). An improvement over [GRS00] for short codes was reported in [GS01] using a geometric approach. The bound presented here is better than its predecessors when the alphabet size q is “large enough”. The threshold alphabet size for the bound to be tighter depends solely on the ratios d/n and t/n. Therefore, for code families such that their alphabet grows with their length (e.g. Reed-Solomon), this bound will be asymptotically tighter. For sample codes, the new bound is compared to the best known bound and are found to offer improvement even for relatively short codes. This encouraging behavior of the bound is further validated by showing that the bound is very close, at most a small constant away, to the algebraic bound of [McE03a] for RS codes, despite being simpler and more general - thus proving that RS codes are not significantly more list-decodable than any other code with the same (n, d, t) parameters. We note that in [RR03], the authors proved a similar conclusion that the Guruswami-Sudan algebraic bound on the decoding radius of a list decoder applies to a general block code. Nevertheless, this result yields no closed form expression for the list size. Such a simple closed form expression is often required to analyze the behavior of the code, as was done in the previous section with lower bounds on the miscorrection probability. Moreover, the bound presented here is more general and may be further tightened using non-Johnson based techniques.

5.7.1 Derivation of the bound

For a code with length n and minimum distance d we want to bound the number of codewords that can reside in an arbitrary Hamming ball of radius t. Similarly to Elias [Eli91] and Goldreich et al [GRS00], our analysis is combinatorial and thus applies to a general code. In distinction from those known bounds, we give a bound which is independent of the alphabet size of the code.

Theorem 5.9 *Let $\mathcal{C}^*$ be a length-n code over any alphabet of size q and with minimum Hamming distance d. Let $M(E)$ be the number of codewords in Hamming distance at most t from a particular word E: $M(E) = |\{C \in \mathcal{C}^* : D(C,E) \leq t\}|$. Then if $\lfloor (d-1)/2 \rfloor < t < d$ we have*

$$\boxed{M(E) \leq A_2(n, 2(d-t), t)} \tag{5.4}$$

where $A_2(n, 2(d-t), t)$ is the size of the largest $\underline{\text{binary}}$ constant-weight code with weight t and minimum distance $2(d-t)$.

Proof: We first consider the maximal number of codewords on the *surface* of the t ball. Let $M'(E) = |\{C \in \mathcal{C}^* : D(C,E) = t\}|$. We fix E and define $M' = M'(E)$. For any pair of codewords C_i, C_j that are both in distance t from E, we have $d \leq D(C_i, C_j) \leq 2t$. We use $X(l)$ to denote the symbol on the l^{th} coordinate of X. Then we define a pair of binary vectors K_i, K_j to be $K_i(l) = 1$ if $C_i(l) \neq E(l)$ and 0 otherwise, similarly for K_j. Then $|\{l : K_j(l) = 1\}| = |\{m : K_i(m) = 1\}| = t$. We define the *span* of two binary vectors as the number of coordinates that are 1 in at least one of the vectors

$$\text{span}(X_i, X_j) \equiv |\{l : X_i(l) = 1\} \cup \{m : X_j(m) = 1\}|$$

We claim that

$$\text{span}(K_i, K_j) \geq d$$

Otherwise there were more than $n-d$ coordinates in which $C_i(l) = E(l) = C_j(l)$, which would contradict the distance requirement $D(C_i, C_j) \geq d$. So a necessary condition to find M' codewords in distance t from E is the existence of M' binary vectors of weight t such that each pair (i, j), $i \neq j$, has $\mathtt{span}(K_i, K_j) \geq d$ (note that this condition is not sufficient since two codewords may have $C_i(l) \neq E(l)$, $C_j(l) \neq E(l)$ but $C_i(l) = C_j(l)$ - thus violating the minimum distance requirement). Therefore, since no two codewords can have the same weight-t binary vector as their K vector[2], an upper bound on the number of such binary K vectors will be an upper bound on M' for any alphabet size. The weight and span requirements together imply that the ones of K_i and K_j are allowed to overlap on at most $2t-d$ coordinates: $|\{l : K_i(l) = 1\} \cap \{m : K_j(m) = 1\}| \leq 2t - d$. This key fact allows bounding M' from above using bounds on the size of *binary* constant-weight codes with minimum distance $2(t-(2t-d)) = 2(d-t)$:

$$M' \leq A_2(n, 2(d-t), t) \tag{5.5}$$

To complete the proof we want to show that the upper bound on M' is also an upper bound on M. We define $W(X)$ to be the Hamming weight of X and claim the following. If we have M binary words such that every pair (K_i, K_j) taken from them satisfies

(G1) $W(K_i), W(K_j) \leq t$

(G2) $\text{span}(K_i, K_j) \geq d$

then there exist M binary words such that any pair satisfies

(P1) $W(K_i), W(K_j) = t$

(P2) $\text{span}(K_i, K_j) \geq d$

This implication is established by the following two arguments. First, increasing the weight of K_i or K_j by changing arbitrary $t - W(K)$ zeros to ones cannot decrease the span. Second, such modification cannot result in having two identical K vectors and so the number of distinct vectors is preserved in the process. The second, more subtle, argument is resolved by observing that $t < d$ and the non-decrease of the span imply that having two identical K vectors with weight t violates $\mathtt{span}(K_i, K_j) \geq d$, in contradiction with (G2) above. □

[2]such a pair would have $\mathtt{span}(K, K) = t < d$

Note that if the alphabet of $\mathcal{C}^*$ is binary ($q = 2$), the inequality (5.4) is useless since the condition $d < 2t + 1$ implies $M' \leq A_2(n, d, t) \leq A_2(n, 2(d - t), t)$. However, as shown in the next sub-section, even for relatively small alphabet sizes, the bound on $A_2(n, 2(d - t), t)$ in the next Corollary turns out to be the best known bound on the list size M.

Corollary 5.10 *Let $\mathcal{C}^*$ be a length-n code over any alphabet of size q and with minimum Hamming distance d. Then if $\lfloor (d-1)/2 \rfloor < t < n\left(1 - \sqrt{1 - d/n}\right)$ we have*

$$M(E) \leq \frac{n(d - t)}{t^2 - 2nt + dn} \tag{5.6}$$

Proof: To use Theorem 5.9, we first prove that

$$t < n\left(1 - \sqrt{1 - d/n}\right) \Rightarrow t < d$$

This can be done by simple manipulation as follows.

$$n\left(1 - \sqrt{1 - d/n}\right) = n - \sqrt{n(n - d)} \leq n - (n - d) = d$$

Now, re-deriving the classical binary Johnson bound [MS77, Ch.17] for the parameters of (5.4) we get

$$\frac{t^2 M^2}{n} - tM \leq (2t - d)M(M - 1)$$

$$\left(\frac{t^2}{n} - 2t + d\right) M \left[M - \frac{d - t}{\frac{t^2}{n} - 2t + d}\right] \leq 0$$

Solving for M, we get

$$M \leq \frac{n(d - t)}{t^2 - 2nt + dn}$$

under the condition

$$\frac{t^2}{n} - 2t + d > 0 \tag{5.7}$$

Solving for the condition (5.7)

$$t < n\left(1 - \sqrt{1 - d/n}\right) \tag{5.8}$$

□

We next turn to analyze the proposed bound (5.6). In sub-section 5.7.2, we evaluate it in comparison to the best known closed-form combinatorial bounds, and give exact threshold on the alphabet size, above which it is tighter than the previously best known. In sub-section 5.7.3, we explore the strong link the bound has to the seemingly unrelated algebraic bound for Reed-Solomon codes.

5.7.2 Comparison with known combinatorial bounds

A possible justification for a q-independent bound arises from the following. Ignoring the alphabet size in the proof of Theorem 5.9 required us to count the overlapping coordinates towards d, which is less restrictive (and thus result in a looser bound) than the Johnson bound in the binary case. However, if the alphabet size is large ”enough”, overlapping symbols are most likely to be different anyway, and the span requirement will capture the limitation on the number of codewords in the ball. As it turns out, this simplification proves advantageous for giving strictly tighter bounds for alphabets above some threshold.
To simplify the analysis we fix the relative distance by $\gamma = 1 - d/n$ and the decoding radius by $\delta = 1 - t/n$. Now the bound (5.6) is rewritten as

$$M \le \frac{\delta - \gamma}{\delta^2 - \gamma} \tag{5.9}$$

Henceforth we denote the bound in the right hand side of (5.9) by M_C. For nontrivial codes we require $0 < \gamma < 1$ and for δ we require $\sqrt{\gamma} < \delta < \frac{1+\gamma}{2}$. The lower limit is to maintain positive denominator in (5.9) and the upper limit represents decoding beyond half the minimum distance. The main bound of [GRS00, Thm 4.2], which, to the best of our knowledge, is the tightest known, asserts

$$M \leq \frac{(1-\gamma)(1-\frac{1}{q})}{(\delta-\frac{1}{q})^2-(1-\frac{1}{q})(\gamma-\frac{1}{q})} \tag{5.10}$$

which for large q tends to $\frac{1-\gamma}{\delta^2-\gamma}$, a value larger than (5.9) since $\delta < 1$. The exact alphabet size q_0, above which (5.9) is tighter than (5.10) can be recovered, as a function of γ, δ, by solving the following inequality for q

$$\frac{(1-\gamma)(1-\frac{1}{q})}{(\delta-\frac{1}{q})^2-(1-\frac{1}{q})(\gamma-\frac{1}{q})} > \frac{\delta-\gamma}{\delta^2-\gamma}$$

The above simplifies to a linear inequality and yields the threshold

$$q > q_0(\gamma,\delta) = \frac{\delta(1+\gamma)-2\gamma}{\delta^2-\gamma}$$

Hence we proved the following proposition.

Proposition 5.11 *For $\delta > \sqrt{\gamma}$ and $q > q_0(\gamma,\delta)$, the bound $M_C = (\delta-\gamma)/(\delta^2-\gamma)$ is the best known closed-form upper bound on the list size of a general code with relative distance $1-\gamma$, decoded to relative radius $1-\delta$.*

Table 5.2 shows a comparison of the bounds for sample codes. The rightmost column is the q_0 found above for the corresponding parameters n, d, t.

$(n,d,t),q$	(5.6)	[GRS00]	q_0
$(31,17,9),32$	4	10	2
$(31,17,10),32$	31	51	11
$(255,33,17),256$	120	239	9
$(18,17,13),19$	10	18	8

Table 5.2: Bound comparison for sample decoders

We note that the numbers in column (5.6) of the table do not improve over values computed by the *non-closed-form* bound of [RR03]. However, the general inequality $M \leq A_2(n, 2(d-t), t)$ allows to use stronger bounds on binary constant-weight codes to potentially improve over the particular Johnson-technique used in both Corollary 5.10

and [RR03]. Such tighter bounds do exist for specific parameters or families of parameters (e.g. the Erdös-Hanani exact evaluation of $A_2(n, 2t-2, t)$ [MS77, Ch.17]). It should also be noted that the upper bound on the list size given in (5.10) is identical to the q-ary Johnson bound for constant-weight codes, hence the new upper bound (5.9) is tighter than the q-ary Johnson bound as a bound on constant-weight codes.

5.7.3 Comparison with algebraic bound for Reed-Solomon codes

The decoding radii for which the proposed bound applies are those that satisfy (5.8). For Reed-Solomon codes that implies

$$t < n - \sqrt{(k-1)n},$$

which equals exactly the famous Guruswami-Sudan bound for decoding Reed-Solomon codes efficiently using the GS algorithm [GS99]. This coincidence of domains between the bounds allows us to set forth a comparison between the general combinatorial list-size bound, and the Reed-Solomon specific algebraic bound.

Algebraic list-size Bound

In [McE03a] McEliece provides a two step, closed form list-size bound, derived from arguments on maximal degrees of bivariate polynomials. The first step is determining the minimum interpolation multiplicity[3] required to achieve decoding radius of t

$$m > (k-1) \cdot \frac{t + \sqrt{n(2t+k-1-n)}}{2((n-t)^2 - (k-1)n)} = \gamma \frac{1 - \delta + \sqrt{\gamma - 2\delta + 1}}{2(\delta^2 - \gamma)} \tag{5.11}$$

The second step uses a list-size bound M_A that is given as a function of the multiplicity m.

$$M_A \approx \left(m + \frac{1}{2}\right) \sqrt{\frac{n}{k-1}} \tag{5.12}$$

$\approx$ here means that the right hand side is less than 1 greater than the true value of the bound (this notation was chosen over the usage of $\lceil \cdot \rceil$ to obtain cleaner expressions). Substituting

[3] In [McE03a] t is bounded given m so the expression here is the corresponding bound on m given t.

m from (5.11) into (5.12) we get

$$M_A \approx \frac{\sqrt{\gamma}}{2} \cdot \frac{1 - \delta + \sqrt{1 - 2\delta + \gamma} + \frac{1}{\gamma}(\delta^2 - \gamma)}{\delta^2 - \gamma}$$

So far we have a combinatorial bound $M_C = \frac{\delta - \gamma}{\delta^2 - \gamma}$ and an algebraic bound M_A above. We want to argue that M_C is close to M_A despite being more general. The following theorem shows that when approaching the strongest GS decoder (decoding radii that attain the GS bound) M_C and M_A converge to the same bound.

Theorem 5.12 $\lim_{\delta \to \sqrt{\gamma}} \frac{M_A}{M_C} = 1$

Proof: Elementary substitution $\delta = \sqrt{\gamma}$ into $\frac{M_A}{M_C}$.

□

It is also possible to show that the difference $M_C - M_A$ is small for general γ, δ.

Theorem 5.13 *For every pair γ, δ the combinatorial and algebraic bounds on the list size satisfy*

$$M_C - M_A < \frac{1}{4}\left[1 + \frac{2}{1 - \sqrt{\gamma}}\right]$$

Proof: We first prove a simple lemma.

Lemma 5.14 *If $\sqrt{\gamma} < \delta < \frac{1+\gamma}{2}$ then*

$$\sqrt{1 - 2\delta + \gamma} > \frac{1 - 2\delta + \gamma}{1 - \sqrt{\gamma}} \tag{5.13}$$

Proof:

$$\left(\frac{1 - 2\delta + \gamma}{1 - \sqrt{\gamma}}\right)^2 - \left(\sqrt{1 - 2\delta + \gamma}\right)^2 =$$

$$= \frac{4\delta^2 - \delta(2\gamma + 2 + 4\sqrt{\gamma}) + 2\sqrt{\gamma}(1 + \gamma)}{(1 - \sqrt{\gamma})^2} =$$

$$= \frac{4\overbrace{(\delta - \sqrt{\gamma})}^{>0}\overbrace{(\delta - \frac{1+\gamma}{2})}^{<0}}{(1 - \sqrt{\gamma})^2} < 0$$

the lemma follows since both sides of (5.13) are positive so $x^2 - y^2 < 0 \Rightarrow x < y$. $\square$

We are now ready to prove the theorem

$$M_A + \frac{1}{4}\left[1 + \frac{2}{1-\sqrt{\gamma}}\right] - M_C =$$

$$= \frac{\sqrt{\gamma}}{2} \cdot \frac{1 - \delta + \sqrt{1 - 2\delta + \gamma} + \frac{1}{\gamma}(\delta^2 - \gamma)}{\delta^2 - \gamma} + \frac{3 - \sqrt{\gamma}}{4(1-\sqrt{\gamma})} - \frac{\delta - \gamma}{\delta^2 - \gamma} >$$

$$> \frac{\sqrt{\gamma}}{2} \cdot \frac{1 - \delta + \frac{1 - 2\delta + \gamma}{1 - \sqrt{\gamma}} + \frac{1}{\gamma}(\delta^2 - \gamma)}{\delta^2 - \gamma} + \frac{3 - \sqrt{\gamma}}{4(1-\sqrt{\gamma})} - \frac{\delta - \gamma}{\delta^2 - \gamma} =$$

$$= \frac{(\delta - \sqrt{\gamma})(2 + \sqrt{\gamma} - \gamma)}{4\sqrt{\gamma}(1-\sqrt{\gamma})(\delta + \sqrt{\gamma})} > 0$$

The first inequality follows from Lemma 5.14, the equality from straightforward rearrangements and the last inequality from the positivity of both the numerator and denominator for $0 < \gamma < 1$, $\sqrt{\gamma} < \delta < \frac{1+\gamma}{2}$. $\square$

Substituting sample values of γ we get the following corollary.

Corollary 5.15

(1) $M_C = M_A$ *for all* δ *when* $\gamma \leq 0.11$

(2) $M_C - M_A \leq 1$ *for all* δ *when* $\gamma \leq 0.51$

(3) $M_C - M_A \leq 4$ *for all* δ *when* $\gamma \leq 0.8$

(4) $M_C - M_A \leq 9$ *for all* δ *when* $\gamma \leq 0.9$

It is thus concluded that the list decodability of Reed-Solomon codes is not known to be significantly better than that of any other code with the same parameters (apart from the existence of a constructive way to list-decode them, of course).

5.8 Notes and Open Questions

Analytic treatment of soft-decision algebraic list decoders is hard in general. However, analyzing the interpolation costs of restricted soft-decision decoders (such as the one used here with only two non-zero multiplicities), can help improving their average running time.

There is still a large gap between the lower and upper bounds on the miscorrection probability, mainly because of the coarse bounding technique used here. It is an interesting open question whether finer arguments on the degrees of error words in the decodability graph can be used to obtain tighter lower bounds. It is plausible that using knowledge on the particular code for such degree arguments will improve over our current method that is general to any code of the given parameters.

For the problem of finding upper bounds on the codeword-list size, it is interesting to note that there exist either pure combinatorial bounds (for general codes), or pure algebraic bounds (for specific codes e.g. Reed-Solomon), but potential progress may come by combining combinatorial and algebraic arguments to obtain tighter bounds.

Chapter 6

Forward-Looking Summary

In this short chapter, the author will take a step back from his emotional attachment to the research results above, and will instead wear the hat of an unbiased (but positive) critique. The purpose of this process is to depart from the serial, application-driven mode of presentation, and try to sieve out and evaluate core concepts that are introduced throughout the thesis. Then, when a general concept is identified, the scope of the thesis suddenly looks quite limited, and projecting intriguing and farther-reaching research directions becomes simple and natural.

- In symmetric channels and error models, alphabet symbols are abstract objects that carry no geometric meaning. The key idea in the constructions of Chapter 2, is that when alphabet symbols *do* encompass structure, a powerful technique is to construct codes that in addition to constraints on the code block, use clever mappings between the code alphabet and lower alphabets that capture the geometry of the error model. Admittedly, the error model considered here is a relatively simple instantiation of this idea, so it is a wide and interesting research trajectory to extend this method to other error models that are motivated by other applications.

- Chapter 3 introduces a valuable new characterization of code-location sets, that is used to propose an error model called *Clustered erasures*. Even that specific error model is only addressed for the case of up to 4 erasures. A very interesting open question is then whether this useful new cluster-based characterization lends itself to a nice coding-theoretic treatment in a much broader scope.

Another important outcome of the results of Chapter 3 is that in some cases departing from the MDS requirement on the code has minimal negative impact on its correction capability and significant positive impact on its implementation complexity. MDS erasure codes are heavily used in many fields as an abstract object (often called "k out of n schemes"). In many of these other usages of MDS codes, the MDS property is too strong, and a refinement of the particular model requirements can similarly lead to algorithmic savings.

- The very regular structure of the new array codes of Chapter 4, and their low-density parity-check matrices, make them excellent candidates to be decoded using iterative message-passing decoders. However, such decoders would view these array codes as one-dimensional binary codes, not utilizing the structure of their column-based error model. To improve iterative decoding of array codes, a very promising research path follows the idea of augmenting the code graphs of array codes with auxiliary nodes that bias the decoder toward errors that fall into a small number of columns. It is conjectured that performance gain can be achieved even without assuming any knowledge on the distribution of errors.

- If we needed more evidence of the great structure of Reed-Solomon codes, Chapter 5 has added its small share: analytically characterizing interpolation costs given instantaneous error weights. Examining and improving Reed-Solomon decoders under low instantaneous error weights can be generalized to doing the same for different restrictions on the error vectors. The mathematical richness of Reed-Solomon codes suggests that this may be doable for other such restrictions as well.

- The list-size upper bound in the second part of Chapter 5 has the following remarkable property. Even though it does *n*ot take into account the code alphabet size q in its derivation, it gives strictly and significantly tighter bounds compared to the Johnson bound that is a function of q (and has a much more complicated expression because of that). That happens when q is above some relatively small threshold. The new bound thus proves that generalizing results from binary codes to q-ary codes in the obvious way does not necessarily give the best results. Another difficulty in generalizing binary results to higher alphabets is encountered in Conjecture 2.1 in Chapter 2, whose settlement to the affirmative would prove optimality of the t asymmetric ℓ-limited magnitude code construction for any ℓ and t. Both examples indicate that more attention and more clever techniques are needed toward improving our understanding of q-ary coding in general.

Bibliography

[AAK02] R. Ahlswede, H. Aydinian, and L.H. Khachatrian. Unidirectional error control codes and related combinatorial problems. In *proc. of the Eighth International Workshop on Algebraic and Combinatorial Coding Theory (ACCT-8), St. Petersburg, Russia (Extended version available at: http://arxiv.org/abs/cs/0607132)*, pages 6–9, 2002.

[AHU74] A. Aho, J. Hopcroft, and J. Ullman. *The design and analysis of computer algorithms*. Addison-Wesley, Reading, MA USA, 1974.

[AKS04] A. Ahmed, R. Kötter, and N. Shanbhag. VLSI architectures for soft-decision decoding of Reed-Solomon codes. In *International Conference on Communications*, pages 2584–2590. IEEE, June 2004.

[And73] B. Anderson. Finite topologies and Hamilton paths. *J. Combinatorial Theory (B)*, 14:87–93, 1973.

[BBBM95] M. Blaum, J. Brady, J. Bruck, and J. Menon. EVENODD: an efficient scheme for tolerating double disk failures in RAID architectures. *IEEE Transactions on Computers*, 44(2):192–202, 1995.

[BBV96] M. Blaum, J. Bruck, and A. Vardy. MDS array codes with independent parity symbols. *IEEE Transactions on Information Theory*, 42(2):529–542, 1996.

[BFvT98] M. Blaum, P. Farrell, and H. van Tilborg. Array codes. *Handbook of Coding Theory, V.S. Pless and W.C. Huffman*, pages 1855–1909, 1998.

[Bla83] R.E Blahut. *Theory and Practice of Error Control Codes*. Addison-Wesley, Reading, Massachusetts, 1983.

[BM85] E. Berlekamp and R. McEliece. Average-case optimized buffered decoders. In *The impact of processing techniques on communications*, pages 145–158, Martinus Nijhoff publishers, The Netherlands, 1985. NATO Advanced science institutes.

[Bor81] J.M Borden. Bounds and constructions for error correcting/detecting codes on the Z-channel. In *Proc. IEEE International Symposium on Information Theory*, pages 94–95, 1981.

[BR93] M. Blaum and R.M Roth. New array codes for multiple phased burst correction. *IEEE Transactions on Information Theory*, 39(1):66–77, 1993.

[BR99] M. Blaum and R.M Roth. On lowest density MDS codes. *IEEE Transactions on Information Theory*, 45(1):46–59, 1999.

[BSH05] A. Bandyopadhyay, G. Serrano, and P. Hasler. Programming analog computational memory elements to 0.2% accuracy over 3.5 decades using a predictive method. In *proc. of the IEEE International Symposium on Circuits and Systems*, pages 2148–2151, 2005.

[CB04] Y. Cassuto and J. Bruck. Miscorrection probability beyond the minimum distance. In *Proc. of the IEEE International Symposium on Info. Theory*, page 523, Chicago, Illinois, June 2004.

[CB05] Y. Cassuto and J. Bruck. On the average complexity of Reed-Solomon algebraic list decoders. In *Proc. of the 8th International Symposium on Communication Theory and Applications*, pages 30–35, Ambleside, UK, July 2005.

[CB06] Y. Cassuto and J. Bruck. Cyclic low density MDS array codes. In *Proc. of the IEEE International Symposium on Info. Theory*, pages 2794–2798, Seattle, Washington, July 2006.

[CB07] Y. Cassuto and J. Bruck. Cyclic lowest density MDS array codes. *IEEE Transactions on Information Theory*, submitted, August 2007.

[CEG$^+$04] P. Corbett, B. English, A. Goel, T. Grcanac, S. Kleiman, J. Leong, and S. Sankar. Row-diagonal parity for double disk failure correction. In *In Proceedings of the 3rd USENIX Conference on File and Storage Technologies*, San-Francisco CA, 2004.

[Che92] K.M Cheung. On the decoder error probability of block codes. *IEEE-Trans-Comm*, 40(5):857–859, May 1992.

[CLO97] D. Cox, J. Little, and D. O'Shea. *Ideals, Varieties, and Algorithms*. Springer, New York NY, 1997.

[CSBB07] Y. Cassuto, M. Schwartz, V. Bohossian, and J. Bruck. Codes for multi-level Flash memories: correcting asymmetric limited-magnitude errors. In *Proc. of the IEEE International Symposium on Info. Theory*, pages 1176–1180, Nice, France, June 2007.

[Eli57] P. Elias. List decoding for noisy channels. Technical report, Technical report, Research Laboratory of Electronics, MIT, 1957.

[Eli91] P. Elias. Error-correcting codes for list decoding. *IEEE Transactions on Information Theory*, 37(1):5–12, 1991.

[ER99] B. Eitan and A. Roy. Binary and multilevel Flash cells. *Flash Memories, P. Cappelletti, C. Golla, P. Olivo, E. Zanoni Eds. Kluwer*, pages 91–152, 1999.

[GCKT03] S. Gregori, A. Cabrini, O. Khouri, and G. Torelli. On-chip error correcting techniques for new-generation Flash memories. *Proceedings of the IEEE*, 91(4):602–616, 2003.

[Gib92] G. Gibson. *Redundant Disk Arrays*. MIT Press, Cambridge MA, USA, 1992.

[GKKG06] W. Gross, F. Kschischang, R. Kötter, and P. Gulak. Applications of algebraic soft-decision decoding of reed-solomon codes. *IEEE-Trans-Comm*, 54(7):1224–1234, 2006.

[GRS00] O. Goldreich, R. Rubinfeld, and M. Sudan. Learning polynomials with queries: The highly noisy case. *SIAM J. Discrete Math*, 13(4):535–570, November 2000.

[GS99] V. Guruswami and M. Sudan. Improved decoding of Reed-Solomon and algebraic-geometric codes. *IEEE Transactions on Information Theory*, 45:1755–1764, 1999.

[GS01] V. Guruswami and M. Sudan. Extensions to the johnson bound. *Manuscript*, 2001.

[Gur01] V. Guruswami. *List decoding of error-correcting codes*. Ph. D. Dissertation, Massachusetts Institute of Technology, 2001.

[HCL07] C. Huang, M. Chen, and J. Li. Pyramid codes: flexible schemes to trade space for access efficiency in reliable data storage systems. In *In Proceedings of the Sixth IEEE International Symposium on Network Computing and Applications*, Cambridge, MA USA, 2007.

[HM77] Z.M Huntoon and A.M Michelson. On the computation of the probability of post-decoding error events for block codes. *IEEE Transactions on Information Theory*, 23(3):399–403, May 1977.

[HP03] W.C Huffman and V. Pless. *Fundamentals of Error-Correcting Codes*. Cambridge university press, Cambridge, UK, 2003.

[HX05] C. Huang and L. Xu. Star: An efficient coding scheme for correcting triple storage node failures. In *In Proceedings of the 4th USENIX Conference on File and Storage Technologies*, San-Francisco CA, 2005.

[KF59] W. Kim and C. Freiman. Single error-correcting codes for asymmetric binary channels. *IRE Transactions on Information Theory*, 5(2):62–66, 1959.

[Klø81] T. Kløve. Error correcting codes for the asymmetric channel. Technical Report 18-09-07-81, Dept. Mathematics, University of Bergen, Norway, 1981.

[Köt96] R. Kötter. Fast generalized minimum-distance decoding of algebraic-geometry and Reed-Solomon codes. *IEEE Transactions on Information Theory*, 42(3):721–736, 1996.

[KV03a] R. Kötter and A. Vardy. Algbraic soft-decision decoding of Reed-Solomon codes. *IEEE Transactions on Information Theory*, 49(11):2809–2825, November 2003.

[KV03b] R. Kötter and A. Vardy. A complexity reducing transformation in algebraic list decoding of Reed-Solomon codes. In *proc. of ITW*, Paris, 2003.

[LC83] S. Lin and D. Costello. *Error Control Coding: Fundamentals and Applications*. Prentice-Hall, Englewood Cliffs, NJ, 1983.

[LN86] R. Lidl and H. Niederreiter. *Introduction to finite fields and their applications*. Cambridge University Press, Cambridge UK, 1986.

[LR06] E. Louidor and R.M Roth. Lowest-density MDS codes over extension alphabets. *IEEE Transactions on Information Theory*, 52(7):3186–3197, 2006.

[McE73] R.J McEliece. Comment on ”a class of codes for asymmetric channels and a problem from the additive theory of numbers”. *IEEE Transactions on Information Theory*, 19(1):137, 1973.

[McE03a] R.J McEliece. The guruswami-sudan algorithm for decoding Reed-Solomon codes. Technical Report IPN progress report 42-153, JPL, http://www.ipnpr.jpl.nasa.gov/progress_report/42-153/, 2003.

[McE03b] R.J McEliece. On the average list size for the guruswami-sudan decoder. In *proc. of the International Symposium on Communication Theory and Applications*, pages 2–6. IEEE, July 2003.

[MS77] F.J MacWilliams and N.J.A Sloane. *The Theory of Error-Correcting Codes*. North Holland, Amsterdam, The Netherlands, 1977.

[NH00] R.R Nielsen and T. Hoholdt. Decoding Reed-Solomon codes beyond half the minimum distance. *Cryptography and Related Areas, J. Buchmann et al. eds. Springer-Verlag 2000*, pages 221–236, 2000.

[PGK88] D. A. Patterson, G. A. Gibson, and R. Katz. A case for redundant arrays of inexpensive disks. In *Proc. SIGMOD Int. Conf. Data Management*, pages 109–116, 1988.

[PW72] W.W Peterson and E.J Weldon. *Error-Correcting Codes*. MIT Press, Cambridge, MA, 1972.

[RR03] G. Ruckenstein and R.M Roth. Bounds on the list-decoding radius of Reed-Solomon codes. *SIAM J. Discrete Math*, 17(2):171–195, November 2003.

[Sha48] C.E Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(9):379–423, October 1948.

[Sha56] C.E Shannon. The zero error capacity of a noisy channel. *IRE Transactions on Information Theory*, 2:S8–S19, September 1956.

[Ste84] S. Stein. Packings of $\mathbb{R}^n$ by certain error spheres. *IEEE Transactions on Information Theory*, 30(2):356–363, March 1984.

[Sud97] M. Sudan. Decoding of Reed-Solomon codes beyond the error correction bound. *J. Complexity*, 12:180–193, 1997.

[TB97] L. Trefethen and D. Bau. *Numerical linear algebra*. Society for industrial and applied mathematics, Philadelphia PA, 1997.

[TW67] R.L Townsend and E.J Weldon. Self-orthogonal quasi-cyclic codes. *IEEE Transactions on Information Theory*, 13(2):183–195, 1967.

[Var73] R. Varshamov. A class of codes for asymmetric channels and a problem from the additive theory of numbers. *IEEE Transactions on Information Theory*, 19(1):92–95, 1973.

[Web92] J. Weber. Necessary and sufficient conditions on block codes correcting/detecting errors of various types. *IEEE Transactions on Computers*, 41(9):1189–1193, 1992.

[Wol06] J.K Wolf. An introduction to tensor product codes and applications to digital storage systems. In *Proc. IEEE Information Theory Workshop*, pages 6–10, Chengdu, China, 2006.

[Woz58] J.M Wozencraft. List decoding. Technical Report 48, Quarterly progress report, Research Laboratory of Electronics, MIT, 1958.

[XB99] L. Xu and J. Bruck. X-code: MDS array codes with optimal encoding. *IEEE Transactions on Information Theory*, 45(1):272–276, 1999.

[XBBW99] L. Xu, V. Bohossian, J. Bruck, and D.G Wagner. Low-density MDS codes and factors of complete graphs. *IEEE Transactions on Information Theory*, 45(6):1817–1826, 1999.

[ZZS81] G.V Zaitsev, V.A Zinov'ev, and N.V Semakov. Minimum-check-density codes for correcting bytes of errors, erasures, or defects. *Problems Inform. Transm.*, 19:197–204, 1981.