List Decoding of Error-Correcting Codes

by

Venkatesan Guruswami

B.Tech., Indian Institute of Technology, Madras (1997)
S.M., Massachusetts Institute of Technology (1999)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2001

Author . . . .
Department of Electrical Engineering and Computer Science
August 29, 2001

Certified by . . . .
Madhu Sudan
Associate Professor
Thesis Supervisor

Accepted by . . . .
Arthur C. Smith
Chairman, Departmental Committee on Graduate Students

List Decoding of Error-Correcting Codes

by
Venkatesan Guruswami

Submitted to the Department of Electrical Engineering and Computer Science
on August 29, 2001, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Error-correcting codes are combinatorial objects designed to cope with the problem of reliable transmission of information on a noisy channel. A fundamental algorithmic challenge in coding theory and practice is to efficiently decode the original transmitted message even when a few symbols of the received word are in error. The naive search algorithm runs in exponential time, and several classical polynomial time decoding algorithms are known for specific code families. Traditionally, however, these algorithms have been constrained to output a unique codeword. Thus they faced a "combinatorial barrier" and could only correct up to $d/2$ errors, where d is the minimum distance of the code.

An alternate notion of decoding called *list decoding*, proposed independently by Elias and Wozencraft in the late 50s, allows the decoder to output a list of *all* codewords that differ from the received word in a certain number of positions. Even when constrained to output a relatively small number of answers, list decoding permits recovery from errors well beyond the $d/2$ barrier, and opens up the possibility of meaningful error-correction from large amounts of noise. However, for nearly four decades after its conception, this potential of list decoding was largely untapped due to the lack of *efficient* algorithms to list decode beyond $d/2$ errors for useful families of codes.

This thesis presents a detailed investigation of list decoding, and proves its potential, feasibility, and importance as a combinatorial and algorithmic concept. We prove several combinatorial results that sharpen our understanding of the potential and limits of list decoding, and its relation to more classical parameters like the rate and minimum distance. The crux of the thesis is its algorithmic results, which were lacking in the early works on list decoding. Our algorithmic results include:

- Efficient list decoding algorithms for classically studied codes such as Reed-Solomon codes and algebraic-geometric codes. In particular, building upon an earlier algorithm due to Sudan, we present the *first* polynomial time algorithm to decode Reed-Solomon codes beyond $d/2$ errors for every value of the rate.
- A new *soft* list decoding algorithm for Reed-Solomon and algebraic-geometric codes, and novel decoding algorithms for concatenated codes based on it.
- New code constructions using concatenation and/or expander graphs that have good (and sometimes near-optimal) rate and are efficiently list decodable from extremely large amounts of noise.
- Expander-based constructions of *linear time encodable and decodable* codes that can correct up to the maximum possible fraction of errors, using unique (not list) decoding.

Thesis Supervisor: Madhu Sudan
Title: Associate Professor

Acknowledgments

We know too much for one man to know much.
J. Robert Oppenheimer

When I made a decision to go to MIT for grad school in the spring of 1997, I was not aware that Madhu Sudan would be joining its faculty that Fall. And while in the last four years I've found MIT to be every bit the wonderful place I anticipated it to be and more, I'd still not be exaggerating if I said that my academic experience at MIT might not have been nearly as enjoyable and fulfilling had I not had Madhu as an advisor. For the collaborations which led to several of the key chapters of this thesis, for all his patient advice and support on matters technical and otherwise, and for all the things I learned from him during my stay at MIT, I'll be forever grateful to Madhu.

I'm greatly indebted to Johan Håstad, Piotr Indyk, Amit Sahai, Madhu Sudan, and David Zuckerman for their collaborations which led to several of the results discussed in this thesis. I also wish to thank the several other people with whom I've had useful discussions on topics related to this thesis. These include Noga Alon, Sasha Barg, Peter Elias, Tom Hϕholdt, Jϕrn Justesen, Ralf Koetter, Hendrik Lenstra, Rasmus Nielsen, Amin Shokrollahi, Dan Spielman, Salil Vadhan, and Alex Vardy, though undoubtedly I've left out a few.

I'd like to thank all the other people (I'm fortunate there are so many) with whom I have had the pleasure of collaborating in the last few years on topics not discussed in this thesis. They all have contributed immensely towards my development as a researcher. Of these I'd especially like to mention: (i) Johan, who is a great person to work with and whom I admire not only for his well-known ingenious ideas, but also for his rigorous standards of writing, and (ii) Sanjeev Khanna, who was my mentor during my first summer at Bell Labs, and who has been a great collaborator and well-wisher ever since.

A special thanks is due to the members of my thesis reading committee: Peter Elias, Dan Spielman, and Madhu Sudan. They kindly found the time to read large portions of my thesis and gave numerous useful comments and suggestions. Technically, it was only appropriate that I had these three people on my committee: Peter first defined the notion of list decoding; Madhu discovered the first non-trivial efficient list decoding algorithm; and Dan constructed the first linear-time codes (the subject of Chapter 11 of this thesis).

I thank Mike Sipser for giving me the opportunity to twice serve as a TA for his Theory of Computation class. Mike is a truly wonderful teacher, and working with him was a great experience and a lot of fun.

During my graduate study, I had the good fortune of being a summer student at Bell Labs and at IBM Almaden. These experiences were invaluable to me, if only for the great people I got to know and the wonderful collaborators I found. I'm most grateful to my mentors, managers, and collaborators at these places. A special thanks to my good friends Moses Charikar and Amit Sahai for all the good times we shared during the summer when we were all at Almaden.

During my stay at MIT, inspite of my high-speed connection at home and the amazing ugliness of the immediate surroundings of LCS, I spent a lot of time hanging out at LCS. This was largely due to the wonderfully stimulating work atmosphere and the company of a great group of friends and colleagues that I found in the theory group. The good time I had at MIT owes a lot to the wonderful student body I had the privilege of being a part of. I would like to thank Salil, Yevgeniy, Eric, Amit, Raj, Anna, Sofya, Adam K, Adam

S, Maria, Leo, Zully, Matthias, Feldman, Abhi, Tal, Daniele, Alantha, Dunagan, Prahladh, and many others, for numerous conversations on all sorts of topics, and for making my life at LCS so much fun. A special thanks to Maria Minkoff for putting up with me as officemate for a good chunk of my stay and for her wonderful company in the office. Thanks are due to Sofya and Adam for being my co-TAs for 6.840 which made my task even more easy and enjoyable. Adam Klivans also deserves a special mention for all those great discussions we had on a wide spectrum of topics ranging from baseball to derandomization. I was lucky that Luca Trevisan was at MIT the year I started; from him I learned a lot, and with him (and Danny and Madhu) I shared my first research experience in grad school.

My last year at MIT benefited a great deal from the time I spent working and hanging out with Piotr Indyk, for which I sincerely thank him. Piotr also gave me invaluable help in my job application process. Lars Engebretsen, who together with Piotr and me comprised the infamous coffee-club, deserves a special mention for all the time we spent sipping espresso and talking about PCPs and all sorts of things. It goes without saying that the company of many of these people extended beyond the confines of LCS, and in this regard I'd like to thank Piotr, Lars, Adam, Anna, Prahladh, Matthias, Amit, and Yevgeniy, to name a few.

My thanks to the theory group staff, and in particular Joanne Talbot and Be Blackburn, for their good cheer and all their administrative and other help. Joanne was amazingly efficient with processing my several travel reimbursement requests, which was most useful.

My life outside LCS and the time I spent at home were so enjoyable thanks to my good friends Harish and Sreeram, who were my housemates for three years. A most special thanks to both of them. Harish has been a great friend and has been of big help on a number of occasions in the last three years. Thanks also to Mahadevan, my neighbor for three years, if only for all those good dinners we cooked together.

I guess it is common to make one's best friends in college. I'm no exception to this. When at IIT Madras, I was fortunate to have a most wonderful and inspiring set of fellow CS majors, many of whom are among my closest friends. True friends are those who take pride in your achievements, and I am grateful that I have several who meet this definition.

I owe a lot to two professors from college: C. Pandu Rangan for encouraging me in every possible way and getting me started on research well before I started grad school; and S. A. Choudum whose wonderful Graph Theory course sparked my interest in algorithmic graph theory and eventually theoretical computer science.

My most important acknowledgment is to my close and loving family: my parents and my sister, who have filled my life with joy and have *always* had confidence in me, even on occasions when I myself didn't. Words cannot express my thanks to my parents for all that they have gone through and done for me.

Cambridge, Massachusetts **_Venkatesan Guruswami_**
August, 2001

I gratefully acknowledge the fellowships and grants that supported my research at MIT. My research was supported in part by funding from NSF CCR 9875511, NSF CCR 9912342, and NTT Award MIT 2001-04, and in part by an IBM Graduate Fellowship.

To Mummy and Appa

Contents

List of Figures

Chapter 1

Introduction

In the everyday situation where one party wishes to communicate a message to another distant party, more often than not, the intervening communication channel is "noisy" and distorts the message during transmission. The problem of reliable communication of information over such a noisy channel is a fundamental and challenging one. *Error-correcting codes* (or simply, *codes*) are objects designed to cope with this problem. They are now ubiquitous and found in all walks of life, ranging from basic home and office appliances like compact disc players and computer hard disk drives to deep space communication.

The theory of error-correcting codes, which dates back to the seminal works of Shannon [Sha48] and Hamming [Ham50], is a rich, beautiful and to-date flourishing subject that benefits from techniques developed in a wide variety of disciplines such as combinatorics, probability, algebra, geometry, number theory, engineering, and computer science, and in turn has diverse applications in a variety of areas.

This thesis studies the performance of error-correcting codes in the presence of very large amounts of noise, much more than they were "traditionally designed" to tolerate. This situation poses significant challenges not addressed by the classical decoding procedures. This thesis addresses these challenges with a focus on the algorithmic issues that arise therein. Specifically, we establish limits on what can be achieved in such a high-noise situation, and present algorithms for classically studied codes that decode significantly more errors than all previously known methods. We also present several novel code constructions designed to tolerate extremely large amounts of noise, together with efficient error-correcting procedures. The key technical notion underlying our work is *"List Decoding"*. This notion will be defined, and our contributions will be explained in further detail, later on in this chapter.

1.1 Basics of Error-correcting codes

Informally, error-correcting codes provide a systematic way of adding redundancy to a message before transmitting it, so that even upon receipt of a somewhat corrupted message, the redundancy in the message enables the receiver to figure out the original message that the sender intended to transmit.

The principle of redundant encoding is in fact a familiar one from everyday language. The set of all words in English is a small subset of all possible strings, and a huge amount of redundancy is built into the valid English words. Consequently, a misspelling in a word usually changes it into some incorrect word (i.e., some letter sequence that is not a valid word in English), thus enabling *detection* of the spelling error. Moreover, the resulting misspelled word quite often resembles the correct word more than it resembles any other word, thereby permitting *correction* of the spelling error. The "ispell" program used to spell-check this thesis could not have worked but for this built-in redundancy of the English language! This simple principle of "built-in redundancy" is the essence of the theory of error-correcting codes.

In order to be able to discuss the context and contributions of this thesis, we need to define some basic notions concerning error-correcting codes.[1] These are discussed below.

- Encoding. An *encoding function* with parameters k, n is a function $E : \Sigma^k \to \Sigma^n$ that maps a *message* m consisting of k symbols over some alphabet Σ (for example, the binary alphabet $\Sigma = \{0, 1\}$) into a longer, redundant string $E(m)$ of length n over Σ. The encoded string $E(m)$ is referred to as a *codeword*.
- Error-Correcting code. The *error-correcting code* itself is defined to be the image of the encoding function. In other words, it is the set of all codewords which are used to encode the various messages.
- Rate. The ratio of the number of information symbols to the length of the encoding — the quantity k/n in the above definition — is called the *rate* of the code. It is an important parameter of a code, as it is a measure of the amount of redundancy added by the encoding.
- Decoding. Before transmitting a message, the sender of the message first encodes it using the error-correcting code and then transmits the resulting codeword along the channel. The receiver gets a possibly distorted copy of the transmitted codeword, and needs to figure out the original message which the sender intended to communicate. This is done via a *decoding function*, $D : \Sigma^n \to \Sigma^k$, that maps strings of length n (i.e., noisy received words) to strings of length k (i.e., what the decoder thinks was the transmitted message).
- Distance. The *minimum distance* (or simply, *distance*) of a code quantifies how "far apart" from each other different codewords are. Define the *distance* between words as the number of coordinates at which they differ. The (minimum) distance of a code is then defined to be the smallest distance between two distinct codewords.

Historical Interlude: We now briefly discuss the history behind the definition of these concepts. The notions of encoding, decoding, and rate appeared in the work of Shannon [Sha48]. The notions of an error-correcting code itself, and that of the distance of a code, originated in the work of Hamming [Ham50].

Shannon proposed a *stochastic model* of the communication channel, in which distortions are described by the conditional probabilities of the transformation of one symbol into another. For every such channel, Shannon proved that there exists a precise real number, which he called the channel's *capacity*, such that in order to achieve reliable communication over the channel, one has to use a encoding function with rate less than its capacity. He also proved the converse result — namely, for every rate below capacity, there *exist* encoding and decoding schemes which can used to achieve reliable communication, with a probability of miscommunication as small as one desires. This remarkable result, which precisely characterized the amount of redundancy needed to cope with a noisy channel, marked the birth of information theory and coding theory.

[1] Here we only define the most basic notions. Further definitions appear in Chapter 2.

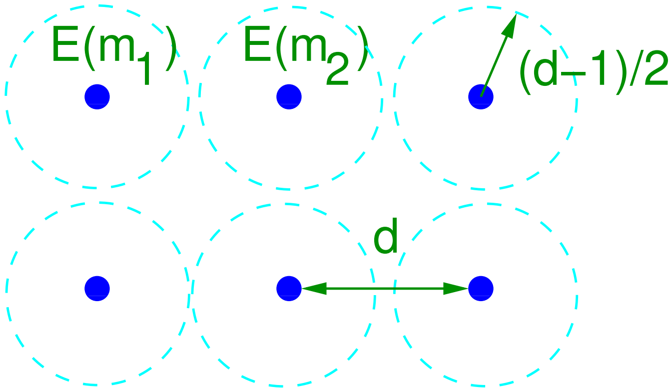

Figure 1-1: A code of minimum distance d. Spheres of radius $(d-1)/2$ around the codewords are all disjoint.

However, Shannon only proved the *existence* of good coding schemes at any rate below capacity, and it was not at all clear how to perform the required encoding or decoding efficiently. Moreover, the stochastic description of the channel did not highlight any simple criterion of when a certain code is good.

Intuitively, a good code should be designed so that the encoding of one message will not be confused with that of another, even if it is somewhat distorted by the channel. Now, if the various codewords are all *far apart* from one another, then even if the channel distorts a codeword by a small amount, the resulting string will still resemble the original codeword much more than any other codeword, and can therefore be "decoded" to the correct codeword. In his seminal work, Hamming [Ham50] realized the importance of quantifying how far apart various codewords are, and defined the above notion of *distance* between words, which is now appropriately referred to as *Hamming distance*. He also formally defined the notion of an error-correcting code as a collection of strings no two of which are close to each other, and defined the (minimum) distance of a code as the smallest distance between two distinct codewords. This notion soon crystallized as a fundamental parameter of an error-correcting code. Figure 1-1 depicts an error-correcting code with minimum distance d, which, as the figure illustrates, implies that Hamming balls of radius $(d-1)/2$ around each codeword are all disjoint. In this model, an optimal code is one with the largest minimum distance among all codes that have a certain number of codewords. As Figure 1-1 indicates, finding a good code in this model is a particular kind of "sphere-packing" problem. Unlike Shannon's statistical viewpoint, this combinatorial formulation permitted a variety of techniques from combinatorics, algebra, geometry, and number theory to be applied in attempts to solve the problem. In turn, this led to the burgeoning of coding theory as a discipline.

1.2 The Decoding Problem for Error-correcting Codes

The two main algorithmic tasks associated with the use of an error-correcting code are implementing the encoding function E and the decoding function D. The former task is usually easy to perform efficiently, since the "construction" of a code often works by giving

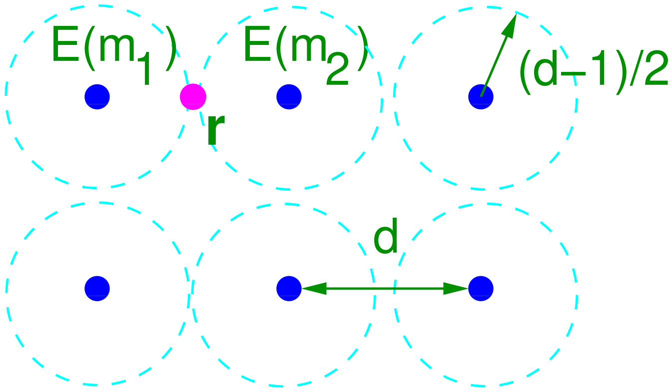

Figure 1-2: A code of distance d cannot correct $d/2$ errors. The figure shows a received word $\mathbf{r}$ at a distance of $d/2$ from two codewords corresponding to the encodings of m_1 and m_2. In such a case, $\mathbf{r}$ could have resulted from $d/2$ errors affecting either $E(m_1)$ or $E(m_2)$.

such an encoding procedure.

For the decoding problem, we would ideally like $D(E(m)+\text{noise}) = m$ for every message m, and every "reasonable" noise pattern that the channel might effect. Now, suppose that the error-correcting code has minimum distance d (assume d is even) and m_1, m_2 are two messages such that the Hamming distance between $E(m_1)$ and $E(m_2)$ is d. Then, assume that $E(m_1)$ is transmitted and the channel effects $d/2$ errors and distorts $E(m_1)$ into a word $\mathbf{r}$ that is right in between $E(m_1)$ and $E(m_2)$ (see Figure 1-2). In this case, upon receiving $\mathbf{r}$, the decoder has no way of figuring out which one of m_1 or m_2 was the intended message, since $\mathbf{r}$ could have been received as a result of $d/2$ errors affecting either $E(m_1)$ or $E(m_2)$.

Therefore, when using a code of minimum distance d, a noise pattern of $d/2$ or more errors cannot always be corrected. On the other hand, for any received word $\mathbf{r}$, there can be only one codeword within a distance of $(d-1)/2$ from $\mathbf{r}$. This follows using the triangle inequality (since Hamming distance between strings defines a metric). Consequently, if the received word $\mathbf{r}$ has at most $(d-1)/2$ errors, then the transmitted codeword is the *unique* codeword within distance $(d-1)/2$ from $\mathbf{r}$ (see Figure 1-3). Hence, by searching for a codeword within distance $(d-1)/2$ of the received word, we can recover the correct transmitted codeword as long the number of errors in the received word is at most $(d-1)/2$.

Due to these facts, a well-posed algorithmic question that has been the focus of most of the classical algorithmic work on efficient decoding, is the problem of decoding a code of minimum distance d up to $(d-1)/2$ errors. We call such a decoding *unique/unambiguous decoding* in the sequel. The reason for this terminology is that the decoding algorithm decodes only up to a number of errors for which it is *guaranteed* to find a unique codeword within such a distance of the received word.

The obvious unique decoding algorithms which search the vicinity of the received word for a codeword are inefficient and require exponential runtime. Nevertheless, a classic body of literature spanning four decades has now given efficient unique decoding algorithms for several interesting families of codes. These are among the central and most important results in algorithmic coding theory, and are discussed in detail in any standard coding theory text (eg., [MS81, vL99]).

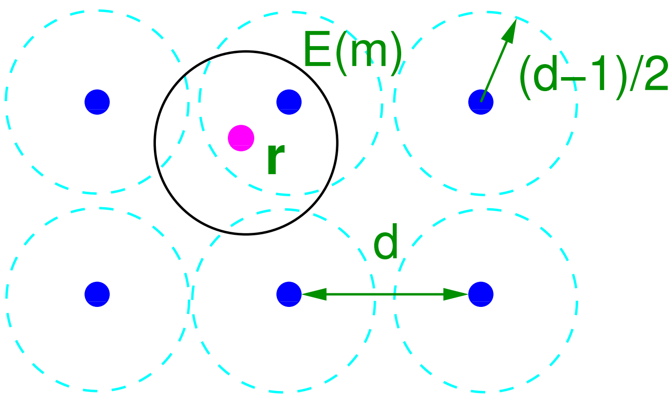

Figure 1-3: A code of distance d can correct up to $(d-1)/2$ errors. For the received word $\mathbf{r}$, $E(m)$ is the unique codeword within distance $(d-1)/2$ from it, so if fewer than $(d-1)/2$ errors occurred, $\mathbf{r}$ can be correctly decoded to m.

We are interested in what happens when the number of errors is greater than $(d-1)/2$. In such a case the unique decoding algorithms could either output the wrong codeword (i.e., a codeword other than the one transmitted), or report a decoding failure and not output any codeword. The former situation occurs if the error pattern takes the received word within distance $(d-1)/2$ of some other codeword. In such a situation, the decoding algorithm, though its answer is wrong, cannot really be faulted. After all, it found some codeword much closer to the received word than any other codeword, and in particular the transmitted codeword, and naturally places its bet on that codeword. The latter situation occurs if there is no codeword within distance $(d-1)/2$ of the received word, and it brings out the serious shortcoming of unique decoding, which we discuss below.

It is true that *some* patterns of $d/2$ errors, as in Figure 1-2, are uncorrectable due to there being multiple codewords at a distance $d/2$ from the received word. However, the situation in Figure 1-2 is quite pathological and it is actually the case that for *most* received words there will be only a single codeword that is closest to it. Moreover, the sparsity of the codewords implies that most words in the ambient space fall outside the region covered by the (disjoint) spheres of radius $(d-1)/2$ around the codewords. Together, these facts imply that most received words have a unique closest codeword (and thus it is reasonable to expect that the decoding algorithm correct them to their closest codeword), and yet unique decoding algorithms simply fail to decode them. Indeed, as Shannon's work [Sha48] already pointed out, for "good" codes (namely, those that approach capacity), if errors happen randomly according to some reasonable probabilistic model, then with high probability the received word will not be correctable by unique decoding algorithms!

In summary, on a overwhelming majority of error patterns, unique decoding uses the excuse that there is no codeword within a distance $(d-1)/2$ from the received word to completely give up on decoding those patterns. This limitation is in turn due to the requirement that the decoding *always* be unique or unambiguous, which, as argued earlier, means there are some (pathological) patterns of $d/2$ errors which are not correctable. It turns out that there is a meaningful relaxation of unique decoding which circumvents this predicament and permits one to decode beyond the perceived "half-the-distance barrier" faced by unique decoding. This relaxed notion of decoding, called *list decoding*, is the subject of this thesis, and we turn to its definition next.

1.3 List Decoding

1.3.1 Definition

List decoding was introduced independently by Elias [Eli57] and Wozencraft [Woz58] in the late 50's. List decoding is a relaxation of unique decoding that allows the decoder to output a *list* of codewords as answers. The decoding is considered successful as long as the codeword corresponding to the correct message is included in the list. Formally, the list decoding problem for a code $E : \Sigma^k \to \Sigma^n$ is defined as follows: Given a received word $\mathbf{r} \in \Sigma^n$, find and output a list of all messages m such that the Hamming distance between $\mathbf{r}$ and $E(m)$ is at most e. Here e is a parameter which is the number of errors that the list decoding algorithm is supposed to tolerate. The case $e = (d-1)/2$ gives the unique decoding situation considered earlier.

List decoding permits one to decode beyond the half-the-distance barrier faced by unique decoding. Indeed, in the situation of Figure 1-2, the decoder can simply output both the codewords that are at a distance of $d/2$ from the received word. In fact, list decoding remains a feasible notion even when the channel effects $e \gg d/2$ errors.

An important parameter associated with list decoding is the size of the list that the decoder is allowed to output. Clearly with a list size equal to one, list decoding just reduces to unique decoding. It is also undesirable to allow very large list sizes. This is due to at least two reasons. First, there is the issue of how useful a very large list is, since it is reasonable that the receiver might finally want to pick one element of the list using additional rounds of communication or using some tie-breaking criteria. Second, the decoding complexity is at least as large as the size of the list that the algorithm must output in the worst-case. Since we want efficient, polynomial time, decoding procedures, the list size should be at most a polynomial in the message length, and ideally at most a constant that is independent of the message length.

It turns out that even with a list size that is a small constant (say, 20), *any* code of distance d can be list decoded well beyond $d/2$ errors (for a wide range of distances d). The bottom line, therefore, is that allowing the decoder to output a small list of codewords as answers opens up the possibility of doing much better than unique decoding. In other words, list decoding is *combinatorially feasible.*

1.3.2 Is list decoding a useful relaxation of unique decoding?

But the above discussion does not answer the obvious question concerning list decoding that comes to one's mind when first confronted with its definition: how useful is the notion of list decoding itself? What does one do with a list of answers, and when too many errors occur, why is receiving an ambiguous list of answers better than receiving no answer at all? We now proceed to answer these questions.

Firstly, notice that list decoding only gives more options than unique decoding. One can always go over the list output by the algorithm and check if there is any codeword within distance $(d-1)/2$ of the received word, thereby using it to perform unique decoding. But the advantage of list decoding is that it also enables meaningful decoding of received words that have no codeword within distance $(d-1)/2$ from them. As discussed earlier, since the codewords are far apart from one another and sparsely distributed, most received words in fact fall in this category. For a large fraction of such received words, one can show that in fact there is at most one codeword within distance e from them, for some bound e which is much greater than $d/2$. Therefore, list decoding up to e errors will usually (i.e., for most received words) produce lists with at most one element, thereby obviating the need of dealing with more than one answer being output! In particular, for a channel that effects errors randomly, this implies that with high probability, list decoding, when it succeeds, will output exactly one codeword.

Furthermore, if the received word is such that list decoding outputs several answers, this is certainly no worse than giving up and reporting a decoding failure (since we can always choose to return a failure if the list decoding does not output a unique answer). But, actually, it is much better. Since the list is guaranteed to be rather small, using an extra round of communication, or some other context or application specific information, it might actually be possible to disambiguate between the answers and pick one of them as the final output. For example, the "ispell" program used to spell-check this thesis often outputs a list of correct English words that are close to the misspelled word. The author of the document can then conveniently pick one of the words based on what he/she actually intended. As another example, consider the situation where a spacecraft transmits the encoding of, say a picture of Saturn, back to the Earth. It is possible that due to some unpredictable interference in space, the transmission gets severely distorted and the noise is beyond the range unique decoding algorithms can handle. In such a case, it might be unreasonable to request a retransmission from the spacecraft. However, if a list decoding algorithm could recover a small list of candidate messages from the received data, then odds are that only one member of the list will look anything like a picture of Saturn, and we will therefore be able to recover the original transmitted image.

Also, we would like to point out that one can always pick from the list the codeword closest to the received word, if there is a unique such codeword, and output it. This gives the codeword that has the highest likelihood of being the one that was actually transmitted, even beyond the half-the-distance barrier. Finding such a codeword is referred to as *maximum likelihood decoding* in the literature. See the "interlude" at the end of this section for further discussion about this point, but in a nutshell, list decoding permits one to perform maximum likelihood decoding as long as the number of errors effected by the channel is bounded by the maximum number of errors that the list decoding algorithm is designed to tolerate.

Finally, error-correcting codes and decoding algorithms play an important role in several contexts outside communication, and in fact they have become fundamental primitives in theoretical computer science. As we will see in this thesis, in many cases list decoding enhances the power of this connection between coding theory and computer science.

Interlude: Maximum likelihood decoding (MLD) is an alternate notion of decoding considered in the literature. The goal of MLD is to output the codeword closest in Hamming distance to the received word (ties broken arbitrarily). This is considered by many to be the "holy grail" of decoding, since it outputs the codeword with the highest likelihood of being the one that was actually transmitted. MLD clearly generalizes unique decoding, since if there is a codeword within distance $(d-1)/2$ of the received, it must be the unique closest codeword. List decoding and MLD are, however, incomparable in power. List decoding can be used to perform MLD as long as the number of errors is bounded by the amount that the list decoding algorithm was designed to tolerate. In such a case, list decoding is in fact a more general primitive since it gives all close-by codewords, including the closest one(s), while a MLD algorithm rigidly makes up its mind on one codeword. On the other hand, MLD does not assume any bound on the number of errors, while list decoding, owing to the requirement of small list size in the worst-case, does. The main problem with MLD is that it ends up being computationally intractable in general, and extremely difficult to solve even for particular families of codes. In fact, the author is unaware of *any* non-trivial code family for which maximum likelihood decoding is solvable in polynomial time. In contrast, list decoding, as this thesis demonstrates, is algorithmically tractable for several interesting families of codes. ***End Interlude***

1.3.3 The challenge of list decoding

The real problem with list decoding was not that it was not considered to be useful, but that there were no known algorithms to *efficiently* list decode well beyond half-the-distance for any useful family of error-correcting codes (even though it was known that, combinatorially, list decoding offered the potential of decoding many more than $d/2$ errors using small lists). The naive brute-force search algorithms all take exponential time, and we next give some idea of why efficient list decoding algorithms have remained so elusive, despite substantial progress on efficient unique decoding.

Classical unique decoding algorithms decode only up to half-the-distance. In particular, they can never decode when more than half the symbols are in error. List decoding, on the other hand, aims to handle errors well beyond half-the-distance, and consequently, must even deal with situations where more than half the symbols are in error, including cases where *the noise is overwhelming and far out-weighs the correct information*. In fact, list decoding opens the potential of decoding when the noise is close to 100%. Realizing the potential of list decoding in the presence of such extreme amounts of noise poses significant algorithmic challenges under which the ideas used in the classical decoding procedures break down.

1.3.4 Early work on list decoding

The early work on list decoding focused only on statistical or combinatorial aspects of list decoding. We briefly discuss these works below. The initial works by Elias [Eli57] and Wozencraft [Woz58], which defined the notion of list decoding, proved tight bounds on the error probability achievable through list decoding on certain probabilistic channels. Results of a similar flavor also appear in [SGB67, For68, Ahl73]. Elias [Eli88] also generalized the zero error capacity of Shannon [Sha56] to list decoding and obtained bounds on the zero error capacity of channels under list decoding with lists of certain size.

The focus in the 80's shifted to questions of a more combinatorial nature, and considered the worst-case list decoding behavior of codes. The central works in this vein are [ZP82, Bli86, Eli91]. These established bounds on the number of errors that could be corrected by list decoding using lists of a certain fixed size, for error-correcting codes of a certain rate. This is in the spirit of the questions that this thesis investigates. However, none of these results presented any non-trivial list decoding algorithms.[2] Thus, despite being an extremely useful generalization of unique decoding, the potential of list decoding was largely untapped due to the lack of good algorithms.

[2]Here triviality is used to rule out both brute-force search algorithms and unique decoding algorithms.

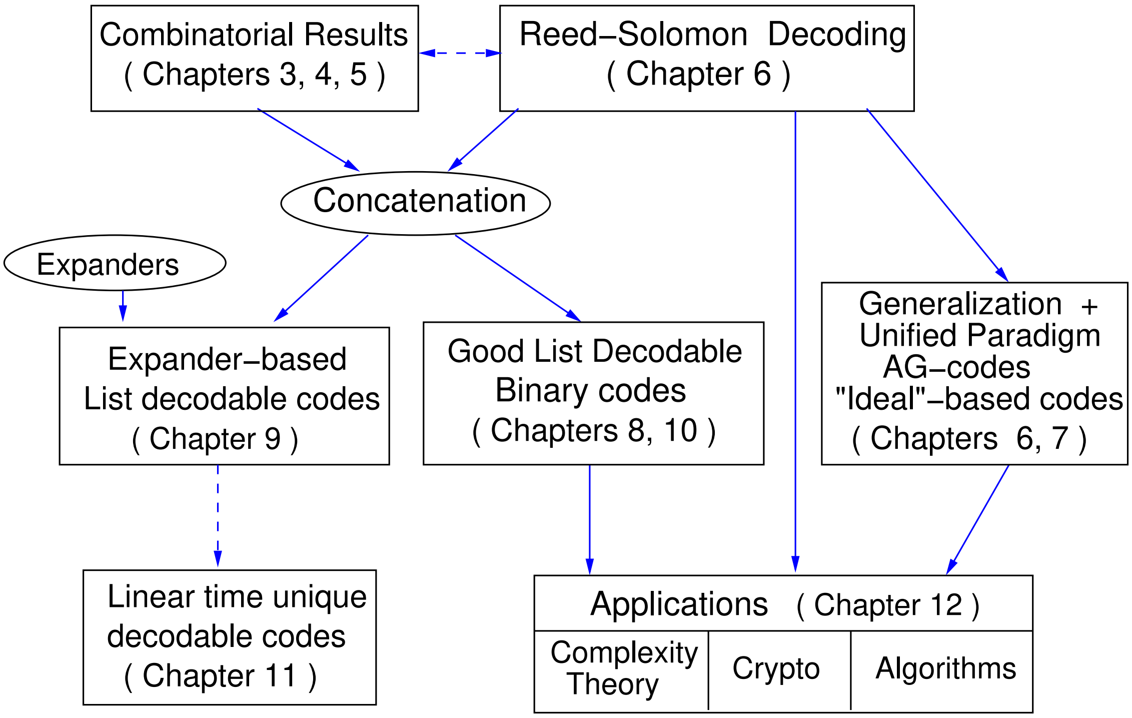

Figure 1-4: Contributions of this thesis: a high level view

1.4 Contributions of this thesis

This thesis presents a systematic and comprehensive investigation of list decoding, with a focus on algorithmic results. Presenting our contributions in sufficient detail would require several more definitions. Therefore, we only give a high level description of the contributions of the thesis here, deferring a detailed discussion of the contributions of the individual chapters and how they fit together to the next chapter. Figure 1-4 gives a bird's eye view of the kind of results in this thesis. The following description is probably best read with Figure 1-4 in mind.

The first part of thesis investigates certain combinatorial aspects of list decoding. We study the trade-offs between the list decodability of a code and the more classical parameters like rate and minimum distance. The results yield a significant sharpening of our understanding of the potential and limits of list decoding. Our combinatorial results are important in their own right and also because they set the stage for, and are repeatedly appealed to or used in, several subsequent results. The crux of the thesis is its algorithmic results, which comprise the second part of the thesis.

The highlight here is a list decoding algorithm for Reed-Solomon codes (Chapter 6). Reed-Solomon codes are among the most important and widely studied families of codes, and several classical unique decoding algorithms are known for them (cf. [MS81, Chapters 9,10]). However, despite over four decades of research, there was no known algorithm to efficiently list decode Reed-Solomon codes well beyond $d/2$ errors where d is the minimum distance. In Chapter 6 of this thesis, we present the *first* polynomial time list decoding algorithm that corrects more than $d/2$ errors for every value of the rate. This result builds upon an earlier breakthrough result of Sudan [Sud97a] who gave such an algorithm for Reed-Solomon codes of low rate. Our result list decodes up to what might well be the true list decoding potential of Reed-Solomon codes. We also generalize the algorithm to algebraic-geometric codes. The novelty of our technique enables us to also get a more general *soft* list decoding algorithm, which can take advantage of reliability information on the various symbols. This is the first non-trivial soft decoding algorithm with a provable performance guarantee for Reed-Solomon codes since the classic 1966 work of Forney [For66b] on Generalized Minimum Distance (GMD) decoding.

Using our decoding algorithms for Reed-Solomon codes at the core, we also obtain several other non-trivial list decoding algorithms. These include novel algorithms for list decoding of several *concatenated codes*.[3] As a result we obtain constructions of binary codes which are efficiently list decodable from extremely large amounts of noise, and which have rate reasonably close to the best possible for such codes. (Prior to our work, there was no known construction of such codes with a positive rate, no matter how low the rate.) We also introduce novel code constructions by combining algebraic list decodable codes with "highly expanding" graphs, and thereby get new list decodable codes which improve these bounds further.

Using an expander-based construction in the same spirit as our construction for list decoding, we also get a significant improvement over a prior result for *unique decoding*. (This shows that techniques developed for list decoding might sometimes also yield new insights towards solving classically studied questions like unique decoding.) Specifically, we prove that for every $\varepsilon > 0$, there are *linear time encodable and decodable* binary codes which can be uniquely decoded up to a fraction $(1/4 - \varepsilon)$ of errors. This fraction of errors is the best possible for unique decoding of binary codes. Therefore, we achieve the maximum possible error-resilience together with the optimal (up to constant factors) encoding and decoding times. In addition, the rate of our codes is also, in a sense that can be made precise, asymptotically the best one could hope for. The only prior known linear time codes, due to Spielman [Spi96], could only correct a tiny constant fraction of errors (of the order of 10^{-6}).

List decoding, while primarily a coding-theoretic notion, has also found applications to other areas of theoretical computer science like complexity theory, cryptography, and algorithms. For these applications unique decoding does not suffice, and moreover, for several of them one needs *efficient* list decoding algorithms. In Chapter 12 of this thesis, we survey some of these "extraneous" applications of list decoding.

Despite its conception more than four decades ago, the long hiatus before efficient algorithms were found means that list decoding is still a subject in its infancy. To the author's best knowledge, this thesis represents the first comprehensive survey of list decoding. Inspite of the length of the thesis, we have attempted a cohesive presentation that hopefully succeeds in highlighting the various aspects of list decoding and how they all fit together nicely. There is lot more work to be done on the subject, and it is our hope that this thesis will inspire at least some of it.

[3] Concatenated codes are obtained by combining two codes. The message is first encoded according to the first code, and then each symbol of the resulting codeword is encoded using the second code. A more detailed description will appear in the next chapter.

1.5 Background assumed of the reader

This thesis faces the situation of having at least two audiences: computer scientists and coding theorists. Hopefully the style of the thesis will be accessible to people with either background. However, the author being a computer scientist by training, the thesis is probably more in line with the language and style of presentation that computer scientists are used to. The only real background required to read the thesis are basic algebra (comfort with finite fields and the like), some amount of probability and combinatorics, etc. Also, the focus of the bulk of the thesis is quite algorithmic, and hence comfort with the analysis of asymptotic complexity of algorithms would be a big plus.

Some portions of the thesis, by the very nature of the topic they discuss, are necessarily somewhat heavy on rather technical and/or algebraic matter. These include: Chapter 4 on the combinatorial limitations of list decoding, the portion of Chapter 6 that deals with algebraic-geometric codes, and Chapter 7 on the decoding of ideal-based codes. In all these cases, we have attempted to clearly state the necessary facts/theorems that we borrow from algebra. Assuming these facts the rest of the presentation should be generally accessible.

Chapter 2

Preliminaries and Thesis Structure

In Galois Fields, full of flowers
primitive elements dance for hours
climbing sequentially through the trees
and shouting occasional parities.

- S.B. Weinstein (IEEE Transactions on Information Theory, March 1971)

In this chapter, we review the basic definitions relating to error-correcting codes and standardize some notation. We then give a brief description of the fundamental code families and constructions that will be dealt with and used in this thesis. Finally, we discuss the structure of the thesis and the main results which are established in the technical chapters that follow, explaining in greater detail how the results of the various chapters fit together.

2.1 Preliminaries and Definitions

In order to avoid introducing too much formalism and notation this early on, we only discuss the most fundamental definitions and will defer a formal treatment of further definitions until they are needed.

2.1.1 Basic definitions for codes

Code, Blocklength, Alphabet size:

Let $q \geq 2$ be an integer, and let $[q] = \{1, 2, \ldots, q\}$.

- An *error-correcting code* (or simply, *code*) C is a subset of $[q]^n$ for some positive integers q, n. The elements of C are called the *codewords* in C.
- The number q is referred to as the *alphabet size* of the code, or alternatively we say that C is a q-ary code. When $q = 2$, we say that C is a *binary* code.
- The integer n is referred to as the *blocklength* of the code C.

Dimension and Rate:

- The *dimension* of a q-ary code C of size $M = |C|$, is defined to be $\log_q M$. (The reason for the term "dimension" will be clear once we discuss linear codes shortly.)

- The *rate* of a q-ary code C of size M, denoted $R(C)$, is defined to be the normalized quantity $\frac{\log_q M}{n}$.

It is often convenient to view a code $C \subseteq [q]^n$ of size M as a function $C : [M] \to [q]^n$. Under this view the elements of $[M]$ are called *messages*, and for a message $x \in [M]$, its *associated codeword* is the element $C(x) \in [q]^n$. Often we will take M to be a perfect power of q, say $M = q^k$, where k is the dimension of the code (this will always be the case, for example, for linear codes which will be discussed shortly). In such a case it is convenient to identify the message space $[M]$ with $[q]^k$, and view messages as strings of length k over $[q]$. Viewed this way, a q-ary error-correcting code provides a systematic way to add redundancy to a string of length k over $[q]$ and encode it into a longer string of n symbols over $[q]$.

(Minimum) Distance and Relative Distance: For strings $\mathbf{x}, \mathbf{y} \in [q]^n$ where $\mathbf{x} = \langle x_1, x_2, \ldots, x_n \rangle$ and $\mathbf{y} = \langle y_1, y_2, \ldots, y_n \rangle$, the Hamming distance between them, denoted $\Delta(\mathbf{x}, \mathbf{y})$, is defined to be the number of coordinates where they differ, that is, the number of i's, $1 \le i \le n$, for which $x_i \neq y_i$.

- The *minimum distance* (or simply *distance*) of a code C, denoted $\mathrm{dist}(C)$, is the minimum Hamming distance between two distinct codewords of C. Formally,

$$\mathrm{dist}(C) = \min_{\substack{c_1, c_2 \in C \\ c_1 \neq c_2}} \Delta(c_1, c_2) \ .$$

- The *relative distance* of a code C, denoted $\delta(C)$, is defined to be the normalized quantity $\frac{\mathrm{dist}(C)}{n}$, where n is the blocklength of C.

<u>Notation</u>: We refer to a general q-ary code of blocklength n, dimension k, and minimum distance d, as an $(n, k, d)_q$-code. We will often omit the distance parameter and refer to a q-ary code of blocklength n and dimension k as an $(n, k)_q$ code. When the alphabet size is clear from the context we will omit the subscript q.[1]

2.1.2 Code families

Since the main thrust of this paper is the *asymptotic* performance of the codes, we define analogs of the quantities above for infinite families of codes. An infinite family of q-ary codes is a family $\mathcal{C} = \{C_i | i \in \mathbb{Z}\}$ where C_i is an $(n_i, k_i)_q$ code with $n_i > n_{i-1}$. We define the *rate* of an infinite family of codes $\mathcal{C}$ to be

$$R(\mathcal{C}) = \liminf_i \left\{ \frac{k_i}{n_i} \right\} .$$

We define the *(relative) distance* of an infinite family of codes $\mathcal{C}$ to be

$$\delta(\mathcal{C}) = \liminf_i \left\{ \frac{\mathrm{dist}(C_i)}{n_i} \right\} .$$

[1]This might appear non-standard to readers already familiar with coding theory who are probably used to the notation $[n, k, d]_q$-code. But this is normally used only for linear codes, and we use $(n, k, d)_q$-code to refer to a general, non-linear code with these parameters. For linear codes, which we define next, we will stick to the standard notation.

Asymptotically good code families

Definition 2.1 *A family $\mathcal{C}$ of codes is said to be* asymptotically good *if both its rate and relative distance are positive, i.e., if $R(\mathcal{C}) > 0$ and $\delta(\mathcal{C}) > 0$.*

By abuse of notation, we will use the phrase "asymptotically good codes" when referring to codes which belong to an asymptotically good code family. The study of the trade-off between the rate and relative distance for asymptotically good codes is one of the main objectives of (asymptotic) combinatorial coding theory.

2.1.3 Linear Codes

Let q be a prime power. Throughout, we denote a finite field with q elements by $\mathbb{F}_q$ or $\mathrm{GF}(q)$ interchangeably. We assume when necessary that the field $\mathbb{F}_q$ can be identified with $[q]$ in some canonical way.

- A *linear* code C of blocklength n is a linear subspace (over some field $\mathbb{F}_q$) of $\mathbb{F}_q^n$.

Clearly, a linear code over $\mathbb{F}_q$ has q^k elements, where k is the dimension of the code as a vector space over $\mathbb{F}_q$. The dimension of a q-ary linear code C is thus the same as its dimension when considered as a vector space over $\mathbb{F}_q$ (hence the terminology "dimension" for the quantity $\log_q |C|$).

As is standard notation, we refer to a q-ary linear code of blocklength n, dimension k and distance d, as an $[n, k, d]_q$ code. We will omit the distance parameter when we do not need to refer to it, and omit the subscript when the alphabet size is clear from the context.

For linear codes, the all-zeroes string is always a codeword. Hence the distance of a linear code equals the minimum *Hamming weight* of a non-zero codeword, where the Hamming weight of a string is defined as the number of coordinates in which it has a non-zero symbol.

An $[n, k]_q$ linear code can be specified in one of two equivalent ways: using the *generator matrix* or the *parity check matrix*.

- An $[n, k]_q$ linear code $\mathbf{C}$ can always be described as the set $\{G\mathbf{x} : \mathbf{x} \in \mathbb{F}_q^k\}$ for an $n \times k$ matrix G; such a G is called a *generator matrix* of $\mathbf{C}$.

- An $[n, k]_q$ linear code $\mathbf{C}$ can also be specified as the subspace $\{\mathbf{y} : \mathbf{y} \in \mathbb{F}_q^n \text{ and } H\mathbf{y} = \mathbf{0}\}$ for an $(n - k) \times n$ matrix H; such an H is called a *parity check matrix* of $\mathbf{C}$.

The above representations of a linear code immediately imply the following for any $[n, k]_q$ linear code:

- (Representation:) It can be succinctly represented using $O(n^2)$ space (by storing either the generator or parity check matrices).

- (Encoding:) A message $\mathbf{x} \in \mathbb{F}_q^k$ can be encoded into its corresponding codeword using $O(nk)$ field operations (by multiplying it with the generator matrix of the code).

The *weight distribution* of a linear code C of blocklength n is defined to be the vector $(A_0, A_1, \ldots, A_n)$, where A_i is the number of codewords of C of Hamming weight i, for $0 \le i \le n$. Note that $A_0 = 1$, and if d is the distance of C, then $A_1, A_2, \ldots, A_{d-1} = 0$.

Given a linear code $C \subseteq \mathbb{F}_q^n$, one can define a relation, say $\sim$, between elements of $\mathbb{F}_q^n$ as follows: $\mathbf{y} \sim \mathbf{z}$ iff $\mathbf{y} - \mathbf{z} \in C$. Since the code is linear, it is easy to check that this defines an equivalence relation. Consequently, it defines a partition of the space $\mathbb{F}_q^n$ into equivalence classes. These equivalence classes are called the *cosets* of the code C.[2] One of these cosets will be the code C itself. The weight distribution of cosets of a linear code in fact provide detailed information about the combinatorial list decodability properties of a code. For sake of simplicity though, we state and prove all our combinatorial results using only the language of list decoding (which we shortly develop in Section 2.1.4).

Additive Codes: A class of codes that lie in between linear and general non-linear codes in terms of "structure" are *additive codes.* These are codes over $\mathbb{F}_q$ which are closed under codeword addition; i.e., if x and y are codewords then so is $x+y$. (For linear codes, we will have the additional property that if x is a codeword then so is αx for every $\alpha \in \mathbb{F}_q$ — here αx stands for the string obtained by coordinate-wise multiplication of x by α.) Note that for binary codes, additive codes define the same class as linear codes.

2.1.4 Definitions relating to List Decoding

Recall that under list decoding, the aim, given a received word, is to output a list of all codewords that lie within a Hamming ball of certain radius around the received word. The radius of the ball corresponds to the number of errors corrected by the list decoding procedure. Hence it is of interest to quantify the maximum number of codewords in a ball of certain radius, or equivalently, to quantify the largest number of errors that can be list decoded with lists of a certain size. We do this by defining the "list decoding radius" of a code below.

Let $q \geq 2$ be the alphabet size of a code C of blocklength n. For a non-negative integer r and $\mathbf{x} \in [q]^n$, let $B_q(\mathbf{x}, r)$ denote the Hamming ball of radius r around $\mathbf{x}$, i.e.,

$$B_q(\mathbf{x}, r) = \{\mathbf{y} \in \mathbb{F}_q^n \mid \Delta(\mathbf{x}, \mathbf{y}) \leq r\} .$$

For the case $q = 2$, we will usually omit the subscript and refer to such a ball as simply $B(\mathbf{x}, r)$.

Definition 2.2 ((e, L)-list decodability) *For positive integers e, L, a code $C \subseteq \mathbb{F}_q^n$ is said to be* (e, L)-list decodable *if* every *Hamming ball of radius e has at most L codewords, i.e.* $\forall\, \mathbf{x} \in \mathbb{F}_q^n$, $|B_q(\mathbf{x}, e) \cap C| \leq L$.

Definition 2.3 (List Decoding Radius) *For a code C of blocklength n and an integer $L \geq 1$, the* list of L decoding radius *of C, denoted $\mathsf{radius}(C, L)$ is defined to be the* maximum *value of e for which C is (e, L)-list decodable. We also define the normalized list-of-L decoding radius, denoted $\mathsf{LDR}_L(C)$, as*

$$\mathsf{LDR}_L(C) = \frac{\mathsf{radius}(C, L)}{n} .$$

As before we would like to extend this definition for families of codes, since our aim is to study the asymptotic performance of codes. To do this, it makes sense to allow the list size to be a function of the blocklength. Accordingly we have the following definition.

[2] This terminology is borrowed from group theory, and the cosets of C defined above are precisely the group-theoretic cosets of C when it is viewed as an additive subgroup of $\mathbb{F}_q^n$.

Definition 2.4 [List Decoding Radius for code families] *For an infinite family of codes* $\mathcal{C} = \{C_i\}_{i \geq 1}$ *where* C_i *has blocklength* n_i*, and a function* $\ell : \mathbb{Z}^+ \to \mathbb{Z}^+$*, define the* list of ℓ decoding radius *of* $\mathcal{C}$*, denoted* $\mathsf{LDR}_\ell(\mathcal{C})$*, to be*

$$\mathsf{LDR}_\ell(\mathcal{C}) = \liminf_i \left\{ \frac{\mathsf{radius}(C_i, \ell(n_i))}{n_i} \right\}.$$

When ℓ *is the constant function that takes on the value* L *on every input blocklength, we denote* $\mathsf{LDR}_\ell(\mathcal{C})$ *as simply* $\mathsf{LDR}_L(\mathcal{C})$*.*

Remark: It will be clear from the context whether the LDR function is being applied to a code or to a code family, and also whether it is applied to a constant list size or to a list size which is a growing function of the blocklength.

Some "informal" usages:

Sometimes we also refer to the phrase ***"list decoding radius"*** without an explicit mention of the list size. In such cases we imply the list decoding radius for a list size which is some polynomially growing function of the blocklength, i.e., for $\ell(n) = n^c$ for some constant c (in fact, in almost every such reference in this thesis setting $c = 2$ will suffice).

We will also use the adjectives "list decodable up to a fraction α of errors" or "list decodable up to (relative) radius α" to refer to codes or code families whose list decoding radius is at least α. We will say a list decoding algorithm can *"correct"* a fraction α of errors (or e errors), if it can perform list decoding up to a fraction α of errors (or up to a radius of e).

2.1.5 Commonly used notation

Much of the notation we use is standard. Throughout the thesis both $\log x$ and $\lg x$ will denote the logarithm of x to the base 2. We denote the natural logarithm of x by $\ln x$. For bases other than 2 and e, we explicitly include the base in the notation; for example logarithm of x to the base q will be denoted by $\log_q x$.

For a real number x, $\lfloor x \rfloor$ will denote the largest integer which is at most x, and $\lceil x \rceil$ will denote the smallest integer which is at least x.

For x in the range $0 \leq x \leq 1$, and an integer $q \geq 2$, we denote by $H_q(x)$ the *q-ary entropy function*, i.e., $H_q(x) = x \log_q(q-1) - x \log_q x - (1-x) \log_q(1-x)$. When $q = 2$, we denote the binary entropy function $H_2(x)$ as simply $H(x)$.

For a finite set S, we denote the number of elements that belong to S by $|S|$.

2.2 Basic Code Families

In this section, we describe the central code families which will be studied in this thesis and also be used as building blocks in several of the new code constructions contributed by this thesis.

2.2.1 Reed-Solomon codes

Reed-Solomon codes are an extremely important and well-studied family of linear codes. They are based on the properties of univariate polynomials over finite fields. Formally, an

$[n, k+1]_q$ Reed-Solomon code, with $k < n$ and $q \geq n$, is defined as follows. Let $\alpha_1, \alpha_2, \ldots, \alpha_n$ be n *distinct* field elements in $\mathbb{F}_q$ (since $q \geq n$, it is possible to pick such α_i's). The message space consists of polynomials $p \in \mathbb{F}_q[x]$ with degree at most k, and a "message" p is encoded as:

$$p \mapsto \langle p(\alpha_1), p(\alpha_2), \ldots, p(\alpha_n) \rangle .$$

Note the message space can be identified with $\mathbb{F}_q^{k+1}$ in the obvious way: view $\langle m_0, m_1, \ldots, m_k \rangle$ as the polynomial $m_0 + m_1 x + \ldots + m_k x^k$.

The following basic proposition follows from the well-known fact from algebra that two degree k polynomials over a field can agree on at most k places.

Proposition 2.1 *The above code is an* $[n, k+1, d = n-k]_q$ *code.*

The singleton bound in coding theory says that the sum of the distance and dimension of a code can be at most $n+1$, where n is the blocklength of the code (cf. [vL99, Section 5.2]). Hence, Reed-Solomon codes "match" the singleton bound. Such codes are called *Maximum Distance Separable* (MDS), since they have the maximum possible distance for a given blocklength and dimension. The MDS property together with the nice algebraic structure of Reed-Solomon codes that facilitates the design of efficient decoding algorithms, have made it one of the most fundamental code families. Reed-Solomon codes have found a wide variety of applications in coding theory and computer science, as well as several applications in the "real world" – examples include compact disc players, disk drives, satellite communications, and high-speed modems such as ADSL, to name a few (see [WB99] for detailed information on the various applications of Reed-Solomon codes).

2.2.2 Reed-Muller codes

Reed-Muller codes are a generalization of Reed-Solomon codes obtained by taking for message space all ℓ-variate polynomials over some finite field $\mathbb{F}_q$ with total degree at most m, subject to the condition that no variable takes on a degree of q or more. A polynomial is again encoded by evaluating it at n distinct elements of $\mathbb{F}_q^\ell$, where n is the blocklength of the code (note that this requires $n \leq q^\ell$). Setting $\ell = 1$ we get the construction of Reed-Solomon codes. The degree parameter m is often referred to as the *order* of the Reed-Muller code. Reed-Muller codes are clearly linear codes. When $m < q$, their dimension equals $\binom{m+\ell}{m}$, and using what is now famous as the Schwartz-Zippel Lemma, it follows that their relative distance is at least $(1 - m/q)$.[3]

Hadamard codes

Of special interest are Reed-Muller codes of order 1, i.e., codes based on *multilinear* polynomials, also known as simplex codes (a detailed discussion of these codes appears in [MS81, Chap. 14]). A variant of these, based on homogeneous polynomials with no constant term, are commonly referred to as *Hadamard codes*. Formally, a Hadamard code of dimension ℓ over $\mathbb{F}_q$ is defined as follows. A message $\mathbf{x} \in \mathbb{F}_q^\ell$ is mapped to the string $\langle \mathbf{x} \cdot \mathbf{z} \rangle_{\mathbf{z} \in \mathbb{F}_q^\ell}$ of length q^ℓ (here by $\mathbf{x} \cdot \mathbf{z}$ we mean the dot product of the vectors $\mathbf{x}$ and $\mathbf{z}$ over the field $\mathbb{F}_q$). The Hadamard code thus has very poor rate since it maps ℓ symbols over $\mathbb{F}_q$ into q^ℓ symbols. But it has very good distance properties — its relative distance equals $(1 - 1/q)$,

[3] When $m \geq q$, in general there is no simple closed form for the dimension, and the relative distance is at least $q^{-\lceil m/q \rceil}$.

and in fact *every* non-zero codeword has Hamming weight equal to $(q^\ell - q^{\ell-1})$. Despite its poor rate, its highly structured distance properties makes it an attractive code for use at the inner level in certain concatenation schemes. Indeed, several of our concatenated code constructions in later chapters use a suitable Hadamard code as an inner code.

2.2.3 Algebraic-geometric codes

Algebraic-geometric codes (or AG-codes, for short) are also a generalization of Reed-Solomon codes. Reed-Solomon codes may be viewed as evaluations of certain functions at a subset S of points on the projective line over $\mathbb{F}_q$ — the functions are those that have a bounded number of "poles" at a certain point that is designated as the "point at infinity" and no poles elsewhere (this corresponds precisely to low-degree polynomials), and the code can be defined based on any subset S of points that does not include the point at infinity. AG-codes are a generalization based on any "nice" algebraic curve playing the role of the projective line. Let Γ be such a curve. Every such curve has an associated *function field* which, roughly, is the set of all "valid" functions that can be evaluated at points on Γ. To construct an AG-code based on Γ, one picks a point P_∞ on the curve and a set S of points on Γ disjoint from $\{P_\infty\}$. The message space of the code will be all functions in the function field of Γ that have a bounded number of poles at P_∞ and no poles elsewhere, and such a function will be encoded by evaluating it at each of the points in S. The precise definition of AG-codes requires a reasonable amount of background in the theory of algebraic function fields and curves, and this will be developed in Chapter 6 where we will give a list decoding algorithm for AG-codes.

2.2.4 Concatenated codes

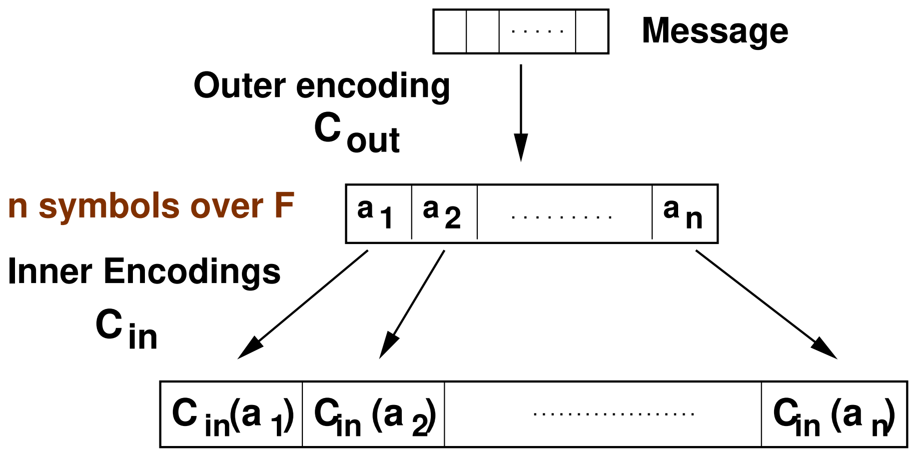

Figure 2-1: Code concatenation. If the outer code C_{out} is over an alphabet F, and the inner code C_{in} has exactly $|F|$ codewords corresponding to the $|F|$ symbols of F, there is a natural way to combine them by "concatenation".

Concatenated coding gives a way to combine two codes, an *outer* code C_{out} over a large alphabet (say $[Q]$), and an *inner* code C_{in} with Q codewords over a small(er) alphabet (say, $[q]$), to get a combined q-ary code that, loosely speaking, inherits the good features of both the outer and inner codes. These were introduced by Forney [For66a] in a classic and seminal work. The basic idea is very natural (see the illustration in Figure 2-1): to encode a message using the *concatenated code*, we first encode it using C_{out}, and then in turn encode each of the resulting symbols (which all belong to $[Q]$) into the corresponding codeword of C_{in}. Since there are exactly Q codewords in C_{in}, the encoding procedure is well defined.

The rate of the concatenated code is the product of the rates of the outer and inner codes, and the distance is at least as large as the product of the distances of the outer and inner codes. The product of the distances of the outer and inner codes is called the *designed distance* of the concatenated code. Thus, concatenated codes have good rate and distance if the outer and inner codes have good rate and distance.

The big advantage of concatenated codes for us is that we can get a good list decodable code over a small alphabet (say, binary codes) based on a good list decodable outer code (like a Reed-Solomon or AG-code) and a "suitable" binary inner code. The dimension of the inner code is small enough to permit a brute-force search for a "good" code in reasonable time. Code concatenation forms the basis of all our code constructions in Chapters 8, 9 and 10, and is a heavily used tool in this thesis.

2.2.5 Number-theoretic codes

The thesis also discusses number-theoretic codes which are based on a similar algebraic principle to the one underlying the construction of Reed-Solomon and AG-codes.

Chinese Remainder codes

Chinese Remainder codes (or CRT codes, for short), also called Redundant Residue codes, are the number-theoretic analogue of Reed-Solomon codes. The messages of the CRT code are integers in $\{0, 1, \ldots, K-1\}$ for some K, and a message m, $0 \le m < K$, is encoded as

$$m \mapsto \langle m \bmod p_1, m \bmod p_2, \ldots, m \bmod p_n \rangle$$

for n relatively prime integers $p_1 < p_2 < \cdots < p_n$. If k is such that $\prod_{i=1}^{k} p_i > K$, then by the Chinese Remainder theorem (hence the name of the code), the residues of m modulo any k of the p_i's uniquely specifies m. Hence any two codewords (corresponding to encodings of m_1, m_2 with $m_1 \neq m_2$) differ in at least $(n-k+1)$ positions. Thus, the distance of the code is at least $(n-k+1)$.

Number field codes

Number field codes are the number-theoretic analogues of AG-codes, and generalize CRT codes akin to the way AG-codes generalize Reed-Solomon codes. The code is based on a suitable "number field" (i.e., a finite extension of the field $\mathbb{Q}$ of rational numbers) and the associated "ring of integers" R. A formal description of these codes will take us too far afield from the main thrust of this thesis. Hence we do not discuss these codes in this thesis; the interested reader is pointed to [Len86, Gur01c] for formal definitions of these codes and details on their properties.

2.3 Results and Structure of this thesis

This thesis presents a thorough and comprehensive investigation of the notion of list decoding. It deals both with fundamental combinatorial questions relating to list decoding and the algorithmic aspects of list decoding. It also discusses a few applications of list decoding both within coding theory (to questions not directly concerned with list decoding) and to certain complexity-theoretic and algorithmic questions outside coding theory.

Though the questions addressed are all intimately related, for purposes of exposition and because they permit such modularity, we structure the results of this thesis into three parts: Combinatorial Results (Part I), Algorithms and Code Constructions (Part II), and Applications (Part III).

The combinatorial results of Part I set the stage for the algorithmic results by highlighting what one can and cannot hope to do with list decoding. The bottom line of Part I can be viewed as the statement: *Good codes with excellent combinatorial list decodability properties exist.*

The algorithmic results attempt to "match" the combinatorial bounds with explicit code constructions and efficient decoding algorithms. These include algorithms for classical and well-studied codes like Reed-Solomon and algebraic-geometric codes, as well as for certain novel code constructions based on suitable concatenation schemes. The bottom line of Part II can be viewed as: *There exist "explicit" constructions of "good" codes together with efficient list decoding algorithms.*

In Part III, we discuss some applications of the results and techniques from earlier chapters to domains both within and outside of coding theory. The notion of list decoding turns out to be central to certain contexts outside of coding theory, for example to several complexity-theoretic questions. These and several other applications are discussed in Part III of the thesis.

We now discuss the results of each of these parts in further detail.

2.3.1 Combinatorial Results

Chapter 3 — The Johnson Bound on List Decoding Radius. We argued in the introduction that unique/unambiguous decoding is not possible when the number of errors exceeds half the minimum distance (say, $d/2$) of the code. The purpose of list decoding is to allow for meaningful recovery when the number of errors exceeds this bound. But for list decoding to be meaningful, and definitely for it to be algorithmically feasible, one needs the guarantee that one can correct many more than $d/2$ errors with fairly *small* lists (say, of size a fixed constant, or a fixed polynomial in the blocklength). In this chapter, we revisit a classical bound from coding theory called "Johnson bound", present extensions of it, and apply it to the context of list decoding. The bound demonstrates that one can always correct more than $d/2$ errors with "small" lists – the exact number of errors to which the bound applies is an explicit function of the distance of the code, and we call this the *Johnson bound on list decoding radius*. One way to view these results is that one can construct good list decodable codes by constructing codes with large minimum distance. There are several proofs known for Johnson-type bounds in the literature – the proof presented in this chapter appears in [GS01a].

Chapter 4 — Limits to List Decodability. We address the natural question raised by the results of Chapter 3 – namely whether the Johnson bound is "tight", that is, whether the Johnson bound is the best possible bound on the list decoding radius (purely as a function of the distance of the code). For general, non-linear codes, it is easy to show that the Johnson bound is indeed tight as a general trade-off between list decoding radius and distance. The more interesting case of linear codes, however, turns out to be significantly harder to resolve, and is the subject of this chapter. We present constructions of linear codes of good distance with several codewords in a "small" Hamming ball. Under a widely believed number-theoretic conjecture (which in particular is implied by a suitably generalized Riemann Hypothesis), we prove that the Johnson bound is indeed a "tight" bound on the list decoding radius (for decoding with polynomial sized lists). We prove such a result unconditionally for list decoding with constant-sized lists. We also prove that the list decoding radius for any fixed polynomial sized list is bounded away from the minimum distance of the code. The results of this chapter are a combination of results from [GHSZ00, JH01, Gur01a].

Chapter 5 — List decodability Vs. Rate. The results of the earlier chapters show that one way to get codes with large list decoding radius is to use codes with large minimum distance. But if our main concern is list-of-L decoding (for some list size L), then is this "two-step" route the best way to get good list decodable codes? The answer turns out to be no, and in this chapter we show that one can achieve a much better rate by directly optimizing the list-of-L decoding radius, than by going through the minimum distance (and using the Johnson bound on list decoding radius). Our results employ the probabilistic method, and are thus non-constructive. Nevertheless, these results set the stage for the algorithmic results of Part II, by highlighting the kind of parameters one can hope for in efficiently list decodable codes. Moreover, for small enough blocklengths, these "good" codes can be found by brute-force search, and this is exploited in our concatenated code constructions. The results in this chapter are a combination of results from [ZP82, GHSZ00, GI01b].

Part I: Summary. The combinatorial results provide a fairly precise understanding of the general trade-off between the list decoding radius of a code, and the more traditional parameters like rate and minimum distance of a code. The Johnson bound asserts that codes with large minimum distance have large list decoding radius, which raises algorithmic questions on list decoding such codes from a large number of errors. This is not the only approach to get good list decodable codes, however, as directly optimizing the list decoding radius can lead to better trade-offs as a function of the rate of the code.

2.3.2 Algorithmic results

Even though the notion of list decoding originated more than 40 years ago [Eli57, Woz58], and some of its combinatorial and information-theoretic aspects (relating to channel capacity under list decoding) received attention, until recently no *efficient* list decoding algorithms were known for any (non-trivial) family of codes that could correct asymptotically more errors than the traditional half the distance bound. Part II of the thesis presents polynomial time list decoding algorithms for several classical families of codes as well as several new constructions of codes that have very efficient list decoding algorithms. Details of the specific chapters and the results therein follow.

Chapter 6 — Reed-Solomon and Algebraic-geometric Codes. We present an efficient algorithm to list decode the important class of Reed-Solomon codes up to the Johnson bound on list decoding radius. Among other things this is the first algorithm to decode

Reed-Solomon codes beyond half the distance for *every* value of the rate. This algorithm was obtained in joint work with Madhu Sudan [GS99], and it builds upon the earlier works by Sudan [Sud97a] and Ar *et al* [ALRS99]. We also present a "weighted" version of the decoding algorithm which can take "soft" inputs – this is a very useful subroutine in soft-decision decoding of Reed-Solomon codes [KV00] and in decoding various concatenated codes. We also present a generalization of the algorithm to list decode algebraic-geometric codes, following some ideas from the earlier work of [SW99]. The family of algebraic-geometric codes are more general than Reed-Solomon codes, and for large enough alphabets contain codes with the best known asymptotic trade-off between the rate and relative distance.

Chapter 7 — Unified Paradigm for List Decoding. We present a unified description of several known algebraic codes including Reed-Solomon, algebraic-geometric and Chinese Remainder (CRT) codes in the language of rings and ideals. We also present a unified list decoding algorithm for ideal-based codes which encompasses and generalizes the algorithms from Chapter 6. As a corollary, we extract an algorithm for list decoding CRT codes up to (almost) the Johnson bound on list decoding radius (suitably adapted to the case of the CRT codes). The unified paradigm emerging out of this study could be of independent interest. These results are based on joint work with Amit Sahai and Madhu Sudan [GSS00].

Chapter 8 — List Decoding of Concatenated Codes. The results of the previous chapters apply to codes over large alphabets (algebraic-geometric codes exist over small alphabets, but their list decodability is limited by certain barriers based on some deep results from algebraic geometry). It is natural to ask if there are codes over fixed small alphabets, say binary codes for concreteness, which can be list decoded efficiently from a large fraction of errors. It turns out that the earlier results for Reed-Solomon and algebraic-geometric codes play a critical role in answering this question — using them as outer codes in suitable concatenation schemes yields constructions of binary codes of good rate and good list decodability. In particular, we present a polynomial time construction of binary codes of rate $\Omega(\varepsilon^4)$ that are list decodable in polynomial time from a fraction $(1/2 - \varepsilon)$ of errors (for $\varepsilon > 0$ as small a constant as we desire). This construction uses a combination of the algorithmic results from Chapters 6 and the combinatorial results from Chapter 5. The material from this chapter is a collection of results from [GS00, GHSZ00], together with some results that appear for the first time in this thesis.

Chapter 9 — New, Expander-based List Decodable Codes. It follows from the results of Chapter 6 (on Reed-Solomon and algebraic-geometric codes) that there are rate $\Omega(\varepsilon^2)$ codes that can be efficiently list decoded up to a fraction $(1 - \varepsilon)$ of errors. Reed-Solomon codes are defined over a large, growing alphabet size, while algebraic-geometric achieve a constant (in fact $\text{poly}(1/\varepsilon)$) alphabet size, but suffer from complicated and inefficient constructions and decoding. It is natural to ask if there is a "better" construction of codes that are list decodable up to a fraction $(1-\varepsilon)$ errors. This chapter answers this question and presents a novel expander-based construction of such codes over a constant-sized alphabet, along with a simple, near-quadratic time decoding procedure. Furthermore, we know from Chapter 5 that, non-constructively, a rate of $\Omega(\varepsilon)$ is feasible for codes with list decoding radius of $(1-\varepsilon)$. Using our basic construction, together with some other ideas, we are able to construct codes of the optimal $\Omega(\varepsilon)$ rate that are list decodable up to a fraction $(1-\varepsilon)$ of errors in *sub-exponential* time. This is the first construction to beat the "ε^2-barrier" on rate and approach the optimal rate in a meaningful way. This chapter also introduces several tools for code constructions such as pseudolinear codes, multi-concatenated codes, and juxtaposed codes, which are interesting in their own right. The material in this chapter is based on joint work with Piotr Indyk [GI01b].

Chapter 10 — List Decoding from Erasures. All prior chapters dealt with the model where a certain fraction of the codeword symbols are adversarially corrupted. A weaker noise model is that of *erasures* where a certain adversarially chosen fraction of the codeword symbols are erased by the channel. While this is an easier model to deal with, it also enables achieving better trade-offs and parameters. We prove combinatorial results akin to those of Chapter 5 specialized for the case of erasures, and then use techniques similar to those used in Chapters 8 and 9 to construct codes with good (and sometimes near-optimal) rate and good erasure list decodability. A side consequence of one of the results in this chapter is a provable asymptotic separation between the performance of linear and general, non-linear codes (with respect to erasure list decodability). Such an asymptotic separation is quite rare in coding theory. The material in this chapter appears in the papers [Gur01b, GI01a].

Part II: Summary. The algorithmic results of the above chapters show that for several important and useful code families, there is an efficient algorithm to list decode them up to (close to) the Johnson bound on list decoding radius. These codes are defined over a large alphabet. However, one can use them as outer codes in concatenated schemes together with suitable inner codes that have list decodability properties similar to those guaranteed by the combinatorial results (from Chapter 5). This enables us to get new constructions of binary codes of good rate and excellent algorithmic list decodability.

2.3.3 Applications

Chapter 11 — Linear-time codes for unique decoding. This chapter uses techniques similar to previous chapters, specifically Chapter 9, to build codes of very good rate together with extremely efficient unique/unambiguous decoding algorithms up to the "maximum" possible radius. Specifically, we construct codes with the optimal $\Omega(\varepsilon)$ rate that can be unambiguously decoded up to a fraction $(1/2 - \varepsilon)$ of errors in linear time. We then concatenate these codes with suitable inner codes to get binary codes of rate $\Omega(\varepsilon^3)$ that can be decoded from a fraction $(1/4 - \varepsilon)$ of errors in linear time. Our codes can also be encoded in linear time. These linear-time codes significantly improve the fraction of errors corrected by the earlier linear-time codes due to Spielman [Spi96]. Our codes are obtained by using Spielman's codes as a building block and then boosting its error-resilience using suitable expander graphs. The results in this chapter are based on joint work with Piotr Indyk [GI01a, GI01b].

Chapter 12 — Sample Applications outside Coding Theory. We discuss some sample applications of list decoding outside coding theory. We present an algorithmic application to the problem of guessing secrets, which is a variant of the "20 questions" game played with more than one secret. List decoding has found several compelling applications in complexity theory and we discuss some of these including hardcore predicate constructions, hardness amplification of boolean functions, constructions of extractors and pseudorandom generators, inapproximability of NP witnesses, etc. The chapter also discusses some applications of list decoding to cryptographic questions such as cryptanalysis of certain block ciphers, finding smooth integers, and traitor tracing.

2.3.4 Conclusions

Chapter 13 — Concluding Remarks. We conclude with a brief summary of the contributions of the thesis and discuss some open questions and possible directions for future research.

2.3.5 Dependencies among chapters

A pictorial depiction of the interdependencies among the various technical chapters is presented in Figure 2-2.

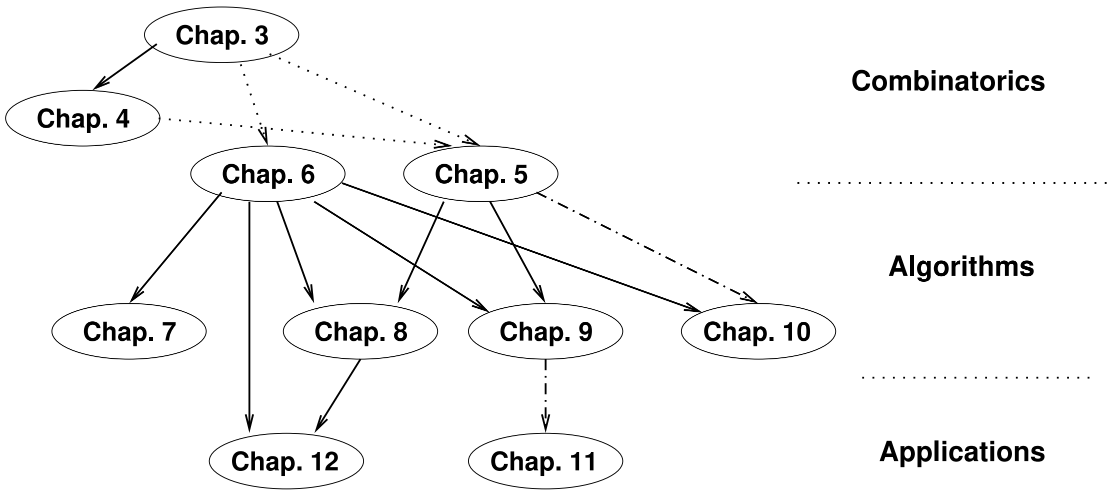

Figure 2-2: The interrelationship between various chapters. A solid line indicates a dependency (in either techniques or results themselves). A dashed arrow from A to B indicates a "soft" dependency; i.e., reading portions of A prior to B would be helpful, but is not strictly necessary. A dotted line from A to B that results of chapter A "motivate" the contents of chapter B, though there is no real dependency in the results or techniques themselves.

We would like to point out that the separation of the combinatorial and algorithmic results in this thesis is not a strict one. We only isolate the most basic combinatorial results in Part I, namely those results which are interesting independent of whether there are algorithmic results or not (though they do end up motivating and being used in several of the algorithms in Part II anyway). Some combinatorial results can also be found in Part II. In all such cases, due to the somewhat "local" nature of their application, we chose to defer the presentation of the concerned combinatorial results to the point where they are actually needed. Examples of such combinatorial results discussed in Part II include: a version of the Johnson bound in Chapter 7 when the various codeword positions have different contributions towards the minimum distance (this happens for the Chinese Remainder code), a Johnson-type bound in Chapter 8 concerning the coset weight distribution of codes as a function of the distance of the code, an existence result for codes whose coset weight distribution has a certain property in Chapter 8, results concerning pseudolinear codes in Chapters 9 and 10, and combinatorial bounds and existence results concerning erasure list decodable codes in Chapter 10.

Part I

Combinatorial Bounds

Chapter 3

Johnson-type Bounds and Applications to List Decoding

Knowledge is of two kinds. We know a subject ourselves,
or we know where we can find information upon it.
James Boswell, in "The Life of Samuel Johnson"

This chapter, as well as the next one, explore the relation between the list decoding radius and minimum distance of a code. Understanding the relation between these parameters is useful for two reasons: (a) for several important families of codes like Reed-Solomon codes, we have precise bounds on the distance, and one can use the relation between list decoding radius and distance to understand the list decoding potential of these codes; and (b) this shows that one approach to construct good list decodable codes is to construct large distance codes, and the latter is a relatively well-studied and better understood problem. Also, historically the most significant algorithmic results on list decoding have been fueled by an attempt to decode codes whose good minimum distance highlighted their good combinatorial list decodability properties.

3.1 Introduction

In order to perform list decoding up to a certain number, say e, errors efficiently, we need the guarantee that every Hamming ball of radius e has a "small" number of codewords. This is because the list decoding algorithm will have a runtime that is at least the size of the list it outputs, and we want the algorithm to be efficient even for the worst-case error pattern. The exact size of the list can be either set to a suitably large constant (independent of the blocklength), or to a fixed polynomial function of the blocklength.

Unique decoding is based upon the fact that in a code of minimum distance d any Hamming ball of radius less than $d/2$ can have at most one codeword. For list decoding we would like upper bounds on the number of codewords in a ball of radius e for e larger than $d/2$. A classical bound in coding theory, called the Johnson bound [Joh62, Joh63] (see also [MS81]), proves an upper bound on the number of codewords at a Hamming distance exactly e from an arbitrary word, as long as e is less than a certain function of the distance and blocklength of the code. Such a bound is of direct interest to constant-weight codes (which are codes all of whose codewords have the same Hamming weight), and is also used in the Elias-Bassalygo upper bound on the dimension of codes with certain minimum distance.

For purposes of list decoding, we need a Johnson-style bound for the number of codewords at a distance of at most e (not exactly e) from a received word. In this chapter, we present a very general version of such a bound. Owing to their strong resemblance to the Johnson bound, we call our bounds Johnson-type (or simply, Johnson) bounds. The main result of this chapter is the fact any q-ary code of blocklength n and distance d is list decodable with "small" lists for up to $e_J(n, d, q)$ errors, where $e_J(n, d, q)$ is a function only of n, d, q (and *not* the structure of the code). We call this quantity $e_J(n, d, q)$ the "Johnson bound on list decoding radius" or "Johnson radius" of the code, and it is always greater than $d/2$.

Proofs of the Johnson bound seem to come in one of two flavors. The original proof and some of its derivatives follow a linear algebra based argument [Joh62, Joh63, Eli91, GRS95, GS00], while more recent proofs, most notably [Lev98, EZ95, AVZ00] are more geometric. Our proof follows the latter spirit, extending these proofs to the case of general alphabets.

Moreover, our techniques easily allow us to extend our results and also prove a weighted version of the Johnson bound which is of interest to some questions raised by the investigations on "soft" list decoding algorithms (more details on this and the connection to soft decoding will be discussed in later chapters in Part II of the thesis).

3.2 Definitions and Notation

We first recall some notation. For $\mathbf{x}, \mathbf{y} \in [q]^n$ the Hamming distance between $\mathbf{x}$ and $\mathbf{y}$ is denoted $\Delta(\mathbf{x}, \mathbf{y})$. For $\mathbf{r} \in [q]^n$ and $0 \le e \le n$, the Hamming ball of radius e around $\mathbf{r}$ is defined by $B_q(\mathbf{r}, e) = \{\mathbf{x} \in [q]^n : \Delta(\mathbf{r}, \mathbf{x}) \le e\}$.

The key quantity to study in our context is the following. Let $A'_q(n, d, e)$ denote the maximum number of points that may be placed in some ball $B_q(\mathbf{r}, e)$ such that all pairwise distances between the points are at least d. More formally,

$$A'_q(n, d, e) = \max\{|S| : S \subseteq B_q(\mathbf{r}, e) \text{ for some } \mathbf{r} \in [q]^n \text{ and } \forall \mathbf{x}, \mathbf{y} \in S,\ \Delta(\mathbf{x}, \mathbf{y}) \ge d\}\ . \quad (3.1)$$

(We use the notation $A'_q(n, d, e)$ instead of the apparently more natural choice $A_q(n, d, e)$ because the notation $A_q(n, d, e)$ in coding theory literature normally refers to the maximum number of points (with pairwise distances at least d) that may be placed **on** the surface of (instead of within) the ball $B_q(\mathbf{r}, e)$. To avoid confusion with this standard terminology, we use $A'_q(n, d, e)$ instead. We clearly have $A_q(n, d, e) \le A'_q(n, d, e)$, and thus any upper bound we derive on $A'_q(n, d, e)$ also applies to $A_q(n, d, e)$.)

Clearly for any code $\mathcal{C} \subseteq [q]^n$ of minimum distance d, $A'_q(n, d, e)$ is an upper bound on the number of codewords of $\mathcal{C}$ that can lie in a Hamming ball of radius e. Hence, our objective in this chapter is to obtain an upper bound on the function $A'_q(n, d, e)$.

It is common practice to denote these functions as $A(n, d, e)$ and $A'(n, d, e)$ for the binary $(q = 2)$ case.

3.3 The Johnson bound on list decoding radius

Theorem 3.1 ([GS01a, AVZ00]) *Let $\mathcal{C}$ be any q-ary code of blocklength n and minimum distance $d = (1 - 1/q)(1 - \delta)n$ for some $0 < \delta < 1$. Let $e = (1 - 1/q)(1 - \gamma)n$ for some $0 < \gamma < 1$ and let $\mathbf{r} \in [q]^n$ be arbitrary. Then, provided $\gamma > \sqrt{\delta}$, we have*

$$|B_q(\mathbf{r}, e) \cap \mathcal{C}| \le \min\Big\{n(q-1), \frac{1-\delta}{\gamma^2 - \delta}\Big\}\ . \quad (3.2)$$

Furthermore, for the case when $\gamma = \sqrt{\delta}$, we have $|B_q(\mathbf{r}, e) \cap \mathcal{C}| \le 2n(q-1) - 1$.

The theorem below is merely a restatement of the above result in different notation, and follows immediately from the above result (it is a straightforward calculation to check this).

Theorem 3.2 *Let q, n, d be arbitrary positive integers with $d < (1 - 1/q)n$.*

(i) Let $e \ge 1$ be any integer that satisfies the condition

$$e < e_J(n,d,q) \stackrel{\text{def}}{=} \Big(1 - \frac{1}{q}\Big)\Big(1 - \sqrt{1 - \frac{q}{q-1} \cdot \frac{d}{n}}\Big)n \ . \tag{3.3}$$

Then we have

$$A'_q(n,d,e) \le \min\{n(q-1), \frac{nd}{nd - 2e(n - \frac{qe}{2(q-1)})}\} \ . \tag{3.4}$$

In other words, for an integer $L \ge 1$, if

$$e \le e_J(n,d,q,L) \stackrel{\text{def}}{=} n\Big(1 - \frac{1}{q}\Big)\Big(1 - \sqrt{1 - \frac{q}{q-1} \frac{L-1}{L} \frac{d}{n}}\Big) \ , \tag{3.5}$$

then $A'_q(n,d,e) \le L$.

(ii) Furthermore, if $e = e_J(n,d,q)$, then $A'_q(n,d,e) \le 2n(q-1) - 1$.

The above theorem says that a q-ary code of blocklength n and distance d can be list decoded with small lists for up to $e_J(n,d,q)$ errors. For purposes of easy future reference, we give the quantity $e_J(n,d,q)$ the label *"Johnson bound on list decoding radius"* , or simply the *"Johnson radius"* of a code. When we want to make the alphabet size explicit, we will refer to $e_J(n,d,q)$ as the *"q-ary Johnson radius"*. For decoding with lists of size L, we give the quantity $e_J(n,d,q,L)$ the label *"Johnson radius for list-of-L decoding"*.

It is easy to verify that the Johnson radius $e_J(n,d,q)$ defined in Equation (3.3) satisfies

$$e_J(n,d,q) > d/2$$

for every n, d, q with $1 \le d \le (1 - 1/q)n$. This captures the claim that *list decoding with polynomial-sized lists always permits one to decode beyond half the distance.* As an illustration, we plot the Johnson radius for binary codes in Figure 3-1, normalized by blocklength, as a function of the relative distance of the code. Note for any every value of the relative distance δ in the range $0 < \delta < 1/2$, the Johnson radius is strictly greater than half the minimum distance.

Before moving on to the proof of Theorem 3.1, we state the following corollary to the above statement. This gives a (weaker) version of the above bounds that ignores the alphabet size q of the code. But it has a simpler, easily stated form, and for large q approaches the above bounds.

Corollary 3.3 *Let q, n, d, e be arbitrary positive integers with $e \le d \le n$.*

(i) If $e < n - \sqrt{n(n-d)}$, then $A'_q(n,d,e) \le n(q-1)$.

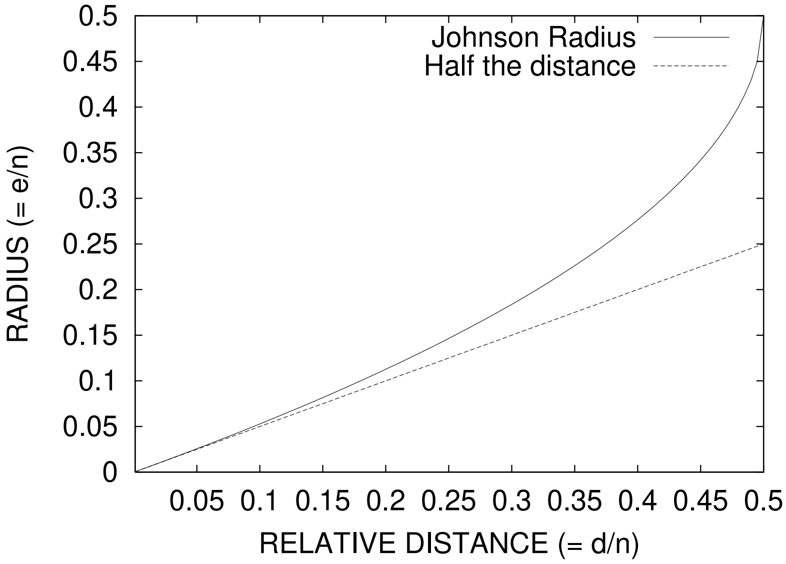

Figure 3-1: Plot of Johnson radius as a function of relative distance for binary codes. This shows that list decoding always permits decoding beyond half the distance.

(ii) if $e \le n - \sqrt{n(n-d+d/L)}$*, then* $A'_q(n,d,e) \le L$.

Proof: The proof follows from Theorem 3.2 and the fact that

$$(1-\sqrt{1-x}) \le (1-1/q)(1-\sqrt{1-\frac{qx}{q-1}})$$

for every integer q and every x, $0 \le x \le (1-1/q)$. The above inequality can be proved using a straightforward calculation. Using the above inequality with $x = d/n$ and $x = \frac{L-1}{L}\frac{d}{n}$ implies that the conditions on e stated in the corollary imply the Conditions (3.3) and (3.5) respectively. □

3.3.1 Proof of Theorem 3.1

Proof Idea: The proof follows a "geometric" approach. We identify elements of $[q]^n$ with vectors in $\mathbb{R}^{nq}$ by replacing the symbol i $(1 \le i \le q)$ by the unit vector of length q with a 1 in position i. This allows us to embed the codewords and the "received" word $\mathbf{r}$ into $\mathbb{R}^{nq}$. Next, by appropriately shifting the set of vectors corresponding to the codewords that are close to $\mathbf{r}$, we get a set of vectors such that the inner product of any two distinct vectors from this set is non-positive. By a standard geometric upper bound on the cardinality of such a set of vectors, we get the required upper bound on the number of codewords that are "close" to $\mathbf{r}$.

Our idea extends proofs for the binary case, given by [EZ95, Lev98, AVZ00]. These works used an appropriate embedding of the binary codewords in $\mathbb{R}^n$ and an appropriate shifting of vectors to establish "Johnson-style" bounds by appealing to bounds on spherical codes, i.e., bounds on the cardinality of a set of unit vectors in real space with a specified minimum angle between any pair of vectors. It may be noted that the generalization to

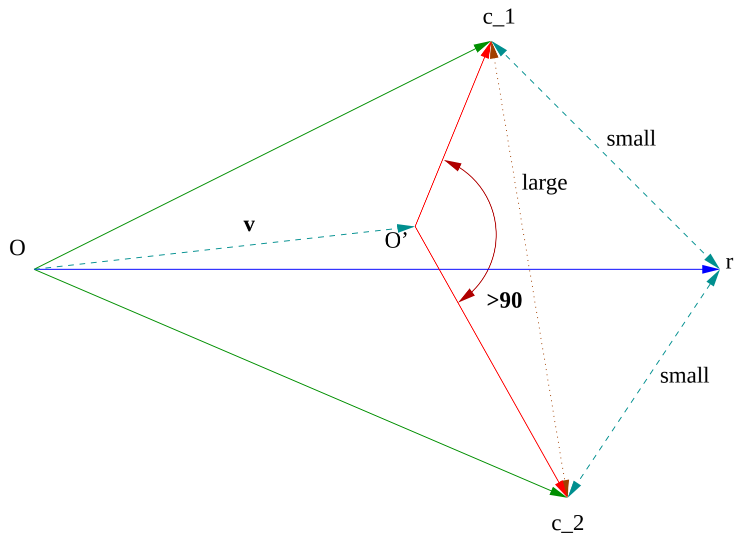

Figure 3-2: Geometric picture behind proof of Theorem 3.1

arbitrary alphabets is not automatic. (Of the several potential approaches, our proof hits upon the right path.)

Proof of Theorem 3.1: Assume without loss of generality that $\mathbf{r} = \langle q, q, \ldots, q\rangle$, i.e is the symbol q repeated n times. Let $C_1, C_2, \ldots, C_m$ be all the codewords of $\mathcal{C}$ that lie within $B_q(\mathbf{r}, e)$ where $e = (1 - 1/q)(1 - \gamma)n$. Our goal is to get an upper bound on m provided γ is large enough.

We associate a vector in $\mathbb{R}^{nq}$ with $\mathbf{r}$ and with each codeword C_i. Each vector is to be viewed as having n blocks each having q components (the n blocks correspond to the n codeword positions). For $1 \le l \le q$, denote by $\hat{e}_l$ the q-dimensional unit vector with 1 in the lth position and 0 elsewhere. For $1 \le i \le m$, the vector $\mathbf{c_i}$ associated with the codeword C_i has in its jth block the components of the vector $\hat{e}_{C_i[j]}$ ($C_i[j]$ is the jth symbol of C_i, treated as an integer between 1 and q). The vector associated with the received word $\mathbf{r}$, which we also denote $\mathbf{r}$ by abuse of notation, is defined similarly. Let $\mathbf{1} \in \mathbb{R}^{nq}$ be the all 1's vector. Now define $\mathbf{v} = \alpha\mathbf{r} + \frac{(1-\alpha)}{q}\mathbf{1}$ for a parameter $0 \le \alpha \le 1$ to be specified later in the proof. Note that the $\mathbf{c_i}$'s and $\mathbf{v}$ all lie in the space defined by the intersection of the n "hyperplanes" $\{\ \mathcal{H}'_j : \ \ \sum_{\ell=1}^{q} x_{j,\ell} = 1\ \}$ for $1 \le j \le n$. Hence the vectors $(\mathbf{c_i} - \mathbf{v})$, for $1 \le i \le m$, all lie in $\mathcal{H} = \bigcap_{j=1}^{n} \mathcal{H}_j$ where $\mathcal{H}_j = \{\mathbf{x} \in \mathbb{R}^{nq} : \sum_{\ell=1}^{q} x_{j,\ell} = 0\}$. It is easy to see that $\mathcal{H}$ is an $n(q-1)$-dimensional subspace of $\mathbb{R}^{nq}$. We thus conclude that the vectors $(\mathbf{c_i} - \mathbf{v})$, $1 \le i \le m$, all lie in an $n(q-1)$-dimensional space.

The idea behind the rest of the proof is the following. We will pick α so that the vectors $(\mathbf{c_i} - \mathbf{v})$, for $1 \le i \le m$, have all pairwise dot products less than 0. Geometrically speaking, we shift the origin O to O' where $OO' = \mathbf{v}$, and require that relative to the new origin the vectors corresponding to the codewords have pairwise angles which are greater than 90 degrees (see Figure 3-2). By a simple geometric fact (stated in Lemma 3.4 below), it will then follow that the number of codewords m is at most the dimension $n(q-1)$ of the space in which all these vectors lie.

For $1 \le i \le m$, let $e_i = \Delta(\mathbf{r}, C_i)$. Note that $e_i \le e$ for every i. Now

$$\begin{aligned}
\langle \mathbf{c_i}, \mathbf{v} \rangle &= \alpha\langle \mathbf{c_i}, \mathbf{r}\rangle + \frac{(1-\alpha)}{q}\langle \mathbf{c_i}, \mathbf{1}\rangle = \alpha(n - e_i) + (1-\alpha)\frac{n}{q} && (3.6)\\
\langle \mathbf{v}, \mathbf{v} \rangle &= \alpha^2 n + 2(1-\alpha)\alpha\frac{n}{q} + (1-\alpha)^2\frac{n}{q} = \frac{n}{q} + \alpha^2\Big(1 - \frac{1}{q}\Big)n && (3.7)\\
\langle \mathbf{c_i}, \mathbf{c_j} \rangle &= n - \Delta(C_i, C_j) \le n - d\ . && (3.8)
\end{aligned}$$

Using (3.6), (3.7) and (3.8), and the fact that each $e_i \le e$, we get, for $i \ne j$,

$$\langle \mathbf{c_i} - \mathbf{v}, \mathbf{c_j} - \mathbf{v} \rangle \le 2\alpha e - d + \Big(1 - \frac{1}{q}\Big)(1-\alpha)^2 n\ . \tag{3.9}$$

Using $e = (1 - 1/q)(1-\gamma)n$ and $d = (1-1/q)(1-\delta)n$ the above simplifies to

$$\langle \mathbf{c_i} - \mathbf{v}, \mathbf{c_j} - \mathbf{v} \rangle \le \Big(1 - \frac{1}{q}\Big)n\Big(\delta + \alpha^2 - 2\alpha\gamma\Big) \tag{3.10}$$

Thus as long as $\gamma > \frac{1}{2}\Big(\frac{\delta}{\alpha} + \alpha\Big)$ we will have all pairwise dot products to be negative just as we wanted. We pick α to minimize $(\frac{\delta}{\alpha} + \alpha)$, or in other words we set $\alpha = \sqrt{\delta}$. Now as long as $\gamma > \sqrt{\delta}$, we will have $\langle \mathbf{c_i} - \mathbf{v}, \mathbf{c_j} - \mathbf{v} \rangle < 0$ for all $1 \le i < j \le m$. To complete the proof, we note that (for the choice $\alpha = \sqrt{\delta}$), for every $1 \le i \le m$, $\langle \mathbf{c_i} - \mathbf{v}, \mathbf{v} \rangle \ge (1 - 1/q)n\sqrt{\delta}(\gamma - \sqrt{\delta}) > 0$ (this is easily checked using (3.6) and (3.7)). Thus provided $\gamma > \sqrt{\delta}$, we have $\langle \mathbf{c_i} - \mathbf{v}, \mathbf{v} \rangle > 0$ for $1 \le i \le m$. Now applying Part (iii) of Lemma 3.4, with the setting $\mathbf{v_i} = \mathbf{c_i} - \mathbf{v}$ and $\mathbf{u} = \mathbf{v}_{|\mathcal{H}}$, the projection of $\mathbf{v}$ onto the subspace $\mathcal{H}$, implies that $m \le n(q-1)$ (recall that the vectors $(\mathbf{c_i} - \mathbf{v})$, $1 \le i \le m$, all lie in $\mathcal{H}$ and $\dim(\mathcal{H}) = n(q-1)$).

We now prove that if $\gamma > \sqrt{\delta}$, then $m \le \frac{1-\delta}{\gamma^2 - \delta}$. For this we set $\alpha = \gamma$. Now from Equation (3.10) we have

$$\langle \mathbf{c_i} - \mathbf{v}, \mathbf{c_j} - \mathbf{v} \rangle \le (1 - 1/q)n(\delta - \gamma^2)\ . \tag{3.11}$$

Thus if $\gamma > \sqrt{\delta}$, we have $\langle \mathbf{c_i} - \mathbf{v}, \mathbf{c_j} - \mathbf{v} \rangle < 0$. Now for the choice $\alpha = \gamma$, we have for each i, $1 \le i \le m$,

$$\|\mathbf{c_i} - \mathbf{v}\|^2 = \langle \mathbf{c_i} - \mathbf{v}, \mathbf{c_i} - \mathbf{v} \rangle \le 2\alpha e + (1 - 1/q)(1-\alpha)^2 n = n(1 - 1/q)(1 - \gamma^2)\ .$$

Denote by $\mathbf{w_i}$ the unit vector $-\frac{\mathbf{c_i} - \mathbf{v}}{\|\mathbf{c_i} - \mathbf{v}\|}$. We then have

$$\langle \mathbf{w_i}, \mathbf{w_j} \rangle \le -\frac{\gamma^2 - \delta}{1 - \gamma^2} \tag{3.12}$$

for $1 \le i < j \le m$ (this follows from (3.11) and (3.12)). By a well-known geometric fact (see Lemma 3.5 for the simple proof), it follows that the number of such vectors, m, is at most $(1 + \frac{1-\gamma^2}{\gamma^2 - \delta}) = \frac{1-\delta}{\gamma^2-\delta}$, as desired.

To handle the case when $\gamma = \sqrt{\delta}$, we can choose $\alpha = \sqrt{\delta}$, and we then have $\langle \mathbf{c_i} - \mathbf{v}, \mathbf{c_j} - \mathbf{v} \rangle \le 0$ for all $1 \le i < j \le m$, and also $\langle \mathbf{c_i} - \mathbf{v}, \mathbf{v} \rangle \ge 0$ for each $i = 1, 2, \ldots, m$. Now applying Part (ii) of Lemma 3.4, we get $m \le 2n(q-1) - 1$. □

3.3.2 Geometric Lemmas

We now state and prove the geometric facts that were used in the above proof.

Lemma 3.4 *Let $\mathbf{v_1}, \ldots, \mathbf{v_m}$ be non-zero vectors in $\mathbb{R}^N$ such that $\langle \mathbf{v_i}, \mathbf{v_j} \rangle \leq 0$ for all $1 \leq i < j \leq m$. Then the following hold:*

(i) $m \leq 2N$.

(ii) Suppose that there exists a non-zero $\mathbf{u} \in \mathbb{R}^N$ such that $\langle \mathbf{u}, \mathbf{v_i} \rangle \geq 0$ for $i = 1, 2, \ldots, m$. Then $m \leq 2N - 1$.

(iii) Suppose there exists an $\mathbf{u} \in \mathbb{R}^N$ such that $\langle \mathbf{u}, \mathbf{v_i} \rangle > 0$ for $i = 1, 2, \ldots, m$. Then $m \leq N$.

A proof of Part (i) of the above lemma can be found, for instance, in [Bol86, Chapter 10, page 71]. The proofs of the other two parts are similar. For completeness, we present a self-contained proof below.

Proof of Lemma 3.4: We first prove (iii). Suppose for contradiction that $m \geq N + 1$. Then since the vectors $\mathbf{v_1}, \ldots, \mathbf{v_m}$ all lie in $\mathbb{R}^N$, they must be linearly dependent. Let $S \subseteq [m]$ be a non-empty set of *minimum* size for which a relation of the form $\sum_{i \in S} a_i \mathbf{v_i} = \mathbf{0}$ holds with each $a_i \neq 0$. We claim that the a_i's must all be positive or all be negative. Indeed, if not, by collecting terms with positive a_i's on one side and those with negative a_i's on the other, we will have an equation of the form $\sum_{i \in T^+} a_i \mathbf{v_i} = \sum_{j \in T^-} b_j \mathbf{v_j} = \mathbf{w}$ (for some vector $\mathbf{w}$) where T^+ and T^- are *disjoint* non-empty sets with $T^+ \cup T^- = S$, and all $a_i, b_j > 0$. By the minimality of S, $\mathbf{w} \neq \mathbf{0}$ and hence $\langle \mathbf{w}, \mathbf{w} \rangle > 0$. On the other hand $\langle \mathbf{w}, \mathbf{w} \rangle = \langle \sum_{i \in T^+} a_i \mathbf{v_i}, \sum_{j \in T^-} b_j \mathbf{v_j} \rangle = \sum_{i,j} a_i b_j \langle \mathbf{v_i}, \mathbf{v_j} \rangle \leq 0$ since $a_i b_j > 0$ and $\langle \mathbf{v_i}, \mathbf{v_j} \rangle \leq 0$ for each $i \in T^+$ and $j \in T^-$. This contradiction shows that we may assume that $a_i > 0$ for all $i \in S$.

Now $\sum_{i \in S} a_i \mathbf{v_i} = \mathbf{0}$, so that $\sum_{i=1}^{s} a_i \langle \mathbf{u}, \mathbf{v_i} \rangle = 0$. But this is impossible since for each i we have $a_i > 0$ and $\langle \mathbf{u}, \mathbf{v_i} \rangle > 0$. We have thus arrived at a contradiction, and therefore such a linear dependence $\sum_{i \in S} a_i \mathbf{v_i} = \mathbf{0}$ does not exist. Thus the vectors $\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_m}$ are linearly independent and we must have $m \leq N$.

To prove (ii), we use induction on N. The statement clearly holds for $N = 1$. For $N > 1$, we proceed exactly as above. If $m \leq N$, we have nothing to prove, so assume $m > N$ so that $\mathbf{v_1}, \ldots, \mathbf{v_m}$ are linearly independent, and as above, let $S \subseteq [m]$ be a non-empty set of minimum size for which a relation of the form $\sum_{i \in S} a_i \mathbf{v_i} = \mathbf{0}$ holds with each $a_i \neq 0$. Arguing as above, we may assume that $a_i > 0$ for every $i \in S$. Assume for definiteness that $S = \{1, 2, \ldots, s\}$. We thus have the linear dependence $\sum_{i=1}^{s} a_i \mathbf{v_i} = \mathbf{0}$ with each $a_i > 0$, and since this is a minimum sized linear dependence, $\mathbf{v_1}, \ldots, \mathbf{v_s}$ must span a subspace W of $\mathbb{R}^N$ of dimension $(s - 1)$.

Since $\sum_{i=1}^{s} a_i \mathbf{v_i} = \mathbf{0}$, we have $\sum_{i=1}^{s} a_i \langle \mathbf{v_i}, \mathbf{v}_\ell \rangle = 0$ for each $\ell = s+1, \ldots, m$. Since $a_i > 0$ for $1 \leq i \leq s$ and $\langle \mathbf{v_i}, \mathbf{v}_\ell \rangle \leq 0$, it must be therefore be the case that $\mathbf{v_i}$ is orthogonal to $\mathbf{v}_\ell$ for all i, ℓ with $1 \leq i \leq s$ and $s < \ell \leq m$. A similar argument shows $\mathbf{u}$ is orthogonal to $\mathbf{v_i}$ for each $i = 1, 2, \ldots, s$. Thus the vectors $\mathbf{v_{s+1}}, \ldots, \mathbf{v_m}$ and $\mathbf{u}$ all lie in $W^\perp$ which has dimension equal to $(N - s + 1)$. Since $s > 1$, the induction hypothesis applied to these vectors implies that $m - s \leq 2(N - s + 1) - 1$, or in other words $m \leq 2N - s + 1 \leq 2N - 1$, as desired.

Finally (i) follows immediately from (ii). Indeed, apply (ii) with vectors $\mathbf{v_1}, \ldots, \mathbf{v_{m-1}}$ and $-\mathbf{v_m}$ playing the role of $\mathbf{u}$. This implies $m - 1 \leq 2N - 1$, or in other words $m \leq 2N$. $\square$

Lemma 3.5 *Let $\varepsilon > 0$ be a positive real and let $\mathbf{w_1}, \mathbf{w_2}, \ldots, \mathbf{w_m}$ be m unit vectors such that $\langle \mathbf{w_i}, \mathbf{w_j} \rangle \le -\varepsilon$ for all $1 \le i < j \le m$. Then $m \le 1 + \frac{1}{\varepsilon}$.*

Proof: We have

$$0 \le \langle \sum_{i=1}^{m} \mathbf{w_i}, \sum_{i=1}^{m} \mathbf{w_i} \rangle = \sum_{i=1}^{m} \langle \mathbf{w_i}, \mathbf{w_i} \rangle + 2 \sum_{1 \le i < j \le m} \langle \mathbf{w_i}, \mathbf{w_j} \rangle \le m - m(m-1)\varepsilon \ ,$$

which gives $m \le 1 + 1/\varepsilon$. □

3.4 Generalization in Presence of Weights

For applications to "soft" list decoding algorithms which will be discussed in Part II of the thesis, it is of interest to prove a version of the Johnson bound in the presence of weights on codeword symbols. Such a bound is also of independent interest, since it covers the case of decoding under errors-and-erasures and the case when for each position one receives a small list of candidate symbols one of which is the correct one, all under a uniformly applicable bound.

We next state the weighted version of the Johnson bound that follows from our proof technique. The bound in Part (i) of the theorem generalizes the result of Theorem 3.2. The result from Part (ii) applies under a more general condition than Condition (3.3) (or even Condition (3.13)), but the upper bound itself is slightly weaker (since it is $(nq-1)$ instead of $n(q-1)$). The result of Part (iii) generalizes the result of Theorem 3.2, Condition 3.5.

Theorem 3.6 *Let $\mathcal{C} \subseteq [q]^n$ be a code of blocklength n and minimum distance d. Let $\{w_{i,j} : 1 \le i \le n; 1 \le j \le q\}$ be an arbitrary set of non-negative real weights. Define $W_i = \sum_{j=1}^{q} w_{i,j}$ and $W_i^{(2)} = \sum_{j=1}^{q} w_{i,j}^2$, $W_{\text{tot}} = \sum_{i,j} w_{i,j}$, and $W_{\text{tot}}^{(2)} = \sum_{i,j} w_{i,j}^2$. Then:*

(i) The number of codewords $C \in \mathcal{C}$ that satisfy

$$\sum_{i=1}^{n} \frac{w_{i,C_i}}{W_i} > \frac{n}{q} + \sqrt{\left(n\left(1 - \frac{1}{q}\right) - d\right)\left(\sum_{i=1}^{n} \frac{W_i^{(2)}}{W_i^2} - \frac{n}{q}\right)} \ . \tag{3.13}$$

is at most $n(q-1)$.

(ii) The number of codewords $C \in \mathcal{C}$ that satisfy

$$\sum_{i=1}^{n} w_{i,C_i} > \frac{W_{\text{tot}}}{q} + \sqrt{\left(n\left(1 - \frac{1}{q}\right) - d\right)\left(W_{\text{tot}}^{(2)} - \frac{(W_{\text{tot}})^2}{nq}\right)} \tag{3.14}$$

is at most $(nq-1)$.

(iii) For any integer $L \ge 2$, the number of codewords $C \in \mathcal{C}$ that satisfy

$$\sum_{i=1}^{n} w_{i,C_i} \ge \frac{W_{\text{tot}}}{q} + \sqrt{\left(n\left(1 - \frac{1}{q}\right) - d + \frac{d}{L}\right)\left(W_{\text{tot}}^{(2)} - \frac{(W_{\text{tot}})^2}{nq}\right)} \tag{3.15}$$

is at most L.

Proof: We do not give a full proof here, rather we indicate the only changes that must be made to the proof of Theorem 3.1 in order to prove our claim. For Part (i), the only modification required in the proof of Theorem 3.1 is to pick $\mathbf{r}$ so that its (i,j)'th component, for $1 \le i \le n$ and $1 \le j \le q$, equals $\frac{w_{i,j}}{W_i}$. The vector $\mathbf{v}$ is defined as before to be $\alpha\mathbf{r} + \frac{(1-\alpha)}{q}\mathbf{1}$ for

$$\alpha = \sqrt{\frac{n(1-1/q)-d}{\sum_i \frac{W_i^{(2)}}{W_i^2} - n/q}} \ .$$

Once once again all the vectors $(\mathbf{c_i} - \mathbf{v})$ lie in an $n(q-1)$-dimensional subspace of $\mathbb{R}^{nq}$. It can be proved as in the proof of Theorem 3.1 that these vectors have pairwise non-positive dot products, which gives the desired $n(q-1)$ upper bound on the number of codewords.

For Parts (ii) and (iii), we pick $\mathbf{r}$ so that its (i,j)'th component for $1 \le i \le n$ and $1 \le j \le q$, equals $\frac{nw_{i,j}}{W_{\text{tot}}}$, and the rest of the proof follows that of Theorem 3.1. Note that W_{tot}/q is the expected value of $\sum_i w_{i,r_i}$ for a random vector $\mathbf{r} \in [q]^n$, and $(W_{\text{tot}}^{(2)} - \frac{(W_{\text{tot}})^2}{nq})$ is proportional to the variance of the $w_{i,j}$'s. Thus, the above theorem states that the number of codewords which have weighted agreement bounded away from the expectation by a certain number of standard deviations is small. The upper bound of $(nq-1)$ (instead of $n(q-1)$) in Part (ii) of above theorem arises since we are only able to ensure that the vectors $(\mathbf{c_i} - \mathbf{v})$ all lie in an $(nq-1)$-dimensional subspace (namely that defined by $\sum_{i,j} x_{i,j} = 0$), and not an $n(q-1)$-dimensional subspace as in Part (i). $\square$

We now state a corollary similar to Corollary 3.3 that ignores the alphabet size in the decoding condition. The proof again follows because it can be verified (after a straightforward but tedious calculation) that the stated conditions in fact imply the Conditions (3.14) and (3.15) above.

Corollary 3.7 *Let $\mathcal{C} \subseteq [q]^n$ be a code of blocklength n and minimum distance d. Let $\{w_{i,j} : 1 \le i \le n; 1 \le j \le q\}$ be an arbitrary set of non-negative real weights.*

(i) The number of codewords $C \in \mathcal{C}$ that satisfy

$$\sum_{i=1}^{n} w_{i,C_i} > \Big((n-d)\sum_{i,j} w_{i,j}^2\Big)^{1/2} \tag{3.16}$$

is at most $(nq-1)$.

(ii) For any integer $L \ge 2$, the number of codewords $C \in \mathcal{C}$ that satisfy

$$\sum_{i=1}^{n} w_{i,C_i} \ge \Big(\Big(n-d+\frac{d}{L}\Big)\sum_{i,j} w_{i,j}^2\Big)^{1/2} \tag{3.17}$$

is at most L.

A bound similar to Corollary 3.7 above can also be worked out for the case when the different codeword positions have different contributions towards the minimum distance. Such a bound is of interest for certain codes like the Chinese Remainder Code and will be stated and formally proved in the form of Theorem 7.9 in Section 7.6.1 of the thesis. We refer the reader interested in seeing a full proof of Corollary 3.7 above to the proof of Theorem 7.9.

3.5 Notes

The quantity $A(n,d,w)$ for constant-weight binary codes has a rich history and has been studied for almost four decades, and its study remains one of the most basic questions in coding theory. The first upper bounds on the quantity $A(n,d,w)$ for constant-weight codes appear in the work of Johnson [Joh62, Joh63]. Since then several proofs have appeared in the literature, including generalizations of the bound to the case of q-ary alphabets for $q > 2$ (cf. [Bro98] for a discussion and detailed bibliography).

The quantity $A'(n,d,e)$, which is of more direct interest to list decoding, seems to have received much less explicit attention. It must be said that several proofs that provide upper bounds on $A(n,d,e)$ work with little or no modification to yield upper bounds on $A'(n,d,e)$ as well. This was made explicit for example in [Eli91, BGS98]. Upper bounds on $A'_q(n,d,e)$ identical to the second upper bound in (3.4) of this chapter are stated in [Bro98]. Proofs of such bounds that follow a linear algebra based argument appear, for instance, in [GRS95, GS00].

The contribution of the results in this chapter is that we extend the more recent upper bounds for the binary case from [AVZ00] (which are based on geometric arguments) to bounds on $A'_q(n,d,e)$, and furthermore we obtain some elegant weighted generalizations of the Johnson bound. In particular, the upper bound $A'_q(n,d,e) \le n(q-1)$ for $e < e_J(n,d,q)$ that we proved in Theorem 3.2 appears to be new. For the case $q = 2$, this result was known. Specifically, Elias [Eli57] proved that if d is *odd*, then $A'(n,d,e) \le n$ as long as e is at most the binary Johnson radius $e_J(n,d,2)$. For even d, however, $A'(n,d,e) = O(n^2)$ was the best known bound that was made explicit till the recent work of Agrell, Vardy and Zeger [AVZ00], who showed that $A'(n,d,e) \le n$ whenever $e < e_J(n,d,2)$. (Actually, Agrell *et al.* claim their result only for $A(n,d,e)$, but their proof works for the case of $A'(n,d,e)$ as well.)

Combinatorial results of a flavor similar to this chapter appear in two other parts of the thesis: (a) in Section 7.6.1 where a bound similar to Corollary 3.7 is proved for the case when the minimum distance is measured with a non-uniform weight on the codeword positions, and (b) in Section 8.5.1 where we prove a result along the lines of Theorem 3.2, but instead of bounding the number of codewords in a Hamming ball of certain radius, we establish a more general result concerning the coset weight distribution of a code, purely as a function of its minimum distance.

The material in this chapter appears in [GS01a].

Chapter 4

Limits to List Decodability

Our difficulty is not in the proofs, but in learning what to prove.
Emil Artin

4.1 Introduction

The previous chapter showed that every code of certain minimum distance has an associated "Johnson radius" which gives a lower bound on the list decoding radius (in other words, every Hamming ball of radius up to the Johnson radius has "few" codewords). This result plays an important role in the development of the subject of list decoding. Indeed, by showing that any code with large distance has large list decoding radius, it raises algorithmic questions concerning list decoding important families of codes beyond half the minimum distance.

But at a purely combinatorial level, this also raises the following natural question on the "optimality" of the results from the previous chapter: Is the Johnson radius the best possible bound on list decoding radius in terms of the minimum distance, or could there be an even better lower bound on the list decoding radius of a code?

This chapter addresses the above question. The results of this chapter demonstrate that the Johnson radius is indeed essentially tight as a general relation between list decodability and minimum distance. Note that this does not say that for *every* code the Johnson bound on list decoding radius is the correct one – rather, it says there exist *some* codes for which this is the case.[1] In other words, purely as a function of the distance of the code, the Johnson radius gives (asymptotically) the best possible bound on list decoding radius. The basic strategy behind showing this is to construct a code family of certain relative minimum distance which has a large (super-constant or super-polynomial, depending upon the actual result) number of codewords within a Hamming ball of radius close to the Johnson radius.

We should remark that for general, non-linear codes, it was already known (by a simple proof) that one can have exponentially many codewords just beyond the Johnson radius (see for instance [GRS95] – their result is formally stated in Section 4.3.1). Indeed a random constant-weight code with suitable parameters has this property with high probability. The thrust of this chapter is, therefore, on linear codes. Note that most of the interesting code families are linear, and it is therefore important to understand the list decodability vs.

[1]Indeed the results of the next chapter demonstrate that for *most* codes, the Johnson radius is *not* the best possible bound on the list decoding radius.

distance trade-off restricted to linear codes. Also, for the sake of simplicity, we focus on binary codes in this chapter.

We stress that imposing the requirement of linearity makes the problem significantly harder, and the results presented in this chapter represent the first asymptotic results that give non-trivial linear code constructions with the property that there exist "several" codewords in a ball of "small" (as a function of the distance) radius.

We should also mention here that the results of this chapter by their very nature are rather technical. While an appreciation for the statement of the results in this chapter is useful to put the various pieces of the thesis in context, the results themselves, and more so the proofs, are fairly independent of the rest of the thesis. The reader might therefore want to skip some of the proofs in a first reading.

4.2 Informal description of results

The main results of this chapter are informally stated below. (A formal description of the results in the form of theorem statements will be given in Section 4.3 after the relevant notation and formalism is developed.) All results are for binary linear codes. Recall from the previous chapter (specifically, Equation (3.3) from Theorem 3.2) that the Johnson radius (normalized by blocklength) of a binary code of relative distance δ ($0 \le \delta \le 1/2$) equals

$$J(\delta) = \frac{1 - \sqrt{1 - 2\delta}}{2} . \tag{4.1}$$

The results proven in this chapter include:

1. The list decoding radius for constant-sized lists approaches the Johnson radius of a code; in other words for any constant L and relative distance δ, there exist linear codes with more than L codewords in a Hamming ball of relative radius close to $J(\delta)$.

2. The list decoding radius for list size being any fixed polynomial of the blocklength is strictly less than the minimum distance of the code.

3. The list decoding radius for list size growing polynomially in the blocklength approaches the Johnson radius. In other words, for every δ, there exist linear codes of relative distance δ that have a super-polynomial number of codewords in some Hamming ball of relative radius close to $J(\delta)$. This result is stronger than that stated above as Result 2. However, we are able to prove this result only under a number-theoretic conjecture. The conjecture is a widely believed one (it is a very special case of the "Artin conjecture"), and is in particular known to hold under the Generalized Riemann Hypothesis (GRH).

4.3 Formal description of results

4.3.1 The result for non-linear codes

Before describing our results (which are for linear codes), we recall the following result from [GRS95] that shows that the Johnson bound on list decoding radius is tight for general, non-linear codes. We state the result only for binary codes – an analogous statement also holds for q-ary codes. The proof is quite straightforward and follows by picking a random constant weight code with a certain number of codewords and then arguing that it has "good" distance with high probability. This gives several codewords in a small Hamming ball centered at the all-zeroes word. The reader is referred to [GRS95] for further details on the proof.

Proposition 4.1 ([GRS95]) *For every δ, $0 < \delta < 1/2$, for all small enough $\varepsilon > 0$, and for all sufficiently large n, there exists a (non-linear) binary code of blocklength n and relative distance at least δ, that has at least $2^{\Omega(\varepsilon^2 n)}$ codewords in a Hamming ball of radius $\frac{n}{2}(1+\varepsilon-\sqrt{1-2\delta-\varepsilon})$.*

The Johnson bound (Theorem 3.2) states that the number of codewords in a Hamming ball of radius $\frac{n}{2}(1-\sqrt{1-2\delta})$ is at most $2n$. Therefore the above establishes that in a ball of radius slightly greater than the Johnson radius, there could in fact be exponentially many codewords.

4.3.2 Definitions

We first develop the necessary notation and definitions in order to describe our results formally.

Definition 4.1 (Lower bound on list decoding radius) *For a distance parameter δ, $0 \le \delta \le 1/2$, and list size $\ell : \mathbb{Z}^+ \to \mathbb{Z}^+$, the* lower bound on list of ℓ decoding radius *for* binary linear *codes of relative distance δ, denoted $L_\ell(\delta)$, is defined to be*

$$L_\ell(\delta) = \inf_{\mathcal{C} \,\mid\, \delta(\mathcal{C}) \ge \delta} \mathsf{LDR}_\ell(\mathcal{C}) \ ,$$

where the infimum is taken over all binary linear *code families of relative distance at least δ.*

One could also define the above function $L_\ell(\delta)$ for relative distance δ which is a function of the blocklength. In this case, the infimum above would be taken over codes families $\mathcal{C} = \{C_i\}_{i \ge 1}$ that satisfy $\mathrm{dist}(C_i) \ge \delta(n_i) \cdot n_i$ where n_i is the blocklength of C_i for $i \ge 1$.

Note that the terminology "lower bound on list decoding radius" comes from the fact that $L_\ell(\delta)$ is the *maximum* fractional radius r for which *every* code of blocklength n and relative distance δ is guaranteed to have at most $\ell(n)$ codewords in any Hamming ball of radius rn. In other words, every code of relative distance δ has list decoding radius *at least* $L_\ell(\delta)$.

We will be interested in studying the above function for the case of list size being a constant or growing as a polynomial in the blocklength. To do so, we need further definitions.

Definition 4.2 *For real distance $0 \le \delta < 1/2$ and an integer constant $c \ge 1$:*

(i) The quantity $L_c^{\mathrm{const}}(\delta)$ is defined to be $L_\ell(\delta)$ where $\ell(n) = c$ is a constant function.

(ii) The quantity $L_c^{\mathrm{poly}}(\delta)$ is defined to be $\limsup_{c_1 \to \infty} L_{\ell_{c_1}}(\delta)$ where $\ell_{c_1}(n) = c_1 n^c$.

(iii) The quantity $L^{\mathrm{const}}(\delta)$ is defined as $\limsup_{c \to \infty}\{L_c^{\mathrm{const}}(\delta)\}$.

(iv) The quantity $L^{\mathrm{poly}}(\delta)$ is defined as $\limsup_{c \to \infty}\{L_c^{\mathrm{poly}}(\delta)\}$.

4.3.3 Statement of results

It is straightforward to verify that, in the above notation, the result of Theorem 3.2 from the previous chapter implies the following:

Proposition 4.2 *For every δ, $0 \le \delta \le 1/2$ and integer $c \ge 1$,*

$$L_c^{\text{const}}(\delta) \ge \frac{1}{2}\left(1 - \sqrt{1 - 2\delta + \frac{2\delta}{c}}\right) .$$

As a corollary, we have $L^{\text{const}}(\delta) \ge J(\delta) = (1 - \sqrt{1-2\delta})/2$.

Thus the above gives a lower bound on $L^{\text{const}}(\delta)$. The first result is that this lower bound is tight:

Theorem 4.3 *For every δ, $0 \le \delta \le 1/2$, $L^{\text{const}}(\delta) = J(\delta) = (1 - \sqrt{1-2\delta})/2$.*

Given the precise understanding of the L^{const} function from the above theorem, we are next interested in understanding the functions L_c^{poly} and L^{poly}. Here we make the following conjecture that the Johnson bound is in fact tight even for list decoding with polynomial-sized lists. (The result of Proposition 4.1 implies that the conjecture holds if one allows non-linear codes, but for linear codes, which are the focus of this chapter, the bound on $L^{\text{poly}}(\delta)$ stated below remains a conjecture.)

Conjecture 4.1 *For every $0 < \delta < 1/2$, $L^{\text{poly}}(\delta) = \frac{1}{2} \cdot (1 - \sqrt{1-2\delta})$.*

If true, the above can be viewed as the main point of this chapter. However, we are as yet unable to settle the conjecture. But we are able to prove it assuming a widely accepted number-theoretic conjecture which is in turn implied by the Generalized Riemann Hypothesis. Hence, there is strong evidence for the truth of the conjecture. We are also able to prove some non-trivial unconditional results that lend further support to the conjecture. We list the relevant results below. The proofs will be given in later sections.

Theorem 4.4 *For every ε, $0 < \varepsilon < 1/2$, for some $\delta : \mathbb{Z} \to \mathbb{Z}$ satisfying $\delta(n) = \frac{1}{2}(1 - \Theta((\log n)^{\varepsilon - 1}))$, we have $L^{\text{poly}}(\delta) \le \frac{1}{2}[1 - (1-2\delta)^{1/2+\varepsilon}]$.*

The above can be viewed as a "resolution" of Conjecture 4.1 for the case $\delta = 1/2 - o(1)$. For constant δ, we are able to show, unconditionally, that the list decoding radius for polynomial-sized lists is strictly bounded away from the distance of the code.

Theorem 4.5 *For every integer $c \ge 1$ and every δ, $0 < \delta < 1/2$, we have $L_c^{\text{poly}}(\delta) < \delta$.*

Theorem 4.6 *Assume that there exist infinitely many primes p such that 2 is a generator of the cyclic multiplicative group $\mathbb{F}_p^*$ of $\mathbb{F}_p$. Then $L^{\text{poly}}(\delta) = J(\delta)$.*

The Artin conjecture and the hypothesis of Theorem 4.6: The hypothesis of the above theorem is a special case of the *Artin conjecture* (see [Art65]) which gives an estimate of the density of primes p for which 2, or for that matter any fixed prime g, is a generator for the cyclic group $\mathbb{F}_p^*$ (for most g this density is conjectured to be quite close to 0.4). It is known that the Artin conjecture, with some correction factors in the density estimate, holds under the

Generalized Riemann Hypothesis (GRH) [Hoo67] (see also [IR90] for an account on the some of the remarkable progress that has been made towards resolving the Artin conjecture). It follows that the hypothesis of Theorem 4.6 holds under the GRH.

Note that the bound of Theorem 4.5 is implied by that of Theorem 4.6 (since $J(\delta) < \delta$ for every δ, $0 < \delta < 1/2$). But the result of Theorem 4.5 is unconditional and does not rely on any unproven number-theoretic conjecture, and is thus not "dominated" by that of Theorem 4.6.

4.4 Proof for constant-sized lists

In this section, we will prove Theorem 4.3. The results of this section are based on the ideas of Justesen and Hϕholdt [JH01], though we fill in several details in their proofs and also need some new ideas to prove our claim. In particular, Justesen and Hϕholdt state their results only for large alphabets while we are interested in results for binary codes. In fact their results hold for Maximum Distance Separable (MDS) codes which are $[N, K, D]$ linear codes whose dimension and minimum distance satisfy the optimal trade-off $K + D = N + 1$ (i.e., they match the *Singleton Bound*). Such a code is characterized by an $N \times K$ generator matrix which has the property that every $K \times K$ submatrix has rank K (cf. [MS81, Chap. 11]).

4.4.1 The Basic Construction

The following lemma is at the core of the results of this section. It is proved using the basic construction scheme in [JH01]. However, we need an explicit upper bound on the size of the field over which the code is defined, while [JH01] were content in getting MDS codes over some large enough field. Consequently, we have to be more careful in our proof, specifically when proving a certain linear algebraic claim from [JH01], because we need an explicit upper bound on the field size. We isolate the necessary linear-algebraic fact and prove it as a separate technical lemma in Section 4.4.3.

Lemma 4.7 *For all large enough integers m, s with $2 < s < m + 1$, and for all sufficiently large f, there exists a linear MDS code $\mathbf{C}$ over $\mathrm{GF}(2^f)$ of blocklength $N = \binom{m+1}{s}$, relative distance $\delta = 1 - \frac{s(s-1)}{m(m+1)}$, and dimension $(1 - \delta)N + 1$, with the property that there exists a Hamming ball of radius τN where $\tau = 1 - s/(m + 1)$ containing at least $(m + 1)$ codewords of $\mathbf{C}$. In other words, $\mathbf{C}$ is* not *$(\tau N, m)$-list decodable. Moreover, it suffices if $f \geq \Omega(N)$ for the above claim, so that an MDS code with the stated properties exists over a field of size at most $2^{O(N)}$.*

Proof: The code construction and the configuration of $(m + 1)$ codewords that lie in a small Hamming ball will be based on a certain "combinatorial design". Since the design used is a trivial one, we simply present the actual construction without developing or using any design-theoretic notation or terminology.

Let $2 < s < m + 1$ and $N = \binom{m+1}{s}$. Let $B_1, B_2, \ldots, B_N$ be the set of N s-elements subsets of $\{0, 1, 2, \ldots, m\}$, ordered so that the $r = \binom{m}{s-1}$ sets containing 0 are the first r sets $B_1, B_2, \ldots, B_r$. For $j = 0, 1, \ldots, m$, define the N-dimensional vector $\mathbf{c_j}$ as follows: $\mathbf{c_0} = \mathbf{0}$ and for $1 \leq j \leq m$ and $1 \leq l \leq N$,

$$\mathbf{c_j}(l) = \begin{cases} 0 & \text{if } j \in B_l \text{ and } 0 \in B_l, \\ 1 & \text{if } j \in B_l \text{ and } 0 \notin B_l, \end{cases} \tag{4.2}$$

c_1	0	0	0				1	1	1	
c_2	0			0	0		1	1		1
c_3		0		0		0	1		1	1
c_4			0		0	0		1	1	1

Figure 4-1: The construction for $m = 4$, $s = 3$ and $N = 10$. We assume a lexicographic ordering of the 3-element subsets of $\{0, 1, 2, 3, 4\}$, so that $B_1 = \{0, 1, 2\}$, $B_2 = \{0, 1, 3\}$, $B_3 = \{0, 1, 4, \}$, $B_4 = \{0, 2, 3\}$, and so on.

Figure 4-1 illustrates the construction of the $\mathbf{c_j}$'s for the case $m = 4$ and $s = 3$; in this case $N = 10$ so that each vector has 10 coordinates out of which $\binom{m}{s-1} = 6$ are filled with 0's or 1's. The values of the $\mathbf{c_j}$'s in the remaining positions (not fixed by (4.2) above) will be suitably picked elements from $\mathrm{GF}(2^f) \setminus \{0, 1\}$, no element being used more than once. Here f is any sufficiently large integer. The exact choice of f that will suffice and the choice of elements at the remaining positions will be described shortly.

Define $\delta = (1 - \frac{s(s-1)}{m(m+1)})$. Now, for any pair of elements out of $\{0, 1, \ldots, m\}$, the number of sets B_i which contain both of them is precisely $\binom{m-1}{s-2}$. Using this fact, it is follows that the Hamming weight of each $\mathbf{c_j}$, $1 \le j \le m$, as well as the Hamming distance $\Delta(\mathbf{c_i}, \mathbf{c_j})$ between $\mathbf{c_i}$ and $\mathbf{c_j}$ for $1 \le i < j \le m$, all equal

$$D \stackrel{\mathrm{def}}{=} \binom{m+1}{s} - \binom{m-1}{s-2} = N\Big(1 - \frac{s(s-1)}{m(m+1)}\Big) = \delta N \ . \tag{4.3}$$

Our goal will be to realize an $[N, K, D]$ MDS code with $K = N - D + 1$ that contains all the $\mathbf{c_j}$'s, $0 \le j \le m + 1$, as codewords. Furthermore all these $(m + 1)$ codewords will lie in a Hamming ball, say Γ, of radius $E = \tau N$ where $\tau = 1 - \frac{s}{m+1}$. Together, these will imply the statement of the lemma. The latter of the above goals is easier to meet, while the former requires some work. Therefore, we first specify the center $\mathbf{x} = \langle x_1, x_2, \ldots, x_N \rangle$ of the Hamming ball Γ. This is defined as:

$$x_l = \begin{cases} 0 & \text{for } 1 \le l \le r \\ 1 & \text{for } r < l \le N. \end{cases}$$

(Recall that $r = \binom{m}{s-1} = \frac{sN}{m+1}$ is the number of B_i's that contain 0.) It is clear that Hamming distance between $\mathbf{x}$ and $\mathbf{c_0} = \mathbf{0}$ is

$$\Delta(\mathbf{x}, \mathbf{c_0}) = N - r = N\Big(1 - \frac{s}{m+1}\Big) = \tau N \ .$$

For $j \ge 1$, $\mathbf{c_j}$ has a 0-entry in $\binom{m-1}{s-2}$ of the first r entries (corresponding to s-sets that contain both 0 and j) and a 1-entry in $\binom{m-1}{s-1}$ of the last $(N - r)$ entries (corresponding to

s-sets that contain j but not 0). Hence we have

$$\Delta(\mathbf{x}, \mathbf{c_j}) = N - \left(\binom{m-1}{s-2} + \binom{m-1}{s-1}\right) = N - \binom{m}{s-1} = \left(1 - \frac{s}{m+1}\right)N = \tau N \ .$$

Thus, we have all the $m+1$ "codewords" $\mathbf{c_j}$, $0 \le j \le m$, in a Hamming ball of radius τN.

It remains to realize the strings $\mathbf{c_1}, \ldots, \mathbf{c_m}$ as codewords in an $[N, K = N - D + 1, D]$ MDS code over a sufficiently large field $\mathbb{F} = \mathrm{GF}(2^f)$. Note that $\mathbf{c_0} = \mathbf{0}$ is always a member of any linear code and hence we have to only worry about the $\mathbf{c_j}$'s, for $1 \le j \le m$. We have quite a bit of flexibility in filling out the entries of the $\mathbf{c_j}$'s in positions other than those already filled with 0s and 1s. We would like to show that this allows us to fill in the remaining positions with distinct entries so that the $\mathbf{c_j}$'s can be realized as codewords in an $[N, K, D]$ MDS code.

To construct this MDS code, we will construct an $N \times K$ matrix G over a large enough finite field $\mathbb{F} = \mathrm{GF}(2^f)$ of such that every $K \times K$ submatrix of G has full rank. The linear code defined by such a generator matrix G is MDS, and this will be our promised MDS code.[2] In addition we will prove that there exists a way to fill in the entries other than $0, 1$ in $\mathbf{c_j}$, $1 \le j \le m$, so that the first m columns of G are $\mathbf{c_1}, \ldots, \mathbf{c_m}$. We will then have the desired MDS code which contains the $\mathbf{c_j}$'s as codewords (since the columns of G are certainly codewords of the code defined with generator matrix G).

To do this we take a two-step approach. In the first step we prove that for all large enough fields $\mathbb{F}$ ($|\mathbb{F}| = 2^{\Omega(N)}$ suffices), it is possible to fill in the missing entries of the $\mathbf{c_j}$'s using *distinct* elements from $\mathbb{F} \setminus \{0, 1\}$, such that the $N \times m$ matrix H with m columns $\mathbf{c_1}, \mathbf{c_2}, \ldots, \mathbf{c_m}$ has the property that every $K \times m$ submatrix has rank m. The details of this step are somewhat tedious and are explained in a separate technical "linear- algebraic" lemma (Lemma 4.10), whose statement and proof we defer to the end of this section (specifically, to Section 4.4.3). In the second step, we show that provided the field $\mathbb{F}$ is large enough (again $|\mathbb{F}| = 2^{\Omega(N)}$ suffices), one can add a further $(K - m)$ columns to H to get a matrix G, so that every $K \times K$ submatrix of G has rank K. This matrix G will be the final matrix which is the generator matrix of the desired MDS code. The second step is easy to establish. In fact, filling the remaining $(K - m)$ columns with *random* entries from $\mathbb{F}$ works with high probability. Indeed, the number of $K \times K$ submatrices to consider is $\binom{N}{K} \le 2^N$. Fix one such submatrix, say M. We know that its first m columns are linearly independent (this follows from the property of the matrix H guaranteed by the first step). Suppose they span a space W of dimension m. The probability that for a random choice of the remaining entries, the $(m+1)$'th column lies in W is at most $q^{-(K-m)}$ where $q = |\mathbb{F}|$. In general the probability that the i'th column does not increase the dimension of the column span is at most $q^{-(K-i+1)}$, for $m < i \le K$. By a union bound, the probability that M has rank less than K is at most

$$\sum_{i=m+1}^{K} q^{-(K-i+1)} \le 2/q \ ,$$

for $q \ge 2$. By a union bound over all $K \times K$ submatrices, the probability that G has some $K \times K$ submatrix of rank less than K is at most $2^N \cdot (2/q) < 1$ provided $q = |\mathbb{F}| = 2^{\Omega(N)}$. In particular, this implies that there exists an $N \times K$ matrix with entries in $\mathbb{F}$ that defines

[2] The one sentence proof of this is as follows. If there exists a non-zero codeword of weight at most $(N - K)$, and which has 0's in, say, the first K positions, then the first K rows of G must satisfy a non-trivial linearly dependence, contradicting the fact that the $K \times K$ submatrix defined by the first K rows of G has rank K.

an MDS code. The proof is thus complete modulo Lemma 4.10 which we state and prove in Section 4.4.3. □

Lemma 4.8 *For all $\varepsilon > 0$ and $\forall\ \delta$, $0 < \delta < 1$ there exist infinitely many N for which the following holds. There exists an $[N, K, D]_Q$ linear code C_N with $D \geq \delta N$ and $Q = 2^{O(N)}$ with the property that there exists a Hamming ball in $[Q]^N$ of radius $E = (1 - \sqrt{1-\delta-\varepsilon})N$ that has at least $\lg N$ codewords of C_N.*

Proof: The idea is to use the codes guaranteed by Lemma 4.7 for infinitely many pairs m, s. Note that as m and s tend to infinity the relative radius of the ball $\tau = 1 - s/(m+1)$ and relative distance $\delta = 1 - s(s-1)/(m(m+1))$ of the code roughly satisfy $\tau \simeq 1 - \sqrt{1-\delta}$. This essentially gives us the claimed result, and we just need to account for the error due to integrality constraints.

Fix $\delta > 0$ and $\varepsilon > 0$. For any m such that $m > 2/\varepsilon$, it is easy to see that there exists an s, $2 < s < (m+1)$, for which

$$\delta \leq 1 - \frac{s(s-1)}{m(m+1)} \leq \delta + \varepsilon\ . \tag{4.4}$$

Consider the code C for this choice of m, s guaranteed by Lemma 4.7. The blocklength of C equals $N = \binom{m+1}{s}$. Now, since $s \leq m+1$, we have $\frac{s}{m+1} \geq \sqrt{\frac{s(s-1)}{m(m+1)}}$, and hence

$$\tau = 1 - \frac{s}{m+1} \leq 1 - \sqrt{\frac{s(s-1)}{m(m+1)}} \leq 1 - \sqrt{1-\delta-\varepsilon}$$

(the last step above follows using Equation (4.4)). Thus there are at least $m+1$ codewords of C in a Hamming ball of radius $(1 - \sqrt{1-\delta-\varepsilon})N$. Since $N \leq 2^{m+1}$, we get at least $\lg N$ codewords in a ball of radius $(1 - \sqrt{1-\delta-\varepsilon})N$, as desired. □

With Lemma 4.8 in place, proving the main result of this section, namely that $L^{\text{const}}(\delta) = (1 - \sqrt{1-2\delta})/2$, is fairly straightforward.

Proof of Theorem 4.3: The proof uses the idea of code concatenation and the reader might want to recall this notion from Chapter 2. Concatenate the $[N, K, D]_Q$ code C_N from Lemma 4.8 with the binary Hadamard code of dimension $\lg Q$ and blocklength Q to obtain the concatenated code $\tilde{C}$. The blocklength of $\tilde{C}$ is then $n = NQ = N \cdot 2^{O(N)}$. The relative distance of $\tilde{C}$ is at least $\delta' = \delta/2$. Let $\mathbf{x} = \langle x_1, x_2, \ldots, x_N \rangle \in [Q]^N$ be such that $|B(\mathbf{x}, E) \cap C_N| \geq \lg N$, where $E = (1 - \sqrt{1-\delta-\varepsilon})N$. Define $\mathbf{y} = \mathrm{Had}(x_1) \circ \mathrm{Had}(x_2) \cdots \mathrm{Had}(x_N)$. Since every two distinct codewords of the Hadamard code differ in a fraction $1/2$ of positions, it is easy to see that

$$|B(\mathbf{y}, EQ/2) \cap \tilde{C}| \geq \lg N = \Omega(\lg \lg n)\ .$$

Thus there are super-constant number of codewords in a ball of radius $EQ/2$. Now

$$\frac{EQ/2}{NQ} = \frac{E}{2N} = \frac{(1 - \sqrt{1-\delta-\varepsilon})}{2} = \frac{(1 - \sqrt{1-2\delta'-\varepsilon})}{2}\ ,$$

and since $\delta(\tilde{C}) \geq \delta'$, we get that there exists a code family $\mathcal{C}$ with a super-constant number of codewords in a ball of relative radius $(1 - \sqrt{1-2\delta(\mathcal{C})-\varepsilon})/2$. Since $\varepsilon > 0$ was arbitrary, we get that $L^{\text{const}}(\delta) = (1 - \sqrt{1-2\delta})/2$, as desired. □

The lemma below follows from the above proof. Its rather special form might be mysterious now, but it will only be used in the code constructions of Section 4.6, and is not necessary for the results of this section. We record it here since its proof follows using the code construction and ideas from the proofs of Lemmas 4.7 and 4.8. The main features of it that will be useful in Section 4.6 are that its dimension can be a multiple of any desired integer k, and that it can have several codewords within a ball of radius close to the Johnson bound on list decoding radius. The code will be used in Section 4.6 as an inner code in a concatenation scheme with outer Reed-Solomon code to obtain a code with super-polynomial number of codewords at close to the Johnson radius, modulo some technical conditions.

Lemma 4.9 *For all $\varepsilon > 0$, the following holds. For all large enough integers k and every δ, $0 < \delta < 1/2$, there exist an integer $b \le 2^{O(2^k)}$, and a binary linear code $C^{\rm bin}_{k,\delta}$ that has the following properties:*

(i) It has dimension kb, blocklength $n \le 2^{O(k\sqrt{b})}$, and relative distance δ.

(ii) There exists a Hamming ball of radius $(1 - \sqrt{1-2\delta-\varepsilon})n$ which contains at least 2^k codewords of $C^{\rm bin}_{k,\delta}$.

Proof Sketch: Let us use Lemma 4.7 with the choice $m = 2^k - 1$. This gives us an MDS code of blocklength $n_0 \le 2^{2^k}$, relative distance 2δ and dimension $b_0 = (1-2\delta)n_0 + 1$ over a field $\mathrm{GF}(2^{kn_0})$ (since k is a sufficiently large integer, such a code exists over $\mathrm{GF}(2^{kn_0})$). Now, arguing as in Lemma 4.8, the code will contain at least 2^k codewords in a Hamming ball of radius $(1-\sqrt{1-2\delta-\varepsilon})n_0$. We then concatenate this code with a binary Hadamard code of dimension kn_0. The relative distance of the binary concatenated code will be δ, its dimension kb where $b \stackrel{\rm def}{=} b_0 n_0$, and its blocklength will be $n = n_0 2^{kn_0} \le 2^{O(k\sqrt{b})}$ (since $b = b_0 n_0$ and $b_0 = \Theta(n_0)$). Note that $b \le n_0^2 \le 2^{O(2^k)}$. Furthermore, as argued in the proof of Theorem 4.3, due to the corresponding property of the MDS code, the concatenated code will contain at least 2^k codewords in a ball of radius $\frac{1}{2}(1-\sqrt{1-2\delta-\varepsilon})n$, as claimed. $\square$

4.4.2 Related Constructions

The paper of Justesen and Hϕholdt [JH01] also gives a construction of Reed-Solomon codes of rate r (say, over $\mathrm{GF}(2^m)$) with $n = 2^m - 1$ codewords at the Johnson radius $(1-\sqrt{r})n$ (where n is the blocklength of the code), for certain values of the rate r.[3] As in the above construction, we can then concatenate such a Reed-Solomon code with a binary Hadamard code of dimension m to get a linear code of blocklength N with about $\lg N$ codewords at the Johnson radius. In fact the resulting proof is simpler than that of Lemmas 4.7 and 4.8. However, such a result will only apply for certain values of the relative distance (since the necessary Reed-Solomon code from [JH01] is only shown to exist for certain values of the rate/relative distance). Since our focus was on obtaining a result for the entire range $0 < \delta < 1/2$ for the relative distance, we had to go through the more complicated construction of Lemma 4.7.

More recently, Justesen [Jus01] has shown that certain other families of cyclic codes also have about n codewords at the Johnson radius. But once again these results only apply to

[3]This construction is also of interest to the problem of list decoding Reed-Solomon codes, as it provides some evidence to the "tightness" of the Johnson radius for list decoding with constant-sized lists for Reed-Solomon codes. More on this later when we discuss Reed-Solomon decoding in Chapter 6.

a certain set of values of the relative distance, and it is not clear how to extend them for every value of the relative distance.

4.4.3 The technical "linear-algebraic" lemma

We now state and prove the technical lemma used in the proof of Lemma 4.7. The proof is not difficult, but is somewhat long. Therefore, we deliberately moved it to the end of this section so that only the most persevering of readers will need to read the proof. Actually, we believe that the lemma should have a significantly simpler and shorter proof, and we encourage the reader to try and find an alternate proof.

Lemma 4.10 *For the vectors $\mathbf{c_1}, \mathbf{c_2}, \ldots, \mathbf{c_m}$ partially defined as in Equation (4.2), there is a way to fill in the remaining entries using distinct field elements from $\mathbb{F} \setminus \{0,1\}$, for any field $\mathbb{F}$ that satisfies $|\mathbb{F}| \geq 2^{\Omega(N)}$, such that the following holds. The $N \times m$ matrix H with columns $\mathbf{c_1}, \mathbf{c_2}, \ldots, \mathbf{c_m}$ has the property that every $K \times m$ submatrix of H has rank m over the field $\mathbb{F}$.*

Proof: The particular proof method used here was suggested to us by Madhu Sudan. Consider the matrix H with columns $\mathbf{c_1}, \ldots, \mathbf{c_m}$ and consider filling each of the as yet unfilled entry of the $\mathbf{c_j}$'s with a different "indeterminate". Let us label these indeterminates $x_1, x_2, \ldots, x_T$, where $T \leq mN$. We will then prove that there is a setting of the indeterminates for which each of the $K \times m$ submatrices has rank m.

Let M be any $K \times m$ submatrix of H. We will prove that there exists an $m \times m$ submatrix M' of M whose determinant is non-zero as a polynomial in the x_i's. Note that each such $m \times m$ determinant is a *multilinear* polynomial in the x_i's. Let us give this determinant polynomial a label, say $D(M)$. Once we establish this fact, we can simply consider the polynomial, say P, which is the product all such determinant polynomials $D(M)$ (over all $K \times m$ submatrices M). Any setting of distinct values to the x_i's which makes P evaluate to a non-zero element would then imply a setting of the x_i's for which every $K \times K$ submatrix has some $m \times m$ submatrix with non-zero determinant, or in other words has rank m, as desired. Now P being a product of non-zero polynomials, is itself non-zero. Furthermore, P is a product of $\binom{N}{K}$ multilinear polynomials, and hence the degree of P in each variable is at most $\binom{N}{K} \leq 2^N$. Thus provided $|\mathbb{F}| \geq 2^N + T$, one can set arbitrary distinct values to all but one indeterminate involved in P, say to all but x_1, to give a univariate polynomial $\tilde{P}$ of degree at most 2^N in x_1. $\tilde{P}$ has at most 2^N roots over $\mathbb{F}$, and since $|\mathbb{F}| \geq 2^N + T$, there must exist a value in $\mathbb{F}$ outside the roots of $\tilde{P}$ and the at most $(T-1)$ values given to the indeterminates other than x_1. Giving x_1 such a value implies that P is non-zero for such a setting of the x_i's. Thus, if $|\mathbb{F}| \geq 2^{\Omega(N)}$, there exists a setting of distinct values to the x_i's which makes P evaluate to a non-zero element.

It remains to prove that every $K \times m$ submatrix of H has a $m \times m$ submatrix whose determinant is a *non-zero* multilinear polynomial in the x_i's. Since H already has 0's and 1's filled in at some positions, this fact is somewhat tricky to prove. It would be good to recollect the details of the design construction from Lemma 4.7 and specifically the pattern of 0's and 1's that are already filled in the matrix H. In particular the following easily verified facts will be handy later on in the proof:

Fact 1 Every column of H has exactly $\binom{m-1}{s-2} = (K-1)$ zeroes, and $\binom{m-1}{s-1}$ ones.

Fact 2 No column has 0s or 1s filled in more than $\binom{m-2}{s-3}$ rows out of the rows that have 0s filled in some other column (this is because the sets corresponding to all such rows must contain 3 fixed elements out of $\{0, 1, \ldots, m\}$, and there are clearly $\binom{m-2}{s-3}$ such s-element subsets of $\{0, 1, \ldots, m\}$).

Fact 3 For every two columns, the number of rows in which they both have entries filled in is at most $\binom{m-1}{s-2} = K - 1$.

Now, consider a fixed $K \times m$ submatrix, say M, of H, indexed by a subset $R \subseteq [N]$ of K rows of H. We want to prove that M has a $m \times m$ submatrix M' whose determinant is a non-zero polynomial in the x_i's. Our "rough" goal is to prove that one can identify a subset $S \subset R$ of size m such that for each column j, $1 \le j \le m$, there is a *different* row $s_j \in S$ such that entry in the s_j'th row and j'th column is an indeterminate, say x_{i_j}. Loosely, we call such an S a "matching set". (The reason for this terminology will be clear shortly.) The determinant of the $m \times m$ submatrix defined by the rows in such a subset S is then clearly non-zero, as it contains the term $x_{i_1} x_{i_2} \cdots x_{i_m}$, which cannot be canceled by any other term.

We do not always succeed in finding such a matching set, but we *almost* do, and in any case will always be able to conclude that the concerned determinant is a non-zero polynomial. The term "matching set" should be suggestive, as we will find a good set S of rows by finding a matching between the m columns and a subset of the K rows of M in a suitably defined graph. We will prove such a matching exists by appealing to *Hall's theorem.*[4] The relevant bipartite graph, call it B, is defined as follows. It has one side associated with the m columns of H and the other side with set of rows R. There is an edge between $j \in [m]$ and $\ell \in R$, if the entry in row number ℓ and column number j is an indeterminate. We first claim the following:

Claim A. *For any subset $T \subseteq [m]$ with $|T| = p \ge 3$, the neighborhood of T in the above bipartite graph B, denoted $N_B(T)$, contains at least p elements.*

Indeed, if a row $\ell \in R$ is not adjacent to any element of T, then it means that the s-element subset of $\{0, 1, \ldots, m\}$ that corresponds to row ℓ contains T as a subset. The number of such s-element subsets is at most $\binom{m+1-p}{s-p}$. Hence if $p > s$, the neighborhood of T includes all rows in R, so assume $p \le s$. Now

$$K = \binom{m-1}{s-2} + 1 > \frac{(m-1)(m-2)\cdots(m-p+2)}{(s-2)(s-3)\cdots(s-p+1)} \binom{m+1-p}{s-p} .$$

If m is large enough, and $p \ge 3$, it is now easily verified that the number of rows in R adjacent to some element in T is at least p, as claimed.

The above claim states that Hall's condition is satisfied for subsets of $[m]$ of size 3 or more. Hence in the case when it is also satisfied for subsets of $[m]$ of size at most 2, we will have a desired matching of size m in the graph B. The $m \times m$ submatrix of M indexed by the rows used in the matching will then have non-zero determinant.

It remains to deal with the case when Hall's condition is violated for some subset of $[m]$ of size at most 2. In such a situation we will not be able to find a matching for all m columns, but will find a matching at least $(m-2)$ columns and then carefully pick two more

[4] Hall's theorem (see any standard graph theory text, eg., [Har69]) is a classic result that gives a necessary and sufficient condition for a bipartite graph $B = (X, Y, E)$ to have a matching that matches every vertex in X to a distinct vertex in Y (we are assuming $|X| \le |Y|$, of course). The condition is that the neighborhood of every subset $T \subseteq X$, $N_B(T) \subseteq Y$, satisfies $|N_B(T)| \ge |T|$. This is clearly a necessary condition, and Hall's theorem states that it also sufficient. We will refer to the condition "$|N_B(T)| \ge |T|$ for all $T \subseteq X$" as *Hall's condition.*

rows so that we can still argue that the concerned determinant is non-zero. This involves a somewhat tedious case analysis.

If Hall's condition is violated for some subset of size at most 2, then there must exist some $a \in [m]$ such that $|N_B(a)| \leq 1$. This means that in the matrix H, the column number a has either 0s or 1s already filled in at $(K-1)$ or more of the K rows of R. Assume for definiteness that it has 0s or 1s filled in the first $(K-1)$ rows of R. We distinguish two cases:

<u>Case 1</u>: Column a has only 0s in the first $(K-1)$ rows of R.

In this case, a must have either a 1 or an indeterminate in the K'th row. This follows from Fact 1 that we recalled about the pattern of 0s and 1s in H. Furthermore, by Fact 2, all columns other than a can have at most $\binom{m-2}{s-3}$ entries filled with a 0 or 1, among the first $(K-1)$ rows of R. For large enough m, we have

$$(K-1) - \binom{m-2}{s-3} = \binom{m-1}{s-2} - \binom{m-2}{s-3} \geq 2 \ ,$$

and therefore every column in $[m] \setminus \{a\}$ has at least 2 neighbors among the first $(K-1)$ rows of R. Together with Claim A, this implies that Hall's condition holds in the subgraph of B obtained by deleting the last row of R and the column a. Therefore, there must exist a matching of the $(m-1)$ columns other than a into a set R' of $m-1$ distinct rows (among the first $K-1$ rows of R). Let $x_{i_1}, \ldots, x_{i_{m-1}}$ be the indeterminates at the $m-1$ positions of H defined by this matching. Then, since column a must have either a 1 or an indeterminate in the K'th row, the rows in R' together with the K'th row of R define an $m \times m$ submatrix whose determinant is non-zero.

<u>Case 2</u>: Column a has 0s and 1s filled in the first $(K-1)$ rows of R, and not all these entries are 0s.
By permuting rows if necessary, assume that a has either a 1 or an indeterminate in the K'th row of R. In this situation, we further distinguish two subcases:

<u>Case 2.1</u>: Each $b \in [m] \setminus \{a\}$ has at least two neighbors in R, i.e., has 0s and 1s filled in at most $(K-2)$ of the rows in R.

Once again, we claim that if this were the case, the subgraph of B obtained by deleting the column a and the K'th row of R, obeys Hall's condition. Indeed the hypothesis implies that the condition holds for sets of size one, since each $b \in [m] \setminus \{a\}$ has at least one neighbor among the first $K-1$ rows of R. For subsets T of size $p \geq 2$, we apply Claim A to $T \cup \{a\}$ to conclude that $T \cup \{a\}$ has at least $p+1$ neighbors in R, and hence T at least p neighbors among the first $(K-1)$ rows of R (since a is at best adjacent to the last row of K).

Now as in Case 1, the rows corresponding to the matching of size $(m-1)$ from this subgraph together with K'th row of R give us the necessary $m \times m$ submatrix with non-zero determinant.

<u>Case 2.2</u>: There exists some other element $b \in [m]$, $b \neq a$, such that column number b has 0s and 1s filled in the first $(K-1)$ rows of R.

We can assume that b also does not have all 0's in the first $(K-1)$ rows of R, as otherwise we could apply the argument of Case 1 with b instead of a.

Now, at least one of a, b must have an indeterminate in the K'th row of R. This follows from Fact 3. Assume, without loss of generality, that a has an indeterminate, say x_{a^*}, in the K'th row of R. Further, by permuting the first $(K-1)$ rows of R if necessary, we may without loss of generality that the $(K-1)$'th row of R has entry 1 in column b (this is possible since column b does not have all 0's filled in these rows).

Consider the subgraph of B obtained by deleting the last two rows of R and the columns a, b. Using Claim A, it is easily checked that this subgraph satisfies Hall's condition. Therefore it has a matching which matches each of the $(m-2)$ columns other than a, b to distinct rows where they have an indeterminate. Let these indeterminates be $x_{r_1}, x_{r_2}, \ldots, x_{r_{m-2}}$. Now consider the $m \times m$ submatrix M' indexed by the $(m-2)$ rows from the matching together with the last two rows of R. Since a has an indeterminate x_{i^*} in the last row of R, and b has a 1 in the second last row of R, it follows that the determinant of M' has a term $x_{a^*} x_{r_1} \cdots x_{r_{m-2}}$ which cannot be canceled by any other term in the expansion of the determinant of M'. Therefore the determinant of M', as a multilinear polynomial, is non-zero, as we desired to show. This completes the proof in Case 2.2 as well. □

4.5 Proof for polynomial-sized lists

In this section, we will prove the upper bounds on the function $L^{\text{poly}}(\delta)$ claimed in Theorems 4.4 and 4.5. We will first prove Theorem 4.4 which shows that when $\delta = \frac{1}{2} \cdot (1 - o(1))$, one "almost" has a proof of Conjecture 4.1. A modification of this proof will also yield the proof of Theorem 4.5. The proof of Theorem 4.6 under the Artin conjecture in the next section was inspired by the proofs in this section. Hence the reader wishing to read that section should at least skim through the ideas used in proving Theorems 4.4 and 4.5.

We first review the basic definitions and concepts from (Discrete) Fourier analysis that will be used in some of our proofs.

4.5.1 Fourier analysis and Group characters

For this section, it will be convenient to represent Boolean values by $\{1, -1\}$ with 1 standing for FALSE and -1 for TRUE. This has the nice feature that XOR just becomes multiplication. Thus a binary code of blocklength m will be a subset of $\{1, -1\}^m$. There are 2^t linear functions $\chi_\alpha : \{0,1\}^t \to \{1, -1\}$ on t-variables, one for each $\alpha \in \{0,1\}^t$. The function χ_α is defined by $\chi_\alpha(x) = (-1)^{\alpha \cdot x} = (-1)^{\sum \alpha_i x_i}$. Fixing some representation of the field $\mathrm{GF}(2^t)$ as elements of $\{0,1\}^t$, the linear functions χ_α are the *additive characters* of the field $\mathrm{GF}(2^t)$, and can also be indexed by elements $\alpha \in \mathrm{GF}(2^t)$. We will do so in the rest of the paper. We also have, for each $y \in \mathrm{GF}(2^t)$, $\sum_\alpha \chi_\alpha(y)$ equals 0 if $y \neq 0$ and equals 2^t if $y = 0$, where the summation is over all $\alpha \in \mathrm{GF}(2^t)$.

We can define an inner product $\langle f, g \rangle$ for functions $f, g : \mathrm{GF}(2^t) \to \mathbb{R}$ as $\langle f, g \rangle = 2^{-t} \sum_x f(x) g(x)$. We call this inner product the normalized inner product, in contrast to the unnormalized inner product $\sum_x f(x) g(x)$. The linear functions form an orthonormal basis for the space of real-valued functions on $\mathrm{GF}(2^t)$ with respect to the normalized inner product. Thus every real-valued function on $\mathrm{GF}(2^t)$, and in particular every Boolean function $f : \mathrm{GF}(2^t) \to \{1, -1\}$ can be written in terms of the χ_α's as:

$$f(x) = \sum_{\alpha \in \mathrm{GF}(2^t)} \hat{f}_\alpha \chi_\alpha(x) \; . \tag{4.5}$$

The coefficient f_α is called the *Fourier coefficient* of f with respect to α and satisfies $\hat{f}_\alpha = \langle f, \chi_\alpha \rangle = 2^{-t} \sum_x f(x) \chi_\alpha(x)$. If we define the normalized distance between functions f, g as $\delta(f, g) = \Pr_x [f(x) \neq g(x)]$, then $\hat{f}_\alpha = 1 - 2\delta(f, \chi_\alpha)$. The Fourier coefficients of a Boolean function also satisfy Plancherel's identity $\sum_\alpha \hat{f}_\alpha^2 = 1$.

We now define the the Hadamard code in the ± 1 convention to denote binary symbols:

Hadamard code: For any integer t, the binary Hadamard code Had_t of dimension t, encodes t bits (or equivalently elements of $\mathrm{GF}(2^t)$), into elements in $\{1,-1\}^{2^t}$ as follows: For any $x \in \mathrm{GF}(2^t)$, $\mathrm{Had}_t(x) = \langle \chi_\alpha(x) \rangle_{\alpha \in \mathrm{GF}(2^t)}$.

4.5.2 Idea behind the Construction

Since our aim is to prove lower bounds on the list decoding radius we must construct codes with large minimum distance with a large number of codewords in a ball of desired radius. The specific codes we construct are obtained by concatenating an outer Reed-Solomon code over the field $F = \mathrm{GF}(2^t)$ with the Hadamard code Had_t of blocklength 2^t and dimension t. Thus the messages of this code will be degree ℓ polynomials over $\mathrm{GF}(2^t)$ for some ℓ, and such a polynomial P is mapped into the codeword $\langle \mathrm{Had}_t(P(z_1)), \ldots, \mathrm{Had}_t(P(z_{2^t})) \rangle$ where $z_1, z_2, \ldots, z_{2^t}$ is some enumeration of the elements in $\mathrm{GF}(2^t)$.

Let $n = 2^t$. It is easy to see that this code has blocklength 2^{2t} and minimum distance $\frac{1}{2}(1 - \frac{\ell}{n})2^{2t}$. If $\ell = (1 - 2\delta)n$, then the relative minimum distance is δ, and for future reference we denote this code by $\text{RS-H}{\small\text{AD}}_t(\delta)$.

To construct the *received word* (which will be the center of the Hamming ball with a lot of codewords), consider the following. Suppose we could pick an appropriate subset S of $\mathrm{GF}(2^t)$ and construct a Boolean function $f : \mathrm{GF}(2^t) \to \{1,-1\}$ that has large Fourier coefficient $\hat{f}_\alpha$ with respect to α for $\alpha \in S$. Let $\mathbf{v} \in \{1,-1\}^{2^t}$ be the 2^t-dimensional vector consisting of the values of f on $\mathrm{GF}(2^t)$. The word $\mathbf{v}^{|F|}$, i.e., $\mathbf{v}$ repeated $|F|$ times will be the "received word" (the center of the Hamming ball which we want to show has several codewords). Since f has large Fourier support on S, $\mathbf{v}^{|F|}$ will have good agreement with all codewords that correspond to messages (polynomials) P that satisfy $P(z_i) \in S$ for many field elements z_i. By picking for the set S a multiplicative subgroup of $\mathrm{GF}(2^t)$ of suitable size, we can ensure that there are several such polynomials, and hence several codewords in the concatenated code with good agreement with $\mathbf{v}^{|F|}$.

The main technical component of our construction and analysis is the following Theorem which asserts the existence of Boolean functions f with large support on subgroups S of $\mathrm{GF}(2^t)$. We will defer the proof of the theorem to Section 4.5.5, and first use it to prove Theorems 4.4 and 4.5.

Theorem 4.11 *There exist infinitely many integers s with the following property: For infinitely many integers t, there exists a multiplicative subgroup S of $\mathrm{GF}(2^t)$ of size s such that the following holds: For every $\beta \neq 0$ in $\mathrm{GF}(2^t)$ there exists a function $f : \mathrm{GF}(2^t) \to \{1,-1\}$ with $\sum_{\alpha \in \beta \cdot S} \hat{f}_\alpha \geq \sqrt{\frac{s}{3}}$. (Here $\beta \cdot S$ denotes the coset $\{\beta x : x \in S\}$ of S.)*

Remarks: Our proof of the above theorem in fact gives the following additional features which we make use of in our applications of the theorem.

1. The integers s exists with good density; in particular for any integer $k \geq 4$, there exists an s, with $k \leq s < 3k$, that satisfies the requirements of Theorem 4.11.

2. We can also add the condition that there exist infinitely many t including one that lies in the range $s/2 \leq t \leq s$, and the theorem still holds.

For any subset $S \subseteq \mathrm{GF}(2^t)$, one can show that $\sum_{\alpha \in S} \hat{f}_\alpha$ is at most $|S|^{1/2}$ using Plancherel's identity and Cauchy-Schwartz, and Theorem 4.11 shows that we can achieve a sum of $\Omega(|S|^{1/2})$ infinitely often for appropriate multiplicative subgroups S.

4.5.3 Proof of Theorem 4.4

We now employ Theorem 4.11 to prove Theorem 4.4. We in fact prove the following Lemma which clearly establishes Theorem 4.4.

Lemma 4.12 *For every ε, $0 < \varepsilon < 1/2$, there exist infinitely many integers t such that the following holds: Let $N = 2^{2t}$. There exists a vector $\mathbf{r} \in \{1, -1\}^N$ and $\delta = \frac{1}{2}(1 - \Theta((\log N)^{\varepsilon - 1}))$, such that the number of codewords $\mathbf{c}$ of the code* RS-HAD$_t(\delta)$ *with $\Delta(\mathbf{r}, \mathbf{c}) \leq \frac{N}{2}(1 - (1 - 2\delta)^{1/2+\varepsilon})$ is at least $N^{\Omega(\log^{\varepsilon} N)}$.*

Proof: Let s, t be any pair of integers guaranteed by Theorem 4.11 with $t \leq s \leq 2t$ (we are using one of the remarks following Theorem 4.11 here). Let S be a multiplicative subgroup of GF(2^t) of size s and let $f : \mathrm{GF}(2^t) \to \{1, -1\}$ a function guaranteed by Theorem 4.11 such that

$$\sum_{\alpha \in S} \hat{f}_\alpha \geq \sqrt{\frac{s}{3}} \,. \tag{4.6}$$

Let $n = 2^t$, $N = 2^{2t}$ and $p = (n-1)/s$. Note that $s = \Theta(\log N)$ since we have $t \leq s \leq 2t$. Then $S \cup \{0\}$ consists of all elements in GF(2^t) which are p'th powers of some element of GF(2^t).

We first fix the "received word" $\mathbf{r}$. Let $\mathbf{v} \in \{1, -1\}^n$ be the vector $\langle f(x) \rangle_{x \in \mathrm{GF}(2^t)}$ of all values of f. Then $\mathbf{r} = \mathbf{v}^n$, i.e. the vector $\mathbf{v}$ repeated $n = 2^t$ times, one for each position of the outer Reed-Solomon code.

Let δ be a parameter to be specified later and $\ell = (1 - 2\delta)n$. Consider the binary code $\mathbf{C} = \text{RS-HAD}_t(\delta)$ obtained by concatenating a Reed-Solomon code of dimension $\ell + 1 = (1-2\delta)n + 1$ over GF(2^t) with Had$_t$. $\mathbf{C}$ has blocklength N and minimum distance δN. We now want to exhibit several codewords in $\mathbf{C}$ that are "close" to $\mathbf{r}$. We do this by picking codewords in $\mathbf{C}$ at random from some distribution and showing that the agreement with $\mathbf{r}$ is "large" with good probability.

Let $m = \lfloor \ell/p \rfloor$ and consider a message (degree ℓ polynomial over GF(2^t)) P of $\mathbf{C}$ which is of the form $P(x) = R(x)^p$ for a *random* polynomial R of degree at most m over GF(2^t). The Reed-Solomon encoding $(b_1, b_2, \ldots, b_n)$ of P satisfies $b_i \in S \cup \{0\}$ for every i, $1 \leq i \leq n$. It is easy to see that for each i and each $a \in S$, we have $\mathbf{Pr}[b_i = a] = p/n$, and $\mathbf{Pr}[b_i = 0] = 1/n$. Moreover, the choices of b_i are *pairwise independent*.

Now, by definition of the Fourier coefficient, for each i, the Hadamard codeword Had$_t(b_i)$ and the vector $\mathbf{v}$ we constructed above have an unnormalized inner product equal to $n \cdot \hat{f}_{b_i}$ (or equivalently, agree on a fraction $\frac{1+\hat{f}_{b_i}}{2}$ of positions). For any i, $1 \leq i \leq n$, the expected value of $\hat{f}_{b_i}$ satisfies

$$\frac{p}{n} \sum_{\alpha \in S} \hat{f}_\alpha + \frac{1}{n} \hat{f}_0 \geq \frac{(n-1)}{ns} \sum_{\alpha \in S} \hat{f}_\alpha - \frac{1}{n} \geq \frac{1}{s} \sum_{\alpha \in S} \hat{f}_\alpha - \frac{2}{n} \geq \frac{1}{\sqrt{3s}} - \frac{2}{n} \,, \tag{4.7}$$

(the last inequality follows from Equation (4.6)). Let X denote the random variable which is the unnormalized inner product of the codeword (encoding the message $R(x)^p$ for a random polynomial R of degree at most m) with the received vector $\mathbf{r} = \mathbf{v}^n$. By linearity of expectation and using (4.7), we have

$$\mathbf{E}[X] = \sum_{i=1}^{n} \mathbf{E}[n \hat{f}_{b_i}] \geq \frac{N}{\sqrt{3s}} - 2\sqrt{N} \geq \frac{1.1N}{\sqrt{4s}} \tag{4.8}$$

for large enough N (since $s = \Theta(\log N)$). Now, for each i, $1 \le i \le n$, we have

$$\mathbf{E}[\hat{f}_{b_i}^2] \le \frac{p}{n} \sum_{\alpha \in S \cup \{0\}} \hat{f}_\alpha^2 \le 1/s \ .$$

Since the b_i's are evaluations of the polynomial $R(x)^p$ at the n field elements for a random polynomial R, they are pairwise independent. Thus the variance of the random variable X satisfies

$$\mathrm{Var}(X) = \sum_{i=1}^{n} \mathrm{Var}(n\hat{f}_{b_i}) \le \sum_{i=1}^{n} \mathbf{E}[(n\hat{f}_{b_i})^2] \le \frac{n^3}{s} = \frac{N^{3/2}}{s} \ . \tag{4.9}$$

We now use Chebyshev's inequality to prove that the inner product X is greater than $N/\sqrt{4s}$ with probability at least $1/2$. Indeed

$$\begin{aligned}
\mathbf{Pr}[X \le \frac{N}{\sqrt{4s}}] &\le \mathbf{Pr}[X - \mathbf{E}[X] \le -\frac{N}{10\sqrt{4s}}] \le \mathbf{Pr}[|X - \mathbf{E}[X]| \ge \frac{N}{10\sqrt{4s}}] \\
&\le \frac{400s \cdot \mathrm{Var}(X)}{N^2} \le \frac{400}{\sqrt{N}} \\
&< \frac{1}{2} \qquad \text{(for large enough } N\text{)},
\end{aligned}$$

where we have used the lower bound on $\mathbf{E}[X]$ from Equation (4.8) and the upper bound on $\mathrm{Var}(X)$ from Equation (4.9).

Hence the codewords encoding at least $\frac{1}{2} \cdot n^m$ of the polynomials of the form $R(x)^p$ where R is a polynomial of degree at most m, differ from $\mathbf{r}$ in at most $(\frac{1}{2} - \frac{1}{2\sqrt{4s}})N$ codeword positions.

We now pick parameters (namely m, δ) suitably to conclude the result. Recall that $s = \Theta(\log N)$. Picking $m = \lfloor \ell/p \rfloor = s^\varepsilon$, we have

$$(1 - 2\delta) = \frac{\ell}{n} = \Theta(\frac{\ell}{ps}) = \Theta(\frac{m}{s}) = \Theta(s^{\varepsilon - 1}) = \Theta((\log N)^{\varepsilon - 1}) \ .$$

Thus the minimum distance δ (for our choice of m) satisfies $\delta = \frac{1}{2}(1 - \Theta((\log N)^{\varepsilon-1}))$.

Also we have $(1-2\delta)^{1/2+\varepsilon} \simeq s^{(\varepsilon-1)(1/2+\varepsilon)} \le (4s)^{-1/2}$ for large enough N (since $\varepsilon < 1/2$). Thus there exist $\Omega(n^m) = N^{\Omega(\log^\varepsilon N)}$ codewords of RS-HAD$_t(\delta)$ all of which lie in a Hamming ball of radius $\frac{N}{2}(1 - (1-2\delta)^{1/2+\varepsilon})$. Since Theorem 4.11 implies that there are infinitely many choices for t that we could use, we also have infinitely many choices of blocklengths N available for the above construction, and the proof is thus complete. □

4.5.4 Proof of Theorem 4.5

We now turn to obtaining upper bounds on $L_c^{\mathrm{poly}}(\delta)$ for a fixed constant c. One way to achieve this would be to pick $m \simeq 2c$ in the above proof, and then pick $s \simeq 2c/(1-2\delta)$ and this would give (roughly) $L_c^{\mathrm{poly}}(\delta) \le \frac{1}{2}(1 - \left(\frac{1-2\delta}{6c}\right)^{1/2})$. However this upper bound is better than δ only for δ large enough, specifically for $\delta > \frac{1}{2} - \frac{1}{12c}$. We thus have to modify the construction of Lemma 4.12 in order to prove Theorem 4.5. We prove the following lemma which will in turn imply Theorem 4.5. Since our goal was only to establish Theorem 4.5, we have not attempted to optimize the exact bounds in the lemma below.

Lemma 4.13 *For every integer $c \geq 1$ and every δ, $0 < \delta < 1/2$, we have*

$$L_c^{\text{poly}}(\delta) \leq \min_{0 \leq \alpha \leq 1/2-\delta} \Big\{(\delta+\alpha)(1-(\frac{\alpha}{12(2c+1)})^{1/2})\Big\} . \qquad (4.10)$$

We first prove Theorem 4.5 using the above lemma.

Proof of Theorem 4.5: We want to prove $L_c^{\text{poly}}(\delta) < \delta$. Note that when $\delta > \frac{1}{2} - \frac{1}{48(2c+1)}$, setting $\alpha = (1/2 - \delta)$ in Equation 4.10 gives

$$L_c^{\text{poly}}(\delta) \leq \frac{1}{2}(1-(\frac{1-2\delta}{24(2c+1)})^{1/2}) < \delta .$$

When $\delta \leq \frac{1}{2} - \frac{1}{48(2c+1)}$, setting $\alpha = \frac{\delta^2}{48(2c+1)}$ in Equation 4.10 (this is a valid setting since it is less than $1/2-\delta$), we have $L_c^{\text{poly}}(\delta) \leq \delta + \alpha - \delta(\frac{\alpha}{12(2c+1)})^{1/2} < \delta$. Thus we have $L_c^{\text{poly}}(\delta) < \delta$ in either case. □ *(Theorem 4.5)*

Proof of Lemma 4.13: To prove the claimed upper bound on $L_c^{\text{poly}}(\delta)$, we will closely follow the construction from the proof of Lemma 4.12. Let $0 < \delta < 1/2$, $0 \leq \alpha \leq (1/2-\delta)$, and c be given. Define $\alpha' = 2\alpha$ and pick an integer s, $2(2c+1)/\alpha' \leq s < 6(2c+1)/\alpha'$ such that the conditions of Theorem 4.11 are met (we know such an s exists by the first remark following Theorem 4.11). Let t be *any* integer for which a subgroup S of $\mathrm{GF}(2^t)$ exists as guaranteed by Theorem 4.11 (there are once again infinitely many such values of t).

Now we describe the actual construction for a particular δ, α', s, t. Let $n = 2^t$, $N = n^2$ and $p = (n-1)/s$. As in the proof of Lemma 4.12, the code will again be $\text{RS-H}\textsc{ad}_t(\delta)$ (the messages of the code will thus be polynomials over $\mathrm{GF}(2^t)$ of degree at most $\ell = (1-2\delta)n$ and the code has blocklength N). The only change will be in the construction of the received word $\mathbf{r}$. Now, instead of using as received word the vector $\mathbf{v}^n$ (recall that $\mathbf{v}$ was the table of values of the Boolean function f with large Fourier support on a multiplicative subgroup S of $\mathrm{GF}(2^t)$), we will set the first $B = (\ell - \alpha' n) = (1-2\delta-\alpha')n$ blocks of $\mathbf{r}$ to be all zeroes. The last $(n-B)$ blocks of $\mathbf{r}$ will be vectors $\mathbf{v}^{(i)}$, $B < i \leq n$, which will be specified shortly.

Let $m = 2c+1$. We will consider the messages corresponding to polynomials of the form $P(x) = (x-z_1)\cdots(x-z_B)R(x)^p$, where $z_1, \ldots, z_B$ are the B elements of $\mathrm{GF}(n)$ that correspond to the first B positions of the Reed-Solomon code, and R is a random degree m polynomial over $\mathrm{GF}(n)$. Note that $\text{degree}(P) = B + pm = \ell - \alpha' n + \frac{n-1}{s}(2c+1) \leq \ell$ since we picked $s \geq 2(2c+1)/\alpha'$. By the choice of P, the Reed-Solomon encoding $(b_1, b_2, \ldots, b_n)$ of P, will satisfy $b_i = 0$ for $1 \leq i \leq B$, and for each of the last $(n-B)$ positions i, we will have $b_i \in S_i \cup \{0\}$ where S_i is a certain coset of S (recall that S is s-element multiplicative subgroup of $\mathrm{GF}(2^t)$ consisting of all the p'th powers). Specifically $S_i = \beta_i S$ where $\beta_i = (z_i - z_1)\cdots(z_i - z_B)$. Now, for $B < i \leq n$, define $\mathbf{v}^{(i)} \in \{1,-1\}^{2^t}$ to the value of the functions $f^{(i)}$ where $f^{(i)} : \mathrm{GF}(2^t) \to \{1,-1\}$ is a function with $\sum_{\alpha \in S_i} \hat{f}_\alpha^{(i)} \geq \sqrt{s/3}$ as guaranteed by Theorem 4.11 (for the coset $S_i = \beta_i S$).

Now the final codeword corresponding to P will agree with $\mathbf{r}$ in the first nB positions (since both begin with a string of nB zeroes). Using arguments similar to those in the proof of Lemma 4.12, one can show that with probability at least $1/2$, the codeword corresponding to the polynomial P differs from $\mathbf{r}$ in at most

$$E = (n-B)n\Big(\frac{1}{2} - \frac{1}{2\sqrt{4s}}\Big) = N(\delta+\alpha)\Big(1 - \frac{1}{\sqrt{4s}}\Big)$$

positions. Thus there are at least $\frac{1}{2}n^m$ codewords of $\text{RS-H}\textsc{ad}_t(\delta)$ that lie within a ball of radius E around $\mathbf{r}$. Since $N = n^2$, $m = 2c+1$ and $s < 6(2c+1)/\alpha'$, we have $\omega(N^c)$

codewords in a Hamming ball of radius $N(\delta + \alpha'/2)\Big(1 - \sqrt{\frac{\alpha'}{24(2c+1)}}\Big)$, and recalling that $\alpha' = 2\alpha$, the claimed result follows. To conclude, we just reiterate that by Theorem 4.11, for the picked value of s, there are infinitely many values of t (and therefore the blocklength N) for which the code RS-HAD$_t(\delta)$ has the claimed properties. Thus we get an infinite family of codes with the requisite property, and the proof is complete. □ *(Lemma 4.13)*

4.5.5 Proof of Theorem 4.11

The proof proceeds in several steps. We first prove the following Lemma which shows that if a subset S of $\mathrm{GF}(2^t)$ satisfies a certain property, then there exists a Boolean function $f : \mathrm{GF}(2^t) \to \{1, -1\}$ such that $\sum \hat{f}_\alpha$ is large when summed over $\alpha \in S$.

Lemma 4.14 *For any integer t, let S be an arbitrary subset of elements of the field $\mathrm{GF}(2^t)$ such that no four distinct elements of S sum up to 0. Then there exists a function f : $\mathrm{GF}(2^t) \to \{1, -1\}$ with $\sum_{\alpha \in S} \hat{f}_\alpha \geq \sqrt{\frac{|S|}{3}}$.*

Proof: For any set S, the following simple claim identifies the "best" function f for our purposes.

Claim: Define the function $g : \mathrm{GF}(2^t) \to \mathbb{R}$ by $g(x) = \sum_{\alpha \in S} \chi_\alpha(x)$. Then the maximum value of $\sum_{\alpha \in S} \hat{f}_\alpha$ achieved by a boolean function f is exactly $2^{-t} \cdot \sum_x |g(x)|$.
Proof: Indeed

$$2^t \sum_{\alpha \in S} \hat{f}_\alpha = \sum_{x, \alpha \in S} f(x)\chi_\alpha(x) = \sum_x f(x) \sum_{\alpha \in S} \chi_\alpha(x) = \sum_x f(x) g(x) \leq \sum_x |g(x)|$$

with equality holding when f is defined as $f(x) = \mathrm{sign}(g(x))$. □

Thus the above claim "removes" the issue of searching for an f by presenting the "best" choice of f, and one only needs to analyze the behavior of the above character sum function g, and specifically prove a lower bound on $\sum_x |g(x)|$.[5]

To get a lower bound on $\sum_x |g(x)|$, we employ Hölder's inequality which states that

$$\sum_x |h_1(x) h_2(x)| \leq \left(\sum_x |h_1(x)|^p\right)^{1/p} \left(\sum_x |h_2(x)|^q\right)^{1/q} ,$$

for every positive p and q that satisfy $\frac{1}{p} + \frac{1}{q} = 1$. Applying this with $h_1(x) = |g(x)|^{2/3}$, $h_2(x) = |g(x)|^{4/3}$, $p = 3/2$ and $q = 3$ gives

$$\left(\sum_x |g(x)|\right)^{2/3} \left(\sum_x g(x)^4\right)^{1/3} \geq \sum_x g^2(x). \tag{4.11}$$

This inequality is also a consequence of log convexity of the power means (see Hardy, Littlewood, Polya [HLP52]; Theorem 18).

[5] It can be shown that the representation of the field (as a vector space of dimension t over $\mathrm{GF}(2)$) does not affect the value distribution of g, and thus we can pick an arbitrary representation of the field, and the result will be the same.

Now $\sum_x g^2(x) = \sum_{\alpha_1,\alpha_2} \sum_x \chi_{\alpha_1+\alpha_2}(x)$ which equals $|S| \cdot 2^t$ (the inner sum equals 2^t whenever $\alpha_1 = \alpha_2$ and 0 otherwise, and there are $|S|$ pairs (α_1, α_2) with $\alpha_1 = \alpha_2$). Note that this also follows from Plancherel's identity.

Similarly $\sum_x g^4(x) = \sum_{\alpha_1,\alpha_2,\alpha_3,\alpha_4 \in S} \sum_x \chi_{\alpha_1+\alpha_2+\alpha_3+\alpha_4}(x)$ equals $N_{4,S} \cdot 2^t$ where $N_{4,S}$ is the number of 4-tuples in $(\alpha_1, \alpha_2, \alpha_3, \alpha_4) \in S^4$ that sum up to 0. But the property satisfied by S, no four distinct elements of S sum up to 0, and hence the only such 4-tuples which sum up to 0 are those which have two of the α's equal. There are at most $3|S|^2$ such 4-tuples $(\alpha_1, \alpha_2, \alpha_3, \alpha_4)$ with two of the α's equal. Hence $N_{4,S} \le 3|S|^2$, and hence $\sum_x g^4(x) \le 3|S|^2 2^t$. Plugging this into Equation (4.11) we get, when $f(x) = \mathrm{sign}(g(x))$,

$$\sum_{\alpha \in S} \hat{f}_\alpha = \frac{1}{2^t} \sum_x |g(x)| \ge \sqrt{\frac{|S|^3}{3|S|^2}} = \sqrt{\frac{|S|}{3}} \qquad \Box$$

Given the statement of Lemma 4.14, we next turn to constructing subgroups of $\mathrm{GF}(2^t)$ with the property that no four (or fewer) distinct elements of the subgroup sum up to 0. To construct such subgroups, we make use of the following simple lemma about the existence of certain kinds of cyclic codes. For completeness sake, we quickly review the necessary facts about cyclic codes. A binary cyclic code of blocklength n is an ideal in the ring $R = \mathbb{F}_q[X]/(X^n - 1)$. It is characterized by its generator polynomial $g(X)$ where $g(X)|(X^n - 1)$. The codewords correspond to polynomials in R that are multiples of $g(X)$ (the n coefficients of each such polynomial form the codeword symbols). A (binary) cyclic code is said to be maximal if its generator polynomial is irreducible over $\mathrm{GF}(2)$. A BCH code is a special kind of cyclic code whose generator polynomial is defined to be the minimal polynomial that has roots $\beta, \beta^2, \ldots, \beta^{d-1}$. Here β is a primitive n'th root of unity over $\mathrm{GF}(2)$, and d is the "designed distance" of the BCH code.

Lemma 4.15 *Let $k \ge 4$ be any integer. Then there exists an integer s in the interval $[k, 3k)$ such that a maximal binary BCH code of blocklength s and minimum distance at least 5 exists.*

Proof: Let s be an integer of the form $2^f - 3$ in the range $[k, 3k)$ (such an integer clearly exists). Let β be the primitive s'th root of unity over $\mathrm{GF}(2)$ and let h be the minimal polynomial of β over $\mathrm{GF}(2)$. Clearly, $h(\beta^{2^i}) = 0$ for all $i \ge 1$, and hence $h(\beta^2) = h(\beta^4) = 0$. Since β is an s'th root of unity, we have $\beta^s = 1$, or in other words $\beta^{2^f} = \beta^3$. Therefore we also have $h(\beta^3) = 0$. Now the consider the cyclic code C_h of blocklength s with generator polynomial h. It is clearly maximal since h, being the minimal polynomial of β, is irreducible over $\mathrm{GF}(2)$. Also $h(\beta^i) = 0$ for $i = 1, 2, 3, 4$. Using the BCH bound on designed distance (see, for example, Section 6.6 of [vL99]), this implies that the minimum distance of C_h is at least 5, as desired. $\Box$

Lemma 4.16 *Let $k \ge 4$ be any integer. Then there exists an integer s in the interval $[k, 3k)$ with the following property. For infinitely many integers t, including some integer which lies in the range $s/2 \le t \le s$, there exists a multiplicative subgroup S of $\mathrm{GF}(2^t)$ of size s such that no four or fewer distinct elements of S sum up to 0 (in $\mathrm{GF}(2^t)$). Moreover, for any non-zero $\beta \in \mathrm{GF}(2^t)$ this property holds for the coset βS as well.*

Proof: Given k, let $k \le s < 3k$ be an integer for which there exists a binary BCH code C of blocklength s as guaranteed by Lemma 4.15 exists. Such a code is generated by an irreducible polynomial h where $h(x)|(x^s - 1)$. Let $t = \text{degree}(h)$; clearly $t \le s$. Consider the finite field $F = \mathbb{F}_q[X]/(h(X))$ which is isomorphic to $\text{GF}(2^t)$, and consider the subgroup S of size s of F comprising of $\{1, X, X^2, X^3, \ldots, X^{s-1}\}$. The fact that C has distance at least 5 implies that $\sum_{i \in G} X^i$ is not divisible by $h(X)$ for any set G of size at most 4, and thus no four or fewer distinct elements of S sum up to 0 in the field F. This gives us one value of $t \le s$ for which the conditions of Lemma 4.16 are met, but it is easy to see that any multiple of t also works, since the same S is also a (multiplicative) subgroup of $\text{GF}(2^{kt})$ for all $k \ge 1$. In particular we can repeatedly double t until it lies in the range $s/2 \le t \le s$ (note that we had $t \le s$ to begin with). The claim about the cosets also follows easily, since if $a_1 + a_2 + a_3 + a_4 = 0$ where each $a_i \in \beta S$, then $\beta^{-1}a_1 + \beta^{-1}a_2 + \beta^{-1}a_3 + \beta^{-1}a_4 = 0$ as well, and since $\beta^{-1}a_i \in S$, this contradicts the property of S. $\square$

We now have all the ingredients necessary to easily deduce Theorem 4.11.

Proof of Theorem 4.11: Theorem 4.11 now follows from Lemma 4.14 and Lemma 4.16. Note also that the statement of Lemma 4.16 implies the remarks made after the statement of Theorem 4.11. $\square$ *(Theorem 4.11)*

4.6 Super-polynomial list sizes

We will now combine elements of the constructions from the previous two sections to prove Theorem 4.6, which states that, assuming a certain number-theoretic conjecture, the Johnson radius *is* the true bound on list decoding radius as a function of the distance of the code alone.

4.6.1 Proof Idea

As in the previous section, the high level idea is to use an outer Reed-Solomon code over a finite field $\mathbb{F} = \text{GF}(2^l)$ and concatenate it with an inner code that (roughly stated) maps all elements of a multiplicative subgroup S of $\mathbb{F}^*$, consisting of r'th powers for some suitable r, into some small Hamming ball. Hence all codewords of the concatenated code that correspond to evaluations of polynomials that are perfect r'th powers get mapped into a small Hamming ball, giving the desired construction. In the previous section, the inner code was the Hadamard code and the subgroup was picked so that there existed a Boolean function with large Fourier support on elements of the subgroup.

In this section, the inner code C_{in} will be the one guaranteed by Lemma 4.9, with a suitable choice of k in relation to the size of the subgroup S. But now this inner code is only guaranteed to map *some* 2^k messages into a small Hamming ball, and these 2^k messages need not have anything to do with the elements of the subgroup under consideration. However, here comes the crucial idea. These 2^k messages (viewed as vectors over $\mathbb{F}_2$ of suitable length) must contain a linearly independent subset T of size k. Now, if the elements of $S \subseteq \text{GF}(2^l)$ are linearly independent over GF(2), then there must exist a invertible GF(2)-linear map F which maps elements of S into those of T (in some arbitrary ordering of the elements of S, T).[6]. Define C'_{in} to be $C_{\text{in}} \circ F$ (i.e., the encoding under C'_{in} first encodes the message by the invertible map F, and then encodes the resulting string using C_{in}). Note that since F, C_{in} are linear codes, so is C'_{in}. Now C'_{in} will map all elements of S into a small Hamming

[6]The fact that elements of S are linearly independent over GF(2) is independent of the specific representation of $\text{GF}(2^l)$ over GF(2), since this fact only depends on the additive properties of the subgroup S

ball, since C_{in} does so for elements of T. Using C'_{in} as the inner code with outer Reed-Solomon code, and using arguments similar to those used in Lemma 4.12, we can get, after a suitable choice of parameters, a super-polynomial number of codewords in a Hamming ball of radius close to the Johnson bound on list decoding radius.

The astute reader might be puzzled about the need for the seemingly odd number-theoretic conjecture assumed in the hypothesis of Theorem 4.6. This arises because of the need to have a multiplicative subgroup whose elements are linearly independent over GF(2). We do not know how to prove that such subgroups exist for infinitely many sizes without making any number-theoretic assumption. However, this fact is implied by the conjecture that there exist infinitely many primes p for which 2 is a generator of the cyclic group $\mathbb{F}_p^*$ (this is a special case of a more general and famous conjecture known as the Artin conjecture; recall the discussion following the statement of Theorem 4.6).

4.6.2 The Technical Proof

We first record the number-theoretic fact stated at the end of the previous section.

Lemma 4.17 *Let p be an odd prime and let 2 be a generator of the multiplicative group $\mathbb{F}_p^*$ of the finite field $\mathbb{F}_p$. Let α be the primitive p'th root of unity in the field* $\mathrm{GF}(2^{p-1})$ *(this must exist since $2^{p-1} \equiv 1 \bmod p$ by Fermat's little theorem). Consider the subgroup $(1, \alpha, \ldots, \alpha^{p-1})$ of the multiplicative group of* $\mathrm{GF}(2^{p-1})$*. Then the only non-trivial linear dependence among these p elements is: $1 + \alpha + \cdots + \alpha^{p-1} = 0$. In particular, the elements $\alpha, \alpha^2, \ldots, \alpha^{p-1}$ are linearly independent over* GF(2).

Proof: We claim that if 2 is a generator of $\mathbb{F}_p^*$, then the polynomial $g(x) = 1 + x + x^2 + \ldots + x^{p-1}$ is irreducible over GF(2). Indeed, let α be a primitive p'th root of unity over GF(2); then $g(\alpha) = 0$ and therefore the irreducible polynomial h of α over GF(2) (i.e. the polynomial of lowest degree that α satisfies) must divide g. Since $h(\alpha) = 0$, we also have $h(\alpha^{2^i}) = h(\alpha)^{2^i} = 0$ (as we are working over characteristic two). Now, since 2 generates $\mathbb{F}_p^*$ and α is a primitive p'th root, the quantities α^{2^i}, $0 \le i < p-1$ are all distinct. Thus h has $p-1$ distinct zeroes, and hence it has degree at least $(p-1)$. But since h divides g, and g has degree $(p-1)$, we must have that g is a scalar multiple of h, and since we are working over GF(2), this implies $h = g$. Thus the irreducible polynomial of α over GF(2) is $g(x) = 1 + x + x^2 + \ldots + x^{p-1}$. This implies that α satisfies no polynomial equation of degree $(p-2)$ or less, and moreover g is the only polynomial of degree $(p-1)$ that α is a root of. In other words, the relation $1 + \alpha + \ldots + \alpha^{p-1} = 0$ is the only non-trivial linear dependence among the elements of the subgroup $(1, \alpha, \alpha^2, \ldots, \alpha^{p-1})$. $\square$

Corollary 4.18 *Assume that there exist infinitely many primes p for which 2 is a generator of $\mathbb{F}_p^*$. Then for each one of those infinitely many primes the following holds: For every $b \ge 1$, there exists a multiplicative subgroup S' of* $\mathrm{GF}(2^{(p-1)b}) \setminus \{0\}$ *of size p such that the only* GF(2)*-linear dependence among elements of S' is that their sum is zero. In particular, any set of $(p-1)$ elements of the subgroup are linearly independent over* GF(2).

We now state and prove the main theorem of this section. The result of Theorem 4.6 then follows as a corollary.

Theorem 4.19 *Assume that there exist infinitely many primes p for which 2 is a generator of $\mathbb{F}_p^*$. Then, for every δ, $0 < \delta < 1/2$, and every $\varepsilon > 0$, there exist infinitely many N for* *which there exists a binary linear code of blocklength N and minimum distance at least δN that has at least $N^{\Omega(\varepsilon \log\log\log N)}$ codewords in a Hamming ball of radius $\frac{N}{2}(1-\sqrt{1-2\delta-\varepsilon})$.*

Proof: The proof follows the idea outlined in Section 4.6.1. We just need to pick the various parameters appropriately and then carefully bound the number of codewords within a certain Hamming ball of the desired radius.

Let p be a prime such that 2 is a generator of $\mathbb{F}_p^*$. Apply Lemma 4.9 with the choice of $k = (p-1)$ and $\delta = \delta/(1-\varepsilon)$ to get a binary linear code $C' = C^{\text{bin}}_{(p-1),\delta/(1-\varepsilon)}$ of relative distance at least $\delta' \stackrel{\text{def}}{=} \delta/(1-\varepsilon)$. Let the dimension of C' be $(p-1)b$ and let its blocklength be n'; we have $b \le 2^{O(2^p)}$ and $n' \le 2^{O(p\sqrt{b})}$.

Let $S' = (1, \alpha, \alpha^2, \ldots, \alpha^{p-1})$ be a multiplicative subgroup of $\mathrm{GF}(2^{(p-1)b})$ consisting of the r'th powers in $\mathrm{GF}(2^{(p-1)b}) \setminus \{0\}$ where $r = \frac{2^{(p-1)b}-1}{p}$. By Corollary 4.18, the elements in $S = (\alpha, \alpha^2, \ldots, \alpha^{p-1})$ are linearly independent. By Lemma 4.9, there exist a set of at least 2^{p-1} codewords of C' in a Hamming ball B of radius $\frac{n'}{2}(1-\sqrt{1-2\delta'-\varepsilon})$, centered at some $\mathbf{v} \in \mathbb{F}_2^{n'}$. Among these 2^{p-1} codewords there must exist at least $(p-1)$ non-zero codewords corresponding to a set T of $(p-1)$ linearly independent messages (in $\mathbb{F}_2^{(p-1)b}$). Owing to the linear independence of elements of both S and T, there must be an invertible linear transformation F that maps the elements of S into those of T (for some fixed ordering of the elements of S and T). Consider the linear code $C'' = C' \circ F$, i.e., encoding by F followed by the encoding C'. Since F is an invertible linear transformation, C'' is a binary linear code that has the same dimension (namely $(p-1)b$) and minimum distance as C'. Also, clearly the encodings of the elements of S as per C'' all lie in the ball B, which is of radius $\frac{n'}{2}\left(1-\sqrt{1-\frac{2\delta}{1-\varepsilon}-\varepsilon}\right)$.

The final code C^* we use to prove the claim of the theorem will be a Reed-Solomon code over $\mathrm{GF}(2^{(p-1)b})$ of rate ε and blocklength $n = 2^{(p-1)b}$, concatenated with C'' as the inner code. We first quantify the main parameters of C^*. Its blocklength is

$$N = n \cdot n' = 2^{(p-1)b} n' \le 2^{pb} 2^{O(p\sqrt{b})} = 2^{p \cdot 2^{O(2^p)}}$$

(since $b \le 2^{O(2^p)}$). The relative distance of C^* is at least $(1-\varepsilon)\cdot\delta' = \delta$. Since there are infinitely many choices of p by the hypothesis, we can construct such a code C^* for infinitely many blocklengths N. The messages of C^* are in one-one correspondence with degree εn polynomials over $\mathrm{GF}(n)$. Also note that the blocklength n of the outer Reed-Solomon code satisfies $n = N^{\Omega(1)}$.

Now as in the proof of Lemma 4.12, we will establish a lower bound on the number of polynomials whose evaluations at the field elements of $\mathrm{GF}(2^{(p-1)b})$ belong to the set S for "most" of the n field elements. We will do this by considering polynomials of the form $P(x) = R(x)^r$ where R is a random degree $\varepsilon n/r$ polynomial, and estimating the probability that $P(\gamma_i) \in S$ for a large fraction of i's (here we assume $\gamma_1, \ldots, \gamma_n$ are the elements of $\mathrm{GF}(2^{(p-1)b})$). This will imply that for $\mathbf{r} = \mathbf{v}^n$, i.e. the center $\mathbf{v}$ of the ball B repeated n times, once for each Reed-Solomon codeword position, a ball centered at $\mathbf{r}$ of radius "about" $\frac{N}{2}(1-\sqrt{1-2\delta})$ will have several (about $n^{\varepsilon n/r}$) codewords, and for our choice of parameters this will yield a super-polynomial number of codewords within a ball of radius close to the Johnson radius.

The analysis of this random experiment proceeds very similarly to that of Lemma 4.12. The r'th power of any elements of $\mathrm{GF}(n)$ always lies in $S \cup \{0,1\}$. For the choice of $P(x) = R(x)^r$ for a random R, for a fixed γ_j we have

(a) $\mathbf{Pr}[P(\gamma_j) = 0] = 1/n$;

(b) $\mathbf{Pr}[P(\gamma_j) = 1] = r/n$; and

(c) $\mathbf{Pr}[P(\gamma_j) \in S] = (n - r - 1)/n$;

For $1 \le j \le n$, define the indicator random variable I_j as follows:

$$I_j = \begin{cases} 1 & \text{if } P(\gamma_j) \in S \\ 0 & \text{otherwise} \end{cases}$$

Then the random variable $Z = \sum_{j=1}^{n} I_j$ measures the number of positions j, $1 \le j \le n$, for which $P(\gamma_j) \in S$. By (c), we have, for P chosen randomly as above,

$$\mathbf{E}[Z] = n - r - 1 \tag{4.12}$$

Also the I_j's are pairwise independent 0/1 random variables and hence

$$\mathrm{Var}(Z) = \sum_{j=1}^{n} \mathrm{Var}(I_j) = \sum_{j=1}^{n} \mathbf{E}[I_j](1 - \mathbf{E}[I_j]) = \frac{(n-r-1)(r+1)}{n} \ . \tag{4.13}$$

By Chebyshev's inequality we have

$$\begin{aligned} \mathbf{Pr}[Z \le n - r - 1 - \sqrt{n}] &\le \mathbf{Pr}[|Z - \mathbf{E}[Z]| \ge \sqrt{n}] \\ &\le \frac{\mathrm{Var}(Z)}{n} \\ &\le \frac{(n-r-1)(r+1)}{n^2} \\ &\le 2/p \ (\text{since } r + 1 = (n + p - 1)/p \le 2n/p) \\ &\le 1/2 \ (\text{for } p \ge 4). \end{aligned}$$

Hence at least a fraction 1/2 of the polynomials P of the form $P(x) = R(x)^r$ satisfy $P(\gamma_j) \in S$ for at least $(n - r - \sqrt{n})$ values of j. For these polynomials, their encoding by C^* will differ from $\mathbf{r}$ in at most

$$e = (n - r - \sqrt{n})\frac{n'}{2}\Big(1 - \sqrt{1 - \frac{2\delta}{1-\varepsilon} - \varepsilon}\Big) + (r + \sqrt{n})n'$$

places. Since $n \le 2^{p \cdot 2^{O(2^p)}}$ and $r = (n-1)/p$, we have $r = o(n)$ and hence $(r + \sqrt{n})n' = o(nn') = o(N)$. Thus

$$e \le \frac{N}{2}\Big(1 - \sqrt{1 - \frac{2\delta}{1-\varepsilon} - \varepsilon}\Big) + o(N) \ . \tag{4.14}$$

The number of such polynomials whose encodings differ from $\mathbf{r}$ in at most e positions is at least $\frac{1}{2} \cdot n^{\varepsilon n/r} \ge \frac{1}{2} \cdot n^{\varepsilon p}$. Since $n = N^{\Omega(1)}$ and $N \le 2^{p \cdot 2^{O(2^p)}}$, we have $p = \Omega(\log \log \log N)$. The number of codewords of C^* within a Hamming distance of e from $\mathbf{r}$ is therefore $N^{\Omega(\varepsilon \log \log N)}$.

Since $\varepsilon > 0$ was arbitrary, from Equation (4.14) the result claimed in the theorem follows. □

Letting $\varepsilon \to 0$, the bound in Theorem 4.19 approaches the Johnson radius $J(\delta) = (1 - \sqrt{1 - 2\delta})/2$. As a corollary, therefore, we get Theorem 4.6, our main result of this section.

4.7 Notes and Open Questions

The study of constant-weight codes has been the subject of a lot of research, and lower bounds on the rate of constant-weight codes imply certain limits on the list decodability of codes (by considering the case when the received word is $\mathbf{0}$). The result of [GRS95] which we stated in Proposition 4.1 and which shows the tightness of the Johnson bound for general non-linear codes follows this spirit. Constant-weight codes are however not linear, and the question of the tightness of the Johnson bound for linear codes seems to have been pursued only recently.

The work of Justesen and Hϕholdt [JH01] was primarily motivated by the question of limits of list decodability of Reed-Solomon codes. They also showed that the Johnson bound is tight for list decoding with constant-sized lists for certain MDS codes. And, as we did in Section 4.4 of this chapter, one can use their results to prove a similar result for binary codes as well, though one has to perform a somewhat careful analysis of their construction, and in particular, one needs an explicit upper bound on the alphabet size of the outer MDS code.

Obtaining a fixed polynomial or super-polynomial number of codewords in a small Hamming ball has seen comparatively much less success. Prior to the results of [GHSZ00], which are discussed in Section 4.5 of this chapter, it was not even known that one can find a fixed polynomial number of codewords within a ball of relative radius strictly smaller than the relative distance (i.e., the statement of Theorem 4.5). The results of [JH01, Jus01] obtain a linear number of codewords at the Johnson radius for Reed-Solomon and certain cyclic codes, but these results appear only for a certain set of relative distances δ, and not for every value of δ in the range $0 < \delta < 1$.

The results of Section 4.6 will appear in [Gur01a]. Two of the obvious open questions concerning this chapter are:

Open Question 4.1 *Prove the result of Theorem 4.6 (i.e. $L^{\rm poly}(\delta) = (1 - \sqrt{1-2\delta})/2$) without making any number-theoretic assumption. A good first step would be to unconditionally prove that $L^{\rm poly}(\delta) < \delta$ for every δ, $0 < \delta < 1/2$.*

Open Question 4.2 *For $0 < \delta < 1/2$, does there exist a linear code of relative distance δ with an* exponential *(and not just super-polynomial) number of codewords within a Hamming ball of relative radius equal to (or close to) the Johnson radius $J(\delta)$? As a first step, can one have exponentially many codewords within a ball of relative radius strictly less than δ?*

Finally, our binary code constructions from Theorems 4.3, 4.5 and 4.6 all have vanishing rate. This is all right for the applications in this chapter since we are only concerned about the distance vs. list decodability relation and are not directly concerned about the rate. However, it will still be interesting to obtain asymptotically good code constructions to prove the results of this chapter.

Open Question 4.3 *Can one prove analogs of Theorems 4.3, 4.5 and 4.6 with the additional requirement of the concerned code constructions being asymptotically good ?*

Note that a larger rate should intuitively only help us, since there are more codewords in all and thus it should be easier to have several of them within a Hamming ball of relatively small radius. However, we cannot simply add more codewords to increase the rate, since we also have to maintain a certain distance property, and obtaining high rate, large distance and small list decoding radius simultaneously is a non-trivial task. We expect that by using an outer AG-code instead of a Reed-Solomon code one should be able to prove Theorem 4.5 with asymptotically good codes. Using a similar idea, the construction of Section 4.6 can be viewed as "evidence" towards the fact that a proof of Theorem 4.3 for constant-sized lists with asymptotically good codes will lead to a proof of Theorem 4.6 for super-polynomial list sizes with codes of non-vanishing rate.

The material in this chapter appears in [GHSZ00, JH01, Gur01a].

Chapter 5

List decodability Vs. Rate

Once you eliminate the impossible, whatever remains,
no matter how improbable, must be the truth.
Sherlock Holmes (by Sir Arthur Conan Doyle)

5.1 Introduction

In the previous two chapters, we have seen on the one hand that any code of distance d can be list decoded up to its Johnson radius (which is always greater than $d/2$). On the other hand, we have seen that, in general, the list decoding radius (for polynomial-sized lists), purely as a function of the distance of the code, cannot be larger than the Johnson radius. Together these pose limitations to the performance of list decodable codes if one *only* appeals to the distance-LDR relation of the code in order to bound its list decoding radius. To present a concrete example, these imply that one can use a binary code family of relative distance δ to list decode a fraction $(1 - \sqrt{1 - 2\delta})/2$ of errors, but no better (in general). Hence, to list decode a fraction $(1/2 - \varepsilon)$ of errors, one needs binary codes of relative distance $(1/2 - O(\varepsilon^2))$. The best known explicit constructions of code families of such high relative distance achieve a rate of only $O(\varepsilon^6)$ [ABN+92, She93], and there is an upper bound of $O(\varepsilon^4 \log(1/\varepsilon))$ for the rate of such code families [MRRW77].

This raises several natural questions. Can one achieve rate better than $\Omega(\varepsilon^4)$ for binary codes that have list decoding radius $(1/2 - \varepsilon)$? Note that the limitation discussed above comes in part from the rate vs. distance trade-off of codes, and in part from bounding the list decoding radius purely as a function of the distance of the code (via the Johnson bound). If one is interested in list-of-L decoding for some large constant L, the parameters that are directly relevant to the problem are list-of-L decoding radius and the rate of the code. Note that the distance of the code does not (at least directly) appear to be relevant to the problem at all. Since we are only interested in list-of-L decoding, why should one use codes optimized for the minimum distance (i.e., the list-of-1 decoding radius)? A closer examination of this question suggests the possibility that by "directly" optimizing the rate as a function of the list decoding radius, one might be able to do better than the two-step method that goes via the distance of the code.

This indeed turns out to be the case, as results in this chapter demonstrate. We will exhibit codes that achieve trade-offs between list decodability and rate which are provably beyond what can be achieved by going via the distance of the code. While the rate vs. distance trade-off is one of the central problems in coding theory and has received lots of attention, the list decoding radius vs. rate question has received much less attention. This chapter studies this trade-off and proves non-trivial lower bounds on the rate of certain list decodable codes. The basic approach is to use the probabilistic method to show the existence of certain codes. The results of this chapter highlight the potential and limits of list decoding, which in turn sets up the stage for the algorithmic results of Part II by indicating the kind of parameters one can hope for in *efficiently* list decodable codes. Furthermore, some of the results provide "good" inner codes for some of our later concatenated code constructions.

5.2 Definitions

The aim of this chapter is to study the trade-offs between list decoding radius and the rate of code families. In order to undertake such a study systematically, we first develop some definitions and notation. It might be of help to the reader to recall the definition of list decoding radius from Section 2.1.4.

Definition 5.1 *For an integer q, real p with $0 \le p \le (1 - 1/q)$, and list size function $\ell : \mathbb{Z}^+ \to \mathbb{Z}^+$, the* rate function *for q-ary codes with list-of-ℓ decoding radius p, denoted $R_{\ell,q}(p)$, is defined to be*

$$R_{\ell,q}(p) = \sup_{\mathcal{C}:\mathsf{LDR}_\ell(\mathcal{C}) \ge p} R(\mathcal{C}) \ . \tag{5.1}$$

where the supremum is taken over all q-ary code families $\mathcal{C}$ with $\mathsf{LDR}_\ell(\mathcal{C}) \ge p$.
When ℓ is the constant function which takes on the value L for some integer $L \ge 1$, we denote the above quantity as simple $R_{L,q}(p)$.
For a family of integer-valued functions $\mathcal{F}$, one defines the quantity

$$R_{\mathcal{F},q}(p) = \sup_{\ell \in \mathcal{F}} R_{\ell,q}(p) \ .$$

Remark: We have the restriction $p \le (1 - 1/q)$ in the above definition, since it is easy to see that a q-ary code family of non-vanishing rate can never be list decoded from beyond a fraction $(1 - 1/q)$ of errors with polynomial-sized lists. We will often omit the subscript q when the alphabet size is clear from context, or when referring to the binary case. Whether the list size subscript is a constant, an integer-valued function, or a family of integer-valued functions will be clear from the context.

Note that $R_{L,q}(p)$ is the best (largest) rate of a q-ary code family which can list decoded up to a fraction p of errors using lists of size L. We next define the rate function for list decoding with arbitrary constant-sized lists.

Definition 5.2 *For an integer q and real p, $0 \le p \le (1 - 1/q)$, the rate function for list decoding by constant-sized lists, denoted $R_q^{\text{const}}(p)$, is defined to be*

$$R_q^{\text{const}}(p) = \lim \sup_{L \to \infty} \{R_{L,q}(p)\} \ .$$

We will also be interested in the analogous rate function $R_{L,q}$ when the codes in consideration are restricted to be linear. This is an interesting case to consider both combinatorially and because linear codes are much easier to represent, encode and operate with.

Definition 5.3 *We define the analogous rate functions $R_{\ell,q}$, $R_{L,q}$ and $R_{\mathcal{F},q}$ when restricted to* linear codes *by $R_{\ell,q}^{\text{lin}}$, $R_{L,q}^{\text{lin}}$ and $R_{\mathcal{F},q}^{\text{lin}}$, respectively. Likewise the function R_q^{const}, when restricted to linear codes, is denoted by $R_q^{\text{const,lin}}$.*

5.3 Main Results

With the definitions of the previous section in place, we now move on to studying the properties of the rate functions $R_{L,q}$ and the like. Firstly, note that $R_{1,q}(p)$ is precisely the best asymptotic rate of a q-ary code family of relative distance $2p$, and its study is one of the most important and still widely open problems in coding theory. Similarly, while a precise understanding of $R_{L,q}$ seems hopeless at this point, we can nevertheless focus on obtaining good upper and lower bounds on this function. And, as the result of Theorem 5.1 below states, the function R_q^{const} is in fact precisely known.

5.3.1 Basic lower bounds

We remark here that the results in this section are proved by analyzing the performance of random codes and showing that a random code of a certain rate has the desired list decodability properties with *very high probability.* In other words, *"most"* codes have the rate vs. list decodability trade-off claimed in this section. The following result was implicit in [ZP82] and was explicitly stated and proved in [Eli91].

Theorem 5.1 ([ZP82, Eli91]) *For every q and every p, $0 \le p \le (1 - 1/q)$, we have*

$$R_q^{\text{const,lin}}(p) = R_q^{\text{const}}(p) = 1 - H_q(p) \tag{5.2}$$

(recall that $H_q(x) = x \log_q(q-1) - x \log_q x - (1-x)\log_q(1-x)$ is the q-ary entropy function).

We will defer the proof of the above result to later in this section. It is easy to verify that $H_q(1 - 1/q - \varepsilon) \simeq 1 - O(\varepsilon^2)$ for small $\varepsilon > 0$, and hence the above result implies, in particular, that for each fixed q, the best rate for families of linear q-ary codes list decodable up to a fraction $(1 - 1/q - \varepsilon)$ of errors is $\Theta(\varepsilon^2)$. Recall that the best rate one could hope for via the "distance and Johnson bound" based approach was about ε^4. The conclusion therefore is that there exist codes which are list decodable well beyond their Johnson radius with small lists, and in fact most codes have this property!

"Capacity-theoretic" interpretation of Theorem 5.1

There is a very nice interpretation of the result of Theorem 5.1 by comparing it with Shannon's theorem on capacity of noisy channels, when applied to the specific case of the q-ary symmetric channel, call it qSC_p. The channel qSC_p transmits a q-ary symbol without distortion with probability $(1 - p)$, and with the remaining probability, distorts it to one of the other $(q - 1)$ symbols, picked uniformly at random. In other words, the probability that symbol α is distorted to symbol β equals $\frac{p}{q-1}$ if $\alpha \ne \beta$, and equals $(1 - p)$ if $\alpha = \beta$. The Shannon capacity of such a channel equals $1 - H_q(p)$. Therefore, one can communicate reliably over this channel at a rate as close to $1 - H_q(p)$ as one seeks, but not at any rate greater than $1 - H_q(p)$.

The channel qSC_p makes an expected fraction p of errors, and in fact for all sufficiently large blocklengths, the fraction of errors will be close to p with overwhelming probability (by the Chernoff-Hoeffding bounds for i.i.d. events). However, Shannon's theorem relies on the fact the (close to) p fraction of errors will be randomly distributed. The result of Theorem 5.1 states that by using list decoding with list size a sufficiently large constant, we can communicate at a rate arbitrarily close to the "capacity" $1 - H_q(p)$, even if the channel corrupts an *arbitrary p fraction* of symbols in an *adversarial manner.*

Thus, list decoding allows us to approach the Shannon capacity *even if the errors are adversarially effected*, provided we use lists of large enough size in the decoding. This view indicates that list decoding can achieve the best performance one can hope for under a standard probabilistic error model even under the much stronger adversarial error model.

Proof of Theorem 5.1

In order to prove Theorem 5.1, we first focus on results that obtain lower bounds on the rate function for list decoding with a fixed list size L. We will then apply these results in the limit of large L to deduce Theorem 5.1. We first prove a lower bound on $R_{L,q}(p)$ for general codes, and will then prove a result for linear codes.

Theorem 5.2 ([Eli91]) *For every q and every p, $0 \le p \le (1 - 1/q)$, we have*

$$R_{L,q}(p) \ge 1 - H_q(p)\Big(1 + \frac{1}{L}\Big) . \tag{5.3}$$

Proof: Fix a large enough blocklength n and set $e = \lfloor np \rfloor$. The idea is to pick a *random* code consisting of $2M$ codewords, where M is a parameter that will be fixed later in the proof. We will show that with high probability by removing at most M of the codewords the resulting code will be (e, L)-list decodable. This is a fairly standard method in coding theory and is called "random coding with expurgation".

The probability that a fixed set of $(L+1)$ codewords all lie in a fixed Hamming sphere (in the space $[q]^n$) of radius e equals $(V_q(n,e)/q^n)^{L+1}$ where $V_q(n,e)$ is the volume of a Hamming sphere of radius e in $[q]^n$. It is well-known that $V_q(n,e) \le q^{H_q(e/n)n} \le q^{H_q(p)n}$ (see for example [vL99, Chapter 1]). Hence this probability is at most $q^{-(L+1)(1-H_q(p))n}$.

Therefore, the expected number N_{bad} of sets of $(L+1)$ codewords which all lie in *some* Hamming sphere of radius e is at most

$$\binom{2M}{L+1} \cdot q^n \cdot q^{-(L+1)(1-H_q(p))n} \le (2M)^{L+1} \cdot q^{-Ln+(L+1)H_q(p)n} . \tag{5.4}$$

Let us pick M so that it is at least the upper bound in (5.4). For example, we can pick

$$M = \lceil q^{(1-(1+1/L)H_q(p))n} 2^{1+1/L} \rceil \ge q^{(1-(1+1/L)H_q(p))n} . \tag{5.5}$$

Then the expected value of N_{bad} is at most M, and therefore there exists a code with $2M$ codewords that has at most M sets of $(L+1)$ codewords that lie in a Hamming ball of radius e. Now, we can remove one codeword from each of these (at most M) subsets of $(L+1)$ codewords that lies in a ball of radius e. This process reduces the size of the code by at most M codewords. After this expurgation, we have a code with at least M codewords which is (e, L)-list decodable. Since $e = \lfloor pn \rfloor$, using (5.5) we get the desired lower bound on $R_{L,q}(p)$. □

We next prove the analog of the above result when restricted to linear codes; the lower bound is much weaker than that for general codes in that one needs very large lists to get close to the limiting rate $R_q^{\text{const}}(p) = 1 - H_q(p)$. The result first appeared implicitly in the work of Zyablov and Pinsker [ZP82].

Theorem 5.3 *For every q and every p, $0 \le p \le (1 - 1/q)$, we have*

$$R_{L,q}^{\text{lin}}(p) \ge 1 - H_q(p)\Big(1 + \frac{1}{\log_q(L+1)}\Big) . \tag{5.6}$$

Proof: The idea is to once again pick a random code (specifically a linear code of block-length n and dimension k) and then argue that with high probability it will have the required (e, L)-list decodability property (as before we set $e = \lfloor pn \rfloor$).

The main problem in applying the argument from the proof of Theorem 5.2 is that a subset of L codewords of a random linear code are no longer mutually independent. A random $[n, k]_q$ linear code $\mathbf{C}$ is picked by picking a random $n \times k$ matrix A over $\mathbb{F}_q$, and the code is given by $\{A\mathbf{x} : \mathbf{x} \in \mathbb{F}_q^k\}$. Define $J = \lceil \log_q(J+1) \rceil$. Now every set of L distinct non-zero messages in $\mathbb{F}_q^k$ contain a subset of at least J messages which are linearly independent over $\mathbb{F}_q$. It is easily verified that such linearly independent J-tuples are mapped to J mutually independent codewords by a random linear code. We can then apply estimates similar to the proof of Theorem 5.2 applied to this subset of J codewords.

We now bound from above the probability that a random linear code $\mathbf{C}$ is *not* (e, L)-list decodable. We first make the following useful observation: A linear code $\mathbf{C}$ is (e, L)-list decodable iff no one of the Hamming balls of radius e around points in $B_q(\mathbf{0}, e)$ contain L or more *non-zero* codewords. The condition is clearly necessary; its also sufficient by linearity. Indeed, suppose there is some $\mathbf{y} \in \mathbb{F}_q^k$ with $|B_q(\mathbf{y}, e) \cap \mathbf{C}| \geq L+1$. Let $\mathbf{c} \in B_q(\mathbf{y}, e) \cap \mathbf{C}$. By linearity, we have $|B_q(\mathbf{y} - \mathbf{c}, e) \cap \mathbf{C}| \geq L + 1$ as well. But $\mathbf{w} = \mathbf{y} - \mathbf{c}$ has Hamming weight at most e, and $B_q(\mathbf{w}, e)$ has at least L *non-zero* codewords.

The probability that codewords corresponding to a fixed J-tuple of linearly independent messages all lie in a fixed Hamming ball $B_q(\mathbf{w}, e)$ is at most $(q^{(H_q(p)-1)n})^J$. Multiplying this by the number of such linearly independent J-tuples of messages and the number of choices for the center $\mathbf{w} \in B_q(\mathbf{0}, e)$, we get that the probability that some J-tuple of linearly independent messages all lie in some Hamming ball of radius e is at most

$$q^{kJ} \cdot q^{H_q(p)n} \cdot q^{(H_q(p)-1)Jn} = q^{-nJ\left(1-(1+1/J)H_q(p)-k/n\right)} . \tag{5.7}$$

Since every set of L non-zero codewords has a subset of J codewords corresponding to the encodings of linearly independent messages, the above also gives an upper bound on the probability that $\mathbf{C}$ is not (e, L)-list decodable. Picking the dimension to be, say, $k = \lfloor (1 - (1 + 1/J)H_q(p))n - \sqrt{n} \rfloor$, we get exponentially small failure probability for random linear codes with rates approaching $1 - (1 + 1/J)H_q(p)$. Hence there exists a linear code family of rate $1 - (1 + 1/J)H_q(p)$ and $\mathsf{LDR}_{L,q} \geq p$, as desired. $\square$

Proof of Theorem 5.1: The lower bounds in both Theorems 5.2 and 5.3 approach $1 - H_q(p)$ as the list size $L \to \infty$. It remains to prove the upper bounds. Clearly $R_q^{\text{const,lin}}(p) \leq R^{\text{const}}(p)$, so it suffices to prove $R^{\text{const}}(p) \leq 1 - H_q(p)$. This is quite straightforward. Let C be a q-ary code of blocklength n and rate $r > 1 - H_q(p)$. Pick a *random* $\mathbf{x} \in [q]^n$ and consider the random variable $X = |B_q(\mathbf{x}, pn) \cap C|$. The expected value of X is clearly $|C| \cdot |B_q(\mathbf{0}, pn)|/q^n$ which is at least $q^{(r+H_q(p)-1)n-o(n)}$. If $r > 1 - H_q(p)$, this quantity is of the form $q^{\Omega(n)}$. Hence a random ball of radius pn has exponentially many codewords. We must therefore have $R^{\text{const}}(p) \leq 1 - H_q(p)$. $\square$

We also record the following result which is obtained by combining the Gilbert-Varshamov bound (for rate vs. distance trade-off) with the Johnson bound on list decoding radius (which gives a certain LDR vs. distance trade-off). Such a result was made explicit for binary codes in [Eli91] — below we state it for general alphabets.

Theorem 5.4 *For every prime power q and every p, $0 \le p \le (1 - 1/q)$, and every integer $L \ge 1$, we have*

$$R^{\mathrm{lin}}_{L,q}(p) \ge 1 - H_q\left(\left(1 - \frac{1}{q}\right)\frac{L}{L-1}\left(1 - \left(1 - \frac{qp}{q-1}\right)^2\right)\right) . \tag{5.8}$$

Proof (Sketch): The Gilbert-Varshamov bound (see, for instance, [vL99, Chapter 5]) implies that there exist q-ary linear code families of relative distance δ and rate R where

$$R \ge 1 - H_q(\delta) . \tag{5.9}$$

(In fact a random linear code achieves this trade-off with high probability.) The result of Theorem 3.1 on the Johnson radius for list decodability implies that a q-ary code of relative distance δ and blocklength n is (pn, L)-list decodable for

$$p = \left(1 - \frac{1}{q}\right)\left(1 - \left(1 - \frac{q}{q-1}\frac{L-1}{L}\delta\right)^{1/2}\right) . \tag{5.10}$$

Combining (5.9) and (5.10) gives us the desired result. □

5.3.2 An Improved lower bound for binary linear Codes

Consider the result of Theorem 5.3 for the case of binary linear codes and when $p = 1/2 - \varepsilon$ (i.e. we wish to correct close to the "maximum" possible fraction of errors). For this case it implies that there exist rate $\Theta(\varepsilon^2)$ families which are list decodable to a fraction $(1/2 - \varepsilon)$ of errors with lists of size $2^{O(\varepsilon^{-2})}$. While the list size is a constant, it is exponential in $1/\varepsilon$ and it is desirable to reduce it to polynomial in $1/\varepsilon$. By appealing to the Johnson radius based bound of Theorem 5.4, one can achieve a list size of $O(1/\varepsilon^2)$ for decoding up to a fraction $(1/2 - \varepsilon)$ of errors, but the rate goes down to $O(\varepsilon^4)$.

Next, we present an improved result for *binary linear* codes which combines the optimal $\Omega(\varepsilon^2)$ rate with $O(1/\varepsilon^2)$ list size. Recall that the result of Theorem 5.2 already implies this for general, non-linear codes, and the following result closes the gap between linear and non-linear codes for list decoding up to a fraction $(1/2 - \varepsilon)$ of errors (closing this disparity was highlighted by Elias [Eli91] as an open question).

As we shall show in Section 5.3.4, a list size of $\Omega(1/\varepsilon^2)$ is really necessary (even for general, non-linear codes), and thus this result is optimal up to constant factors for the case $p = (1/2 - \varepsilon)$.

Theorem 5.5 *For each fixed integer $L \ge 1$, and $0 \le p \le 1/2$, we have*

$$R^{\mathrm{lin}}_{L}(p) \ge 1 - H(p) - \frac{1}{L} , \tag{5.11}$$

where $H(x) = -x \lg x - (1 - x)\lg(1 - x)$ denotes the binary *entropy function.*

Proof: For each fixed integer $L \ge 1$ and $0 \le p < 1/2$ and for all large enough n, we use the probabilistic method to guarantee the existence of a binary linear code $\mathbf{C}$ of blocklength n that is (e, L)-list decodable for $e = pn$, and whose dimension is $k = \lfloor (1 - H(p) - 1/L)n \rfloor$. This clearly implies the lower bound on the rate function for binary linear codes claimed in (5.11).

The code $\mathbf{C} = C_k$ will be built iteratively in k steps by randomly picking the k basis vectors in turn. Initially the code C_0 will just consist of the all-zeroes codeword $b_0 = 0^n$. The code C_i, $1 \le i \le k$, will be successively built by picking a random (non-zero) basis vector b_i that is linearly independent of $b_1, \ldots, b_{i-1}$, and setting $C_i = \mathrm{span}(b_1, \ldots, b_i)$. Thus $\mathbf{C} = C_k$ is an $[n,k]_2$ linear code. We will now analyze the list of L decoding radius of the codes C_i, and the goal is to prove that the list of L decoding radius of $\mathbf{C}$ is at least e.

The key to analyzing the list of L decoding radius is the following potential function S_C defined for a code C of blocklength n:

$$S_C = \frac{1}{2^n} \sum_{x \in \{0,1\}^n} 2^{\frac{n}{L} \cdot |B(x,e) \cap C|} . \tag{5.12}$$

For notational convenience, we denote S_{C_i} be S_i. Also denote by T_x^i the quantity $|B(x,e) \cap C_i|$, so that $S_i = 2^{-n} \sum_x 2^{nT_x^i/L}$.

Let $B = |B(\mathbf{0}, e)| = |B(\mathbf{0}, pn)|$; then $B \le 2^{H(p)n}$ (see for example Theorem (1.4.5) in [vL99, Chapter 1]). Clearly

$$S_0 = \frac{(2^n - B) + B \cdot 2^{n/L}}{2^n} \le 1 + B \cdot 2^{-n(1-1/L)} \le 1 + 2^{n(H(p)-1+1/L)} . \tag{5.13}$$

Now once C_i has been picked with the potential function S_i taking on some value, say $\hat{S}_i$, the potential function S_{i+1} for $C_{i+1} = \mathrm{span}(C_i \cup \{b_{i+1}\})$ is a random variable depending upon the choice of b_{i+1}. We consider the expectation $\mathbf{E}[S_{i+1} | S_i = \hat{S}_i]$ taken over the random choice of b_{i+1} chosen uniformly from outside $\mathrm{span}(b_1, \ldots, b_i)$. For better readability, below we sometimes use $\exp_2(z)$ to denote 2^z.

$$\begin{aligned} \mathbf{E}[S_{i+1} | S_i = \hat{S}_i] &= 2^{-n} \sum_x \mathbf{E}[\exp_2(n/L \cdot T_x^{i+1})] \\ &= 2^{-n} \sum_x \mathbf{E}[\exp_2(n/L \cdot (|B(x,e) \cap C_i| + |B(x,e) \cap (C_i + b_{i+1})|))] \\ &= 2^{-n} \sum_x \Big(\exp_2(n/L \cdot T_x^i) \mathop{\mathbf{E}}_{b_{i+1}} [\exp_2(n/L \cdot T_{x+b_{i+1}}^i)] \Big) \end{aligned} \tag{5.14}$$

where in the second and third steps we used the fact that if $z \in B(x,e) \cap C_{i+1}$, then either $z \in B(x,e) \cap C_i$, or $z + b_{i+1} \in B(x,e) \cap C_i$. To estimate the quantity (5.14), we use the fact that the expectation of a positive random variable taken over b_{i+1} chosen randomly from outside $\mathrm{span}(b_1, \ldots, b_i)$ is at most $(1 - 2^{i-n})^{-1}$ times the expectation taken over b_{i+1} chosen uniformly at random from $\{0,1\}^n$. Using (5.14) we therefore get:

$$\begin{aligned} \mathbf{E}[S_{i+1} | S_i = \hat{S}_i] &\le (1 - 2^{i-n})^{-1} 2^{-n} \sum_x \Big(2^{n/L \cdot T_x^i} \cdot \Big(\frac{1}{2^n} \sum_{y \in \{0,1\}^n} 2^{n/L \cdot T_{x+y}^i} \Big) \Big) \\ &= (1 - 2^{i-n})^{-1} \hat{S}_i \cdot 2^{-n} \sum_x 2^{n/L \cdot T_x^i} \\ &= \frac{\hat{S}_i^2}{(1 - 2^{i-n})} . \end{aligned} \tag{5.15}$$

Applying (5.15) repeatedly for $i = 0, 1, \ldots, k-1$, we conclude that there exists an $[n,k]$ binary linear code $\mathbf{C}$ with

$$S_{\mathbf{C}} = S_k \le \frac{S_0^{2^k}}{\prod_{i=0}^{k-1} (1 - 2^{i-n})^{2^{k-i}}}$$

$$\leq \frac{S_0^{2^k}}{(1-2^{k-n})^k} \leq \frac{S_0^{2^k}}{1-k2^{k-n}} \tag{5.16}$$

since $(1-x)^a \geq 1-ax$ for $x, a \geq 0$. Combining (5.16) with (5.13), we have

$$S_k \leq (1-k2^{k-n})^{-1}(1+2^{n(H(p)-1+1/L)})^{2^k}$$

and using $(1+x)^a \leq (1+2ax)$ for $ax \ll 1$, this gives

$$S_k \leq 2 \cdot (1 + 2 \cdot 2^{k+(H(p)-1+1/L)n}) \leq 6 \tag{5.17}$$

(the last inequality follows since $k = \lfloor (1-H(p)-1/L)n \rfloor$). By the definition of the potential S_k from Equation (5.12), this implies that $2^{n/L \cdot |B(x,e) \cap \mathbf{C}|} \leq 6 \cdot 2^n < 2^{n+3}$, or $|B(x,e) \cap \mathbf{C}| \leq (1+\frac{3}{n})L$ for every $x \in \{0,1\}^n$. If $n > 3L$, this implies $|B(x,e) \cap \mathbf{C}| < L+1$ for every x, implying that $\mathbf{C}$ is (e, L)-list decodable, as desired. $\square$ *(Theorem 5.5)*

Remark: One can also prove Theorem 5.5 with the additional property that the relative distance $\delta(\mathbf{C})$ of the code (in addition to its list -of-L decoding radius) also satisfies $\delta(\mathbf{C}) \geq p$. This can be done, for example, by conditioning the choice of the random basis vector b_{i+1} in the above proof so that $\text{span}(b_1, b_2, \ldots, b_{i+1})$ does not contain any vector of weight less than pn. It is easy to see that with this modification, Equation (5.15) becomes

$$\mathbf{E}[S_{i+1}|\hat{S}_i] \leq \frac{\hat{S}_i^2}{(1-2^{i+H(p)n-n})}.$$

Using exactly similar calculations as in the above proof, we can then guarantee that there exists a code $\mathbf{C}$ of dimension $k = \lfloor (1-H(p)-1/L)n \rfloor$ and minimum distance at least pn that satisfies $S_{\mathbf{C}} = O(1)$, and consequently satisfies $\mathsf{LDR}_L(\mathbf{C}) \geq p$.

Note that Theorem 5.5, as with the results from the previous section, is a non-constructive result, in that it only proves the existence of a code with the desired properties, and does not give an explicit or polynomial time construction. In fact, unlike the results of Theorems 5.1, 5.2 or 5.3, it does not even give a high probability result. (For those who might be aware of such terminology on the probabilistic method, the technique used to prove Theorem 5.5 is called the *semirandom method*.) Also the proof seems to work for the binary case and does not generalize, at least in any obvious fashion, to the q-ary case for $q > 2$. The following specific questions, therefore, remain open:

Open Question 5.1 *Does a random binary linear code have the property claimed in Theorem 5.5 with high probability ?*

Open Question 5.2 *Does an analogous result to Theorem 5.5 hold for q-ary linear codes for $q > 2$? Specifically, does $R_{L,q}^{\text{lin}} \geq 1 - H_q(p) - \frac{1}{L}$ hold for every prime power q ?*

We believe that the answer to both of the questions above is yes. Finally, we note the following capacity-theoretic consequence of Theorem 5.5: there exist binary linear codes of rate within ε of the Shannon capacity of the binary symmetric channel with cross-over probability p, namely within ε of $1-H(p)$, even when the fraction p of errors are effected *adversarially* as opposed to randomly, provided we use list decoding with lists of size $1/\varepsilon$.

5.3.3 Upper bounds on the rate function

So far, all of our results concerning the rate functions R_L and R_L^{lin} established lower bounds on these functions. In other words they proved that codes with a certain list-of-L decoding radius and certain rate exist. We now turn to the questions of upper bounds on these functions, namely results which demonstrate that codes of certain rate and list decodability do not exist. We focus on binary codes for this section.

The result of Theorem 5.1 shows that one can achieve a rate arbitrarily close to the optimum rate $1 - H(p)$ for codes with list decoding radius p, provided one allows the list size L to grow beyond any finite bound (i.e. by letting $L \to \infty$). This raises the question whether one can attain the rate $1 - H(p)$ with any finite list size L. The following result, due to Blinovsky [Bli86, Bli97], proves that the unbounded list size is in fact necessary to approach a rate of $1 - H(p)$; in other words, it proves that $R_L(p)$ is strictly smaller than $1 - H(p)$ for any finite L and $0 < p < 1/2$. The proof of the result is quite complicated and we refer the interested reader to [Bli86, Theorem 3] (see also [Bli97, Chapter 2]).

Theorem 5.6 ([Bli86]) *For every integer $L \geq 1$, and each p, $0 \leq p \leq 1/2$, we have*

$$R_L(p) \leq 1 - H(\lambda) \ , \tag{5.18}$$

where λ, $0 \leq \lambda \leq 1/2$, is related to p by

$$p = \sum_{i=1}^{\lceil L/2 \rceil} \binom{2i-2}{i-1} \frac{(\lambda(1-\lambda))^i}{i} \ . \tag{5.19}$$

Corollary 5.7 *For every $L \geq 1$ and every p, $0 < p < 1/2$, we have $R_L(p) < 1 - H(p)$.*

Proof: It is not difficult to see that, for $0 \leq y \leq 1/2$,

$$\sum_{i=1}^{\infty} \binom{2i-2}{i-1} \frac{(y(1-y))^i}{i} = y \ . \tag{5.20}$$

Indeed, this follows from the fact that the generating function $C(x) = \sum_{n \geq 0} c_n x^n$ for *Catalan numbers*, defined by $c_n = \frac{1}{n+1}\binom{2n}{n}$ for $n \geq 0$, equals $C(x) = (1 - \sqrt{1-4x})/2$. Equation (5.20) above follows with the setting $x = y(1-y)$ in the generating function for Catalan numbers. We therefore have that the λ which satisfies Condition (5.19) is strictly greater than p. Hence, $H(\lambda) > H(p)$, and thus $R_L(p) \leq 1 - H(\lambda) < 1 - H(p)$. $\square$

5.3.4 "Optimality" of Theorem 5.5

Consider the case of list decoding radius close to $1/2$, i.e., the case when $p = 1/2 - \varepsilon$. In this case, Theorem 5.5 implies the existence of binary linear code families $\mathcal{C}$ of rate $\Omega(\varepsilon^2)$ and $\mathsf{LDR}_L(\mathcal{C}) \geq 1/2 - \varepsilon$ for list size $L = O(1/\varepsilon^2)$ (Theorem 5.2 showed the same result for general, non-linear codes). We now argue that in light of Theorem 5.6, this result for binary codes for the case $p = 1/2 - \varepsilon$ is in fact asymptotically optimal. That is, the rate and list size guaranteed by Theorems 5.2 and 5.5 are the best possible up to a constant factor.

By Theorem 5.1, $R_L(p) \leq 1 - H(p)$ for any finite L, and hence for $p = 1/2 - \varepsilon$, we get that the rate can be at most $O(\varepsilon^2)$. It remains to show that in order to have list-of-L decoding radius $(1/2 - \varepsilon)$ and a positive rate, one needs $L = \Omega(\varepsilon^{-2})$. To do this we make use of the result of Theorem 5.6.

The λ that satisfies Condition (5.19) must be at least p. Hence if $p = (1/2 - \varepsilon)$, we have $1/2 \geq \lambda \geq (1/2 - \varepsilon)$. Therefore $\lambda(1-\lambda) \geq 1/4 - \varepsilon^2$.

Now for any integer $\ell \geq 0$ we have

$$\begin{aligned}\sum_{i=\ell+1}^{\infty} \binom{2i-2}{i-1} \frac{(\lambda(1-\lambda))^i}{i} &\geq \binom{2\ell}{\ell} \frac{(\lambda(1-\lambda))^{\ell+1}}{\ell+1} \sum_{j=0}^{\infty} (\lambda(1-\lambda))^j \Big(\frac{2(2\ell+1)}{\ell+2}\Big)^j \\ &= \frac{\ell+2}{\ell+1} \binom{2\ell}{\ell} \frac{(\lambda(1-\lambda))^{\ell+1}}{(\ell+2) - 2(2\ell+1)\lambda(1-\lambda)} \end{aligned} \tag{5.21}$$

where in the first step we use the fact that if $i = \ell + 1 + j$,

$$\frac{\binom{2i-2}{i-1}\frac{1}{i}}{\binom{2\ell}{\ell}\frac{1}{\ell+1}} = 2^j \prod_{s=\ell+1}^{\ell+j} \frac{2s-1}{s+1} \geq \Big(\frac{2(2\ell+1)}{\ell+2}\Big)^j .$$

Together with Condition (5.19) and Equation (5.20) applied with the choice $y = \lambda$, Equation (5.21) above implies

$$\lambda \geq p + \frac{\ell+2}{\ell+1} \binom{2\ell}{\ell} \frac{(\lambda(1-\lambda))^{\ell+1}}{(\ell+2) - 2(2\ell+1)\lambda(1-\lambda)} ,$$

where $\ell = \lceil L/2 \rceil$. Plugging in the above into the bound of Theorem 5.6 and using $\lambda(1-\lambda) \geq 1/4 - \varepsilon^2$, we get, after some straightforward algebraic manipulations,

$$\lambda \geq p + \Omega\Big(\frac{(1-4\varepsilon^2)^{\ell+1}}{\ell^{3/2}\varepsilon^2}\Big) .$$

Since $R_L(p) \leq 1 - H(\lambda)$ by Theorem 5.6, we get

$$R_L(p) \leq 1 - H\Big(p + \Omega\Big(\frac{(1-4\varepsilon^2)^{\ell+1}}{\ell^{3/2}\varepsilon^2}\Big)\Big) . \tag{5.22}$$

In order to have positive rate, the argument to the entropy function $H(\cdot)$ in the above bound must be at most $1/2$. When $p = 1/2 - \varepsilon$, this requires $1/(\ell^{3/2}\varepsilon^2) = O(\varepsilon)$, or $\ell = \Omega(\varepsilon^{-2})$. Since $\ell = \lceil L/2 \rceil$, we needs list size $L = \Omega(\varepsilon^{-2})$, as we desired to show. We record this fact in the following result:

Theorem 5.8 *Let $\varepsilon > 0$ be a sufficiently small constant and let $\mathcal{C}$ be a binary code family of rate r that satisfies $\mathsf{LDR}_L(\mathcal{C}) \geq (1/2 - \varepsilon)$. Then we must have $r = O(\varepsilon^2)$ and $L = \Omega(1/\varepsilon^2)$.*

5.4 Prelude to pseudolinear codes

For $q > 2$, the lower bound on rate we know for list decodable q-ary codes is much weaker for linear codes (Theorem 5.3) than for general codes (Theorem 5.2). We conjecture that there exists an answer to open question 5.2 in the affirmative, however a proof of this fact has been elusive.

Linear codes have the advantage of succinct representation and efficient encoding (for example, using the generator matrix). Thus, they are very attractive from a complexity view-point. This is particularly important for us later on when we will use the codes guaranteed by the results of the previous two sections as inner codes in concatenated schemes.

In light of the fact that the existential results are weaker for linear codes, we introduce the notion of "pseudolinear" codes, which albeit non-linear, still have succinct representations and admit efficient encoding.

The basic idea behind pseudolinear codes is the following: to encode a message $\mathbf{x} \in \mathbb{F}_q^k$, first "map" it into a longer string $\mathbf{h_x} \in \mathbb{F}_q^{k'}$ and then encode $\mathbf{h_x}$ using a suitable $n \times k'$ "generator" matrix A into $A\mathbf{h_x}$. The name pseudolinear comes from the fact that the non-linear part of the mapping is confined to the first step which maps $\mathbf{x}$ to $\mathbf{h_x}$. Of course, to make this useful the mapping $\mathbf{x} \mapsto \mathbf{h_x}$ must be easy to specify and compute – this will be the case; in fact the mapping will be explicitly specified.

The crucial property of pseudolinear codes for purposes of list decodability will be that by taking $k' = O(kL)$, we can ensure that under the mapping $\mathbf{x} \mapsto \mathbf{h_x}$, every set of L distinct non-zero $\mathbf{x}$'s are mapped into a set of L *linearly independent* vectors in $\mathbb{F}_q^{k'}$. Then if we pick a "random" pseudolinear code by picking a random $n \times k'$ matrix A, we will have the property that the codewords corresponding to any set of L non-zero messages will be mutually independent. This "L-wise independence property" can then be used to analyze the list-of-L decoding properties of the random code, in a manner similar to the analysis of a general, random code.

In a nutshell, the above allows us to translate the list-of-L decoding performance of general codes into similar bounds for L-wise independent pseudolinear codes. The big advantage of pseudolinear codes over general codes is their succinct representation (since one only needs to store the "generator" matrix A) and their efficient encoding. They are thus attractive for use as inner codes in concatenated schemes.

To avoid burdening the reader at this stage, the formal definitions relating to pseudolinear codes and the analog of Theorem 5.2 and related results for pseudolinear codes are deferred to Chapter 9 (pseudolinear codes will not be used in the thesis until that point). For now, the reader can take comfort in the fact there is a way to achieve the list decoding performance of general codes with the more structured pseudolinear codes.

5.5 Notes

Initial works [Eli57, Woz58, SGB67, Ahl73] on list decoding investigated the notion on probabilistic channels, and used random coding arguments to explore the average decoding error probability of block codes for the binary symmetric and more general discrete memoryless channels. Combinatorial questions of the nature investigated in this chapter (and in this thesis in general), on the other hand, are motivated by worst-case, not average, error-correcting behavior.

The study of the maximum rate of (e, L)-list decodable codes in the limit of large blocklength n with e/n and L fixed originated in the work of Zyablov and Pinsker [ZP82] who were interested mainly in the use of such codes as inner codes in concatenated schemes. The study of the relation between rate and list decodability was undertaken systematically for the first time by Blinovsky [Bli86] (see also [Bli97]), where non-trivial upper and lower bounds on $R_L(p)$ are obtained. The paper of Elias [Eli91] is a very useful resource on this topic as it presents a nice, limpid survey of the relevant results together with some new results.

The result of Theorem 5.1 was first implicitly observed in [ZP82]. The result of Theorem 5.2 and its proof are from [Eli91]. Theorem 5.3 was first observed in [ZP82]; the proof in this chapter follows the presentation in [Eli91]. The result of Theorem 5.4 is the generalization to the q-ary case of a similar result for binary codes that was observed in [Eli91].

Elias [Eli91] was the first to note the disparity between the results for linear and non-linear codes, and posed the open question whether the requirement of very large lists in Theorem 5.3 for linear codes was inherent or, as he correctly suspected, was an artifact of the proof techniques. The result of Theorem 5.5 for binary linear codes can be viewed as a positive resolution of this question. This result appears in a joint paper of the author with Håstad, Sudan and Zuckerman [GHSZ00].

Recently, Wei and Feng [WF94] obtained rather complicated lower bounds for the function $R_L(p)$ as well as its linear counterpart $R_L^{\rm lin}(p)$. Their bounds are hard to state and do not have simple closed forms. They conjecture that their lower bounds for the linear and non-linear case are identical for every value of the list size. However, they are able to prove this only for list size at most 3.

Upper bounds on the rate function $R_L(p)$ have been studied by Blinovsky [Bli86], and his results were mentioned in Section 5.3.3. For the case of list size $L = 2$, an improvement to the upper bound from Theorem 5.6 appears in [ABL00]. A recent paper by Blinovsky [Bli00] revisits the bounds for the linear and non-linear case from [Bli86], and shows that the lower bound proved for linear codes is weaker than the one for non-linear codes for a list size as small as 5.

The notion of pseudolinear codes was defined and basic combinatorial results concerning them were proven by the author in joint work with Indyk [GI01b].

Combinatorial results of a similar flavor to those discussed in this chapter appear in three other places in this thesis: in Chapter 8 where a generalization of Theorem 5.5 is proven, in Chapter 9 where pseudolinear codes are discussed, and in Chapter 10 where we discuss analogous questions for the case of erasures (instead of the errors case discussed in this chapter). Due to the local nature of the use of these results, we chose not to present them in this chapter, but instead postpone them to the relevant chapters where they are needed.

Part II

Code Constructions and Algorithms

Chapter 6

Reed-Solomon and Algebraic-geometric Codes

I don't consider this algebra,
but this doesn't mean that algebraists can't do it.

Garrett Birkhoff

Interlude: The previous several chapters investigated the combinatorics of list decoding and indicated what fraction of errors one can hope to correct with small lists, as a function of the distance and rate of the code. This indicates the "combinatorial" feasibility of list decoding, but provides no way to turn this into an *efficient* (polynomial time) algorithm that outputs the small list of codewords that differ from a received word in a certain number of positions. (As with unambiguous decoding, the naive search algorithm takes exponential time for interesting families of codes, and the problem of list decoding is clearly at least as hard as unambiguous decoding.) With the combinatorial results in place to guide us in what one can hope for using list decoding, the next several chapters present efficient list decoding algorithms for several families of codes. These results take us well on our way to algorithmically realizing the potential of list decoding and correcting "well beyond" half-the-minimum-distance. ***End Interlude***

6.1 Introduction

In this chapter, we present polynomial time list decoding algorithms for two important classes of algebraic linear codes, namely Reed-Solomon codes and Algebraic-geometric codes. In addition to the importance of these algorithms for their own sake (since these are classical, commonly used codes), they will also be used as crucial subroutines in the decoding algorithms from later chapters. In some sense, the results in this chapter lie at the heart of the algorithmic content of this thesis and will be repeatedly appealed to on several occasions in the chapters that follow. Incidentally, the next chapter will present an abstract unified framework for list decoding "ideal-based codes", which captures the algorithms in this chapter. Nevertheless, we chose to present the algorithm specialized to these codes in this chapter for several reasons including: (a) this was the chronological order of the conception of the algorithms, (b) to enable the reader to read and understand these algorithms without the burden of having to deal with abstract algebraic concepts and terminology, and (c) Reed-Solomon and AG-codes are perhaps the most important instantiations of the "ideal-based codes" anyway.

6.1.1 Reed-Solomon Codes

Reed-Solomon codes are among the most basic, important and well-studied codes in the literature. In addition to their numerous "theoretical applications", Reed-Solomon codes are also used in a wide range of "real-world" applications such as compact discs players, hard disk drives, satellite and wireless communications, etc. We point the reader to [WB99] for detailed information on the various applications of Reed-Solomon codes.

Recall that the family of Reed-Solomon codes yields $[n, k+1, d = n-k]_q$ codes for any $k < n \le q$. The alphabet Σ for such a code is a finite field $\mathbb{F}_q$. The message specifies a polynomial of degree at most k over $\mathbb{F}_q$ in some formal variable x (by giving its $k+1$ coefficients). The mapping $\mathcal{C}$ maps this polynomial to its evaluation at n distinct values of x chosen from $\mathbb{F}_q$. (The code therefore needs an alphabet size $q \ge n$.) The distance property follows immediately from the fact that two distinct degree k polynomials can agree in at most k places.

Unique decoding of an $[n, k+1, n-k]$ Reed-Solomon codes is possible up to $(n-k-1)/2$ errors, since this is the half-the-distance bound. It is, however, a non-trivial task to solve the unique decoding problem in time polynomial in the blocklength n. Surprisingly, a classical algorithm due to Peterson [Pet60] manages to solve this problem in polynomial time, as long as the number of errors e satisfies $e < \frac{n-k}{2}$. Faster algorithms, with running time $O(n^2)$ or better, are also well-known: in particular the classical algorithms of Berlekamp and Massey (cf. [Ber68, MS81] for a description) achieve such running time bounds. Of course, if $e \ge (n-k)/2$, then there may exist several different codewords within distance e of a received word, and so one cannot perform unique decoding. Our interest in this chapter is on list decoding Reed-Solomon codes from errors beyond the half-the-distance barrier.

We know from the combinatorial results of the previous chapters that any Hamming ball of up to the Johnson radius will only have a polynomial number of codewords, and hence efficient list decoding up to this radius is potentially possible. For an $[n, k+1, n-k]$ Reed-Solomon code, the Johnson radius is $(n - \sqrt{kn})$ (this is the bound of Corollary 3.3 from Chapter 3). Thus a nice goal is to match this with an algorithm that list decodes up to $(n - \sqrt{kn})$ errors. However, despite four decades of research on Reed-Solomon codes, this problem was not known to have an efficient solution. In fact, it was not known how to correct asymptotically more errors than half-the-minimum-distance, let alone decoding up to the Johnson radius. In this chapter, we will present an algorithm that list decodes Reed-Solomon codes up to their Johnson radius. This is the first algorithm to do so, and in fact is also the first algorithm to decode beyond half-the-minimum distance for every value of the rate. The algorithm builds upon an earlier algorithm due to Sudan [Sud97a], which in turn is based on ideas from [ALRS99]. (See Figure 6-1 for a graphical depiction of the fraction of errors handled by our algorithm in comparison to the previous ones.)

6.1.2 Algebraic-geometric codes

Algebraic-geometric codes are a class of algebraic codes that include Reed-Solomon codes as a special case. The major drawback of Reed-Solomon codes is that they require an alphabet size at least as large as the blocklength. This is not desirable for many applications where codes over a small alphabet are required. Algebraic-geometric codes of growing blocklength

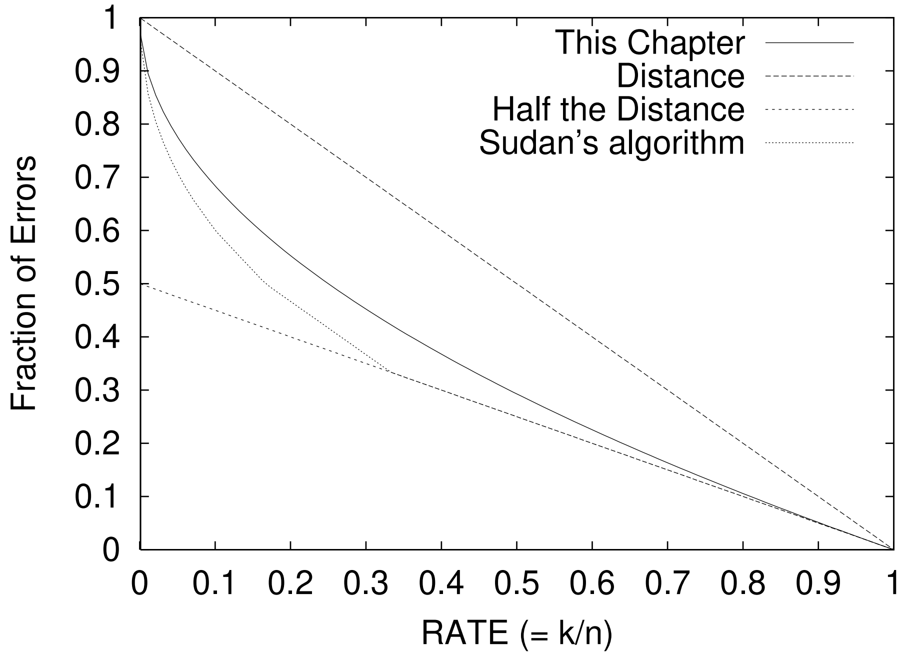

Figure 6-1: Error-correcting capacity plotted against the rate of the code for various Reed-Solomon decoding algorithms

(that tends to infinity) can be defined over fixed, small alphabets. Therefore, they overcome this drawback of Reed-Solomon codes. In fact, algebraic-geometric codes are of significant interest because they yield constructions of codes that beat the Gilbert-Varshamov bound over an alphabet of size q for all $q \geq 49$ which are an even power of a prime [TVZ82]. In other words, for certain choices of the rate and for large enough blocklengths, they achieve a better trade-off between the relative distance and the rate than that achieved by random q-ary codes. Decoding algorithms for algebraic-geometric codes are typically based on decoding algorithms for Reed-Solomon codes. In particular, Shokrollahi and Wasserman [SW99] generalize the algorithm of Sudan [Sud97a] for the case of algebraic-geometric codes. Using a similar approach, we extend our decoding algorithm to the case of algebraic-geometric codes and obtain a list decoding algorithm correcting an algebraic-geometric code of blocklength n and designed distance d^* up to $e < n - \sqrt{n(n-d^*)}$ errors, improving the previously known bound of $n - \sqrt{2n(n-d^*)} - g + 1$ errors (here g is the genus of the algebraic curve underlying the code). The algorithm runs in polynomial time based on a specific (non-standard) polynomial size representation of the underlying algebraic structures.

Applying these results to the best known AG-codes (in terms of the rate vs. distance trade-off) yields constructions of code families efficiently list decodable up to a fraction $(1-\varepsilon)$ of errors (i.e., from very large amounts of noise) over a fixed alphabet of size $O(\varepsilon^{-4})$ and which have rate $\Omega(\varepsilon^2)$.[1] This view of the results is very useful for and motivates some of the results in later chapters when we investigate codes list decodable up to a "maximum" possible fraction of errors (which is $(1-\varepsilon)$ for general codes, and $(1/2-\varepsilon)$ for binary codes).

[1]Reed-Solomon codes offer similar list decodability and rate performance, but have the drawback of very large alphabet size since the alphabet must be at least as large as the blocklength.

6.1.3 Soft-decision decoding algorithms

For both Reed-Solomon and algebraic-geometric codes, we can generalize our algorithms to a "weighted version" that can take weights and can decode as long as a certain weighted condition is satisfied. The weights allow us to encode reliability information about the various symbols into the decoding. Such decoding is referred to as "soft-decision decoding" (or simply, "soft decoding") in the literature. As a simple example, setting a weight to 0 corresponds to deeming that symbol so unreliable as to "erase" it. A more detailed discussion on soft decoding will appear later in the chapter.

6.2 Reed-Solomon codes

We now discuss the list decoding algorithm for Reed-Solomon codes.

6.2.1 Reformulation of the problem

We solve the decoding problem by solving the following (more general) "curve-fitting" or "polynomial reconstruction" problem over a field $\mathbb{F}$: Given n distinct pairs of elements $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ where $x_i, y_i \in \mathbb{F}$, a degree parameter k and an error parameter e, find all univariate polynomials p such that $p(x_i) = y_i$ for at least $n-e$ values of $i \in \{1, \ldots, n\}$. Our algorithm solves this curve-fitting problem for $e < n - \sqrt{nk}$. The algorithm presented here builds upon an earlier algorithm due to [Sud97a] and uses properties of algebraic curves in the plane. The main modification is in the fact that we use the properties of "singularities" of these curves. As in the case of [Sud97a] our algorithm uses the notion of plane curves to reduce our problem to a bivariate polynomial factorization problem over $\mathbb{F}$ (actually only a root-finding problem for univariate polynomials over the rational function field $\mathbb{F}(X)$). This task can be solved deterministically over finite fields in time polynomial in the size of the field or probabilistically in time polynomial in the logarithm of the size of the field. It can also be solved deterministically over the rationals and reals [Gri84, Kal85, Kal92]. Thus our algorithm ends up solving the curve-fitting problem over fairly general fields.

We point out here that the main focus of this chapter is on getting polynomial time algorithms maximizing the number of errors that may be corrected. We do indicate how the algorithms may be implemented with reasonably fast runtimes, and provide pointers to papers that deal with the topic of fast implementation of the various steps in our algorithm.

Other extensions One aspect of interest with decoding algorithms is how they tackle a combination of erasures (i.e., some letters are explicitly lost in the transmission) and errors. Our algorithm generalizes naturally to this case. Another interesting extension of our algorithm is the solution to a *weighted* version of the curve-fitting problem[2]: Given a set of N pairs $\{(x_i, y_i)\}$ and associated non-negative integer weights $w_1, \ldots, w_N$, find all polynomials p such that $\sum_{i:p(x_i)=y_i} w_i > \sqrt{k \cdot \sum_{i=1}^{N} w_i^2}$. We stress here that the x_i's need not be distinct. This generalization is of interest to soft-decision decoding of Reed-Solomon codes – a more detailed discussion on this appears in Section 6.2.10.

[2] The evolution of the solution to the "curve-fitting" problem is somewhat interesting. The initial solutions of Peterson [Pet60] did not explicitly solve the curve-fitting problem at all. The solution provided by Welch and Berlekamp [WB86, Ber96] do work in this setting, even though the expositions there do not mention the curve-fitting problem (see in particular, the description in [GS92]). Their problem statement, however, disallows repeated values of x_i. Sudan's [Sud97a] allows for repeated x_i's but does not allow for repeated pairs of (x_i, y_i). Our solution generalizes this one more step by allowing a weighting of (x_i, y_i)!

Generalized Reed-Solomon Decoding

We now define the problem of decoding a generalization of Reed-Solomon codes (called Generalized Reed-Solomon, or GRS codes). We will then formally define the purely algebraic "polynomial reconstruction" problem. The polynomial reconstruction problem captures the problem of decoding generalized Reed-Solomon codes, and hence also Reed-Solomon codes.

Definition 6.1 (Generalized Reed-Solomon codes) *For parameters n, k and a field $\mathbb{F}_q$ of cardinality q, a vector $\vec{\alpha}$ of* distinct *elements $\alpha_1, \alpha_2, \ldots, \alpha_n \in \mathbb{F}_q$ (hence we need $n \leq q$), and a vector $\vec{v}$ of* non-zero *elements $v_1, \ldots, v_n \in F$, the Generalized Reed-Solomon code $\mathrm{GRS}_{\mathbb{F}_q,n,k,\vec{\alpha},\vec{v}}$, is the function mapping the messages $\mathbb{F}_q^{k+1}$ to code space $\mathbb{F}_q^n$, given by $\mathrm{GRS}_{\mathbb{F}_q,n,k,\vec{\alpha},\vec{v}}(\mathbf{m})_j = v_j \cdot \sum_{i=0}^{k} m_i(\alpha_j)^i$, for $\mathbf{m} = \langle m_0, m_1, \ldots, m_k \rangle \in \mathbb{F}_q^{k+1}$ and $1 \leq j \leq n$.*

(The above generalizes Reed-Solomon codes because we allow arbitrary "multipliers" $v_1, v_2, \ldots, v_n$ for the n codewords positions.)

Problem 1 (Generalized Reed-Solomon decoding)
INPUT: *Field $\mathbb{F}_q$, n, k, $\vec{\alpha}, \vec{v} \in \mathbb{F}_q^n$ specifying the code $\mathrm{GRS}_{\mathbb{F}_q,n,k,\vec{\alpha},\vec{v}}$. A vector $\mathbf{y} \in \mathbb{F}_q^n$ and error parameter e.*
OUTPUT: *All messages $\mathbf{m} \in \mathbb{F}_q^{k+1}$ such that $\Delta(\mathrm{GRS}_{\mathbb{F}_q,n,k,\vec{\alpha},\vec{v}}(\mathbf{m}), \mathbf{y}) \leq e$.*

Problem 2 (Polynomial reconstruction)
INPUT: *Integers k, t and n points $\{(x_i, y_i)\}_{i=1}^n$ where $x_i, y_i \in \mathbb{F}$ for a field $\mathbb{F}$.*
OUTPUT: *All univariate polynomials $p \in \mathbb{F}[x]$ of degree at most k such that $y_i = p(x_i)$ for at least t values of $i \in [n]$.*

The following proposition is easy to establish:

Proposition 6.1 *The generalized Reed-Solomon decoding problem reduces to the polynomial reconstruction problem.*

Proof: It is easily verified that the instance $(\mathbb{F}_q, n, k, \vec{\alpha}, \vec{v}, \mathbf{y}, e)$ of the GRS decoding problem reduces to the instance $(k, n-e, n, \{(\alpha_i, y_i/v_i)\}_{i=1}^n)$ of the polynomial reconstruction problem over the field $\mathbb{F}_q$. □

Organization: In the next few sections, our task will be to solve the polynomial reconstruction problem. We begin with an informal description of the solution next, followed by a formal description in Section 6.2.3. In Section 6.2.4, we prove the correctness of the algorithm. We illustrate the working of the algorithm by a geometric example in Section 6.2.5. In Section 6.2.6, we obtain results for specific list sizes by a careful choice of parameters in the decoding algorithm, and in Section 6.2.7, we present some runtime bounds.

6.2.2 Informal description of the algorithm

We first review Sudan's algorithm [Sud97a] since our algorithm builds upon and generalizes that algorithm. The algorithm has two phases: In the first phase it finds a polynomial Q in two variables which "fits" the points (x_i, y_i), where fitting implies $Q(x_i, y_i) = 0$ for all $i \in [n]$. Then in the second phase it finds all *small degree roots* of Q, i.e., finds all polynomials p of degree at most k such that $Q(x, p(x)) \equiv 0$, or equivalently, such that $y - p(x)$ is a *factor of* $Q(x, y)$. These polynomials p form candidates for the output. The main assertions used to prove the correctness of the algorithm are that:

1. if we allow Q to have a sufficiently large degree then the first phase will be successful in finding such a bivariate polynomial, and

2. if Q and p have low degree in comparison to the number of points where $y_i - p(x_i) = Q(x_i, y_i) = 0$, then $y - p(x)$ will be a factor of Q.

Our algorithm has a similar plan. We will find Q of low "weighted" degree that fits the points. But now we will expect more from the fit. It will not suffice that $Q(x_i, y_i)$ is zero — we will require that every point (x_i, y_i) is a "singularity" of Q. Informally, a singularity is a point where the curve given by $Q(x, y) = 0$ intersects itself. We will make this notion formal as we go along. In our first phase the additional constraints will force us to raise the allowed degree of Q. However we gain (much more) in the second phase. In this phase we look for roots of Q and now we know that p passes through many singularities of Q, rather than just points on Q. In such a case we need only *half* as many singularities as regular points, and this is where our advantage comes from.

Pushing the idea further, we can force Q to intersect itself at each point (x_i, y_i) as many times as we want; in the algorithm described below, this will be a parameter r. There is no limit on what we can choose r to be; only our running time increases with r. We will choose r sufficiently large to handle as many errors as feasible. (In the weighted version of the curve-fitting problem, we force the polynomial Q to pass through different points a different number r_i times, where r_i is proportional to the weight of the point.)

Finally, we come to the question of how to define "singularities". Traditionally, one uses the partial derivatives of Q to define the notion of a singularity. This definition is, however, not good for us since the partial derivatives over fields with small characteristic are not well-behaved. So we avoid this direction and define a singularity as follows: We first shift our coordinate system so that the point (x_i, y_i) is the origin. In the shifted world, we insist that all the monomials of Q with a non-zero coefficient be of sufficiently high degree. This will turn out to be the correct notion. (The algorithm of [Sud97a] can be viewed as a special case, where the coefficient of the constant term of the shifted polynomial is set to zero.)

We first define the shifting method precisely: For a polynomial $Q(x, y)$ and $\alpha, \beta \in F$ we will say that the shifted polynomial $Q_{\alpha,\beta}(x, y)$ is the polynomial given by

$$Q_{\alpha,\beta}(x, y) \stackrel{\text{def}}{=} Q(x + \alpha, y + \beta) \ .$$

Observe that the following explicit relation between the coefficients $\{q_{ij}\}$ of Q and the coefficients $\{(q_{\alpha,\beta})_{ij}\}$ of $Q_{\alpha,\beta}$ holds:

$$(q_{\alpha,\beta})_{ij} = \sum_{i' \geq i} \sum_{j' \geq j} \binom{i'}{i} \binom{j'}{j} q_{i',j'} \alpha^{i'-i} \beta^{j'-j}.$$

In particular observe that the coefficients are obtained by a linear transformation of the original coefficients.

6.2.3 Formal description of the algorithm

We now develop and formally present the list decoding algorithm. We will present a "parameterized" version of the algorithm which works based on some parameters (like list size, number of "singularities" at each point, the maximum number of errors that must be list decoded, etc.). From this we will derive a general decoding condition (viz. Proposition 6.5) for which the algorithm can perform list-of-ℓ decoding, for some parameter ℓ. We will then describe appropriate choices for the various parameters in the algorithm to obtain results for the most interesting cases for us: decoding with constant-sized lists, and decoding with polynomial-sized lists.

Definition 6.2 (weighted degree) *For non-negative weights w_1, w_2, the (w_1, w_2)-weighted degree of the monomial $x^i y^j$ is defined to be $iw_1 + jw_2$. For a bivariate polynomial $Q(x, y)$, and non-negative weights w_1, w_2, the (w_1, w_2)-weighted degree of Q, denoted (w_1, w_2)-wt-deg(Q), is the maximum over all monomials with non-zero coefficients in Q of the (w_1, w_2)-weighted degree of the monomial.*

We now describe our algorithm for the polynomial reconstruction problem.

Algorithm *Poly-Reconstruct*:
Inputs: $n, k, t, \{(x_i, y_i)\}_{i=1}^n$, where $x_i, y_i \in \mathbb{F}$.

Step 0: Compute parameters r, l which satisfy

$$rt > l \text{ and } n\binom{r+1}{2} < \left(\left\lfloor \frac{l}{k} \right\rfloor + 1\right)\left(l + 1 - \frac{k}{2}\left\lfloor \frac{l}{k} \right\rfloor\right) . \tag{6.1}$$

Step 1: Find a polynomial $Q(x, y)$ such that $(1, k)$-wt-deg$(Q) \le l$, i.e., find values (in $\mathbb{F}$) for its coefficients $\{q_{j_1 j_2}\}_{j_1, j_2 \ge 0 : j_1 + k j_2 \le l}$ such that the following conditions hold:[3]

1. At least one q_{j_1, j_2} is non-zero
2. For every $i \in [n]$, if $Q^{(i)}$ is the shift of Q to (x_i, y_i), then all coefficients of $Q^{(i)}$ of total degree less than r are 0. More specifically:

$$\forall i \in [n], \forall\ j_1, j_2 \ge 0, \text{ s.t. } j_1 + j_2 < r,$$
$$q_{j_1 j_2}^{(i)} \stackrel{\text{def}}{=} \sum_{j_1' \ge j_1} \sum_{j_2' \ge j_2} \binom{j_1'}{j_1}\binom{j_2'}{j_2} q_{j_1', j_2'} x_i^{j_1' - j_1} y_i^{j_2' - j_2} = 0.$$

Step 2: Find all polynomials $p \in \mathbb{F}[X]$ of degree at most k such that p is a root of Q (i.e., $y - p(x)$ is a factor of $Q(x, y)$). For each such polynomial p check if $p(x_i) = y_i$ for at least t values of $i \in [n]$, and if so, include p in output list.

End *Poly-Reconstruct*

We will present an analysis of the runtime of the algorithm in Section 6.2.7 along with pointers to the relevant papers. For now, we quickly note that the algorithm can definitely be implemented in polynomial time. This follows since the first step can be accomplished by solving a homogeneous linear system, a task that can be performed in cubic time by Gaussian elimination. The second step can be performed in polynomial time by appealing to a bivariate polynomial factorization algorithm (cf. [Kal85, Gri84, Len85]), though the special nature of the problem in this setting allows for faster solutions, which will be discussed in Section 6.2.7.

[3]We will prove shortly that such a polynomial exists for r, l as in (6.1).

6.2.4 Correctness of the Algorithm

We now prove the correctness of our algorithm assuming the parameters picked by the algorithm satisfy certain constraints. In Section 6.2.6, we will indicate appropriate settings of parameters for which the algorithm achieves useful list decoding performance. In Lemmas 6.2 and 6.3, Q can be *any* polynomial returned in Step 1 of the algorithm.

Lemma 6.2 *If (x_i, y_i) is an input point and p is any polynomial such that $y_i = p(x_i)$, then $(x - x_i)^r$ divides $g(x) \stackrel{\text{def}}{=} Q(x, p(x))$.*

Proof: Let $p_1(x)$ be the polynomial given by $p_1(x) = p(x + x_i) - y_i$. Notice that $p_1(0) = 0$. Hence $p_1(x) = xp_2(x)$, for some polynomial $p_2(x)$. Now, consider $g_1(x) \stackrel{\text{def}}{=} Q^{(i)}(x, p_1(x))$. We first argue that $g_1(x - x_i) = g(x)$. To see this, observe that

$$g(x) = Q(x, p(x)) = Q^{(i)}(x - x_i, p(x) - y_i) =$$

$$Q^{(i)}(x - x_i, p_1(x - x_i)) = g_1(x - x_i).$$

Now, by construction, $Q^{(i)}$ has no coefficients of total degree less than r. Thus by substituting $y = xp_2(x)$ for y, we are left with a polynomial g_1 such that x^r divides $g_1(x)$. Shifting back we have $(x - x_i)^r$ divides $g_1(x - x_i) = g(x)$. $\square$

Lemma 6.3 *If $p(x)$ is a polynomial of degree at most k such that $y_i = p(x_i)$ for at least t values of $i \in [n]$ and $rt > l$, then $y - p(x)$ divides $Q(x, y)$, or equivalently, $Q(x, p(x)) \equiv 0$.*

Proof: Consider the polynomial $g(x) = Q(x, p(x))$. By the definition of weighted degree, and the fact that the $(1, k)$-weighted degree of Q is at most l, we have that g is a polynomial of degree at most l. By Lemma 6.2, for every i such that $y_i = p(x_i)$, we know that $(x - x_i)^r$ divides $g(x)$. Thus if S is the set of i such that $y_i = p(x_i)$, then $\prod_{i \in S}(x - x_i)^r$ divides $g(x)$. (Notice in particular that $x_i \neq x_j$ for any pair $i \neq j \in S$, since then we would have $(x_i, y_i) = (x_i, p(x_i)) = (x_j, p(x_j)) = (x_j, y_j)$.) By the hypothesis $|S| \geq t$, and hence we have a polynomial of degree at least rt dividing g which is a polynomial of degree at most $l < rt$. This can happen only if $g \equiv 0$. Thus we find that $p(x)$ is a root of $Q(x, y)$ (where the latter is viewed as a polynomial in y with coefficients from the ring of polynomials in x). By the division algorithm, this implies that $y - p(x)$ divides $Q(x, y)$. $\square$

All that needs to be shown now is that a polynomial Q as sought for in Step 1 does exist. The lemma below shows this conditionally.

Lemma 6.4 *If*

$$n\binom{r+1}{2} < \left(\left\lfloor \frac{l}{k} \right\rfloor + 1\right)\left(l + 1 - \frac{k}{2}\left\lfloor \frac{l}{k} \right\rfloor\right), \tag{6.2}$$

then a polynomial Q as sought in Step 1 does exist (and can be found in polynomial time by solving a linear system). Furthermore, Condition (6.2) is met if

$$n\binom{r+1}{2} < \frac{l(l+2)}{2k}. \tag{6.3}$$

Proof: Notice that the computational task in Step 1 is that of solving a homogeneous linear system. A non-trivial solution exists as long as the rank of the system is strictly smaller than the number of unknowns. The rank of the system may be bounded from above by the number of constraints, which is $n\binom{r+1}{2}$. The number of unknowns equals the number of monomials of $(1,k)$-weighted degree at most l and this number equals

$$\begin{aligned}\sum_{j_2=0}^{\lfloor \frac{l}{k} \rfloor} \sum_{j_1=0}^{l-kj_2} 1 &= \sum_{j_2=0}^{\lfloor \frac{l}{k} \rfloor} (l+1-kj_2) \\ &= (l+1)\left(\left\lfloor \frac{l}{k} \right\rfloor + 1\right) - \frac{k}{2}\left\lfloor \frac{l}{k} \right\rfloor \left(\left\lfloor \frac{l}{k} \right\rfloor + 1\right) \end{aligned} \tag{6.4}$$

which implies the claimed result. Also the quantity (6.4) is clearly at least

$$\left(\left\lfloor \frac{l}{k} \right\rfloor + 1\right)\left(l+1-\frac{l}{2}\right) \geq \frac{l}{k} \cdot \frac{l+2}{2}$$

which implies the second claim as well. □

We now record the main result of this section which quantifies the performance of the algorithm as a function of the parameters r, t, l.

Proposition 6.5 *Let parameters r, t, l satisfy $rt > l$ and Condition 6.3, or even the weaker Condition 6.2. Then, given any set of n pairs $(x_i, y_i) \in \mathbb{F}^2$, the number of degree k polynomials p that satisfy $p(x_i) = y_i$ for at least t values of i is at most $\lfloor l/k \rfloor$. Moreover the algorithm Poly-Reconstruct with choice of parameters r, t, l, finds and outputs the list of all such polynomials.*

Proof: The correctness of the algorithm, i.e., the fact that it outputs all the relevant polynomials that satisfy $p(x_i) = y_i$ for at least t values of i, follows from Lemmas 6.3 and 6.4. The claimed bound on the number of polynomials follows from the fact that for any such polynomial p, $(y - p(x))$ must be a factor of $Q(x, y)$. The y-degree of Q is at most $\lfloor l/k \rfloor$ since its $(1,k)$-weighted degree is at most l (by the choice of Q). The number of factors $(y - p(x))$ of $Q(x, y)$ is clearly at most the y-degree of Q, and the result follows. □

6.2.5 A "geometric" example

We now present examples that geometrically illustrate how the algorithm works. This will also bring out the necessity for using multiplicities (i.e., the parameter r). The readers who already obtained enough intuition from the analysis of the previous subsection can skip to Section 6.2.6.

To present the examples, we work over the field $\mathbb{R}$ of real numbers. The collection of pairs $\{(x_i, y_i) : 1 \leq i \leq n\}$ then just form a collection of n points in the plane. We will illustrate how the algorithm finds all polynomials of degree one, or in other words lines, that pass through at least a certain number t of the n points. In other words, throughout this section we fix the degree parameter $k = 1$.

Example 1: For the first example, we take $n = 14$ and $t = 5$. The 14 points on the plane are as in Figure 6-2.

We want to find all lines that pass through at least 5 of the above 14 points. Since $k = 1$, the $(1,k)$-weighted degree of a bivariate polynomial is simply its total degree. The

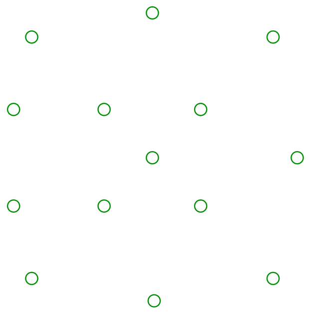

Figure 6-2: Example 1: The set of 14 input points. We assume that the center-most point is the origin and assume a suitable scaling of the other points.

first step of the algorithm must fit a non-zero bivariate polynomial $Q(x, y)$ of total degree l through these 14 points. Let us pick $r = 1$, i.e., we only insist the polynomial Q must have each (x_i, y_i) as a "simple" zero. This gives one constraint for each of the 14 points. Since there are 14 linear constraints in all on the coefficients of the polynomial Q, we can fit a polynomial Q of total degree $l = 4$ (since such a polynomial has $\binom{4+2}{2} = 15 > 14$ coefficients) — this corresponds to Lemma 6.4 applied to this example.

A degree 4 polynomial that passes through the above 14 points is $Q(x, y) = y^4 - x^4 - y^2 + x^2$. To see this pictorially, let us plot the locus of all points on the plane where Q has zeroes. This gives Figure 6-3 below.

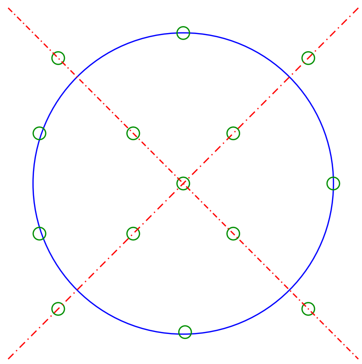

Figure 6-3: A degree 4 fit through the 14 points. The curve is the locus: $y^4 - x^4 - y^2 + x^2 = 0$. The two lines in the picture stretch to infinity, of course.

Note that the two relevant lines that pass through at least 5 points emerge in the picture (these are the dashed lines in the picture). Algebraically, this corresponds to the fact that $Q(x, y)$ factors as $Q(x, y) = (x^2 + y^2 - 1)(y + x)(y - x)$, and the last two factors correspond to the two lines that are the solutions. The fact that the above works correctly, i.e., the fact that the relevant lines must be factors of *any* degree 4 fit through the 14 points, is a consequence of Lemma 6.3 applied to this example (with the choice $l = 4$, $r = 1$ and $t = 5$).

Example 2: For the second example, we take $n = 11$ and $t = 4$. The 11 points on the plane are as in Figure 6-4. We want to find all lines that pass through at least 4 of the above

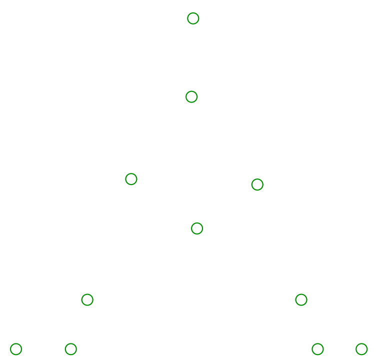

Figure 6-4: Example 2: The set of 11 input points.

11 points. Once again, as in Example 1, we can try and fit a non-zero degree 4 polynomial through these 11 points (degree 3 gives only $\binom{5}{2} = 10$ coefficients, and cannot in general interpolate an arbitrary set of 11 points). However, as we will shortly prove, this strategy, no matter which degree 4 polynomial we fit through the 11 points, will *not work* for this example.

Here is where our general multiplicity based approach kicks in. Specifically, we will now try and fit a polynomial $Q(x, y)$ that has each of the 11 input points as a zero of multiplicity 2. That is, we set the parameter $r = 2$. Now Lemma 6.4 (applied with $n = 11$, $r = 2$) implies that there is a non-zero polynomial Q of degree $l = 7$ that has the required property (since Condition (6.2) is satisfied for these values of n, r, l). Figure 6-5 below is a plot of the locus of zeroes of one such polynomial. Note that the polynomial is a product of seven

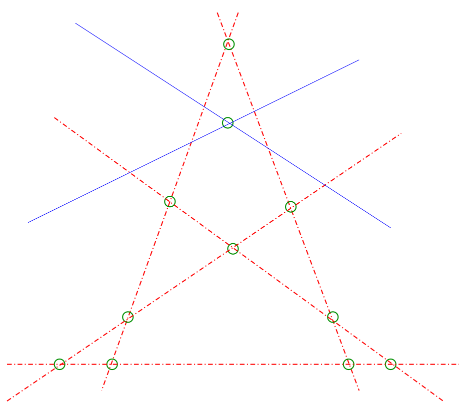

Figure 6-5: A degree 7 polynomial that passes through each of the 11 points twice

degree one polynomials (i.e., lines) and hence has degree 7. Moreover, from the picture, it is "clear" that the curve passes through each of the 11 input points *twice*. Also, the five lines that pass through 4 or more of the points all emerge in the picture (these are the dashed lines). The fact that the relevant lines must be factors of *any* degree 7 fit that passes through each of the 11 points twice, is a consequence of Lemma 6.3 applied to this example (with the choice $l = 7$, $r = 2$ and $t = 4$).

Note that since there are five lines that are solutions, this *proves* that the same approach as in Example 1 (based on fitting a degree 4 bivariate polynomial) will not work for this example. This is because one cannot have five distinct lines all be factors of a degree 4 plane curve. This shows that the multiplicity based approach is necessary for this example.

6.2.6 Results for specific list sizes

Decoding with Constant-sized lists

Suppose we wish to use algorithm *Poly-Reconstruct* to do list-of-L decoding, that is, perform polynomial reconstruction with the guarantee that the list of polynomials that have to be output will be of size at most L. We now indicate the choice of parameters in the algorithm to correct a maximum fraction of errors under this constraint.

Theorem 6.6 (List-of-L decoding) *Consider the polynomial reconstruction problem with inputs n, k, t and pairs $\{(x_i, y_i)\}$, $1 \le i \le n$. Subdivide $(0,1)$ into $L+1$ intervals $(\rho_j, \rho_{j+1}]$, $0 \le j \le L$, where $\rho_j = \frac{j(j+1)}{L(L+1)}$. Let $r = r(k/n)$, $1 \le r \le L+1$ be such that $k/n \in (\rho_{r-1}, \rho_r]$. Then provided*

$$t > \frac{r+1}{2(L+1)} \cdot n + \frac{L}{2r} \cdot k \ , \tag{6.5}$$

the number of solutions to the polynomial reconstruction problem is at most L. Moreover, the algorithm Poly-Reconstruct, when it is run with suitable parameters r, l, finds and outputs a list of size at most L that includes all solution polynomials.

Proof: Since we want to insist that the algorithm *Poly-Reconstruct* outputs at most L solutions, we will also add the condition that y-degree of Q, $\deg_y(Q)$, equals L, in Step 1 of the algorithm. This will imply that Q has at most L roots, and hence the algorithm will output at most L polynomials in Step 2. Now, arguing as in Lemma 6.4, a non-zero polynomial Q with $\deg_y(Q) = L$ and $(1,k)$-wt-deg$(Q) \le l$ as sought in Step 1 will exist provided

$$(L+1)\Big(l + 1 - \frac{Lk}{2}\Big) > n\binom{r+1}{2} \ . \tag{6.6}$$

This condition can be satisfied for any r by choosing

$$l = \left\lfloor \frac{nr(r+1)}{2(L+1)} + \frac{kL}{2} \right\rfloor \ . \tag{6.7}$$

Of course, since $\deg_y(Q) = L$, we must have $l \ge (1,k)$-wt-deg$(Q) \ge Lk$. This gives the condition $\frac{k}{n} \le \frac{r(r+1)}{L(L+1)}$ on r. Let us pick $r = r(k/n)$ to be the smallest integer in the range $1 \le r \le L+1$ such that $k/n \le \frac{r(r+1)}{L(L+1)}$.[4] This is exactly the choice of r specified in the statement of the theorem.

[4]The claim of the theorem actually holds for any value of r in the range $1 \le r \le L$. For values of r greater than or smaller than the one we pick the algorithm will be able to tolerate fewer errors than that achieved by this particular choice of r. Essentially, the plot of the maximum number of errors corrected by the algorithm for each value of r is a straight line, and the best performance is attained by taking the upper envelope of all these straight lines. The value of r we pick is such that the r'th line forms the upper envelope for the given value of k/n.

Now, as in Proposition 6.5, the algorithm *Poly-Reconstruct* succeeds in finding a list of at most L polynomials which satisfy $p(x_i) = y_i$ for at least t values of i, provided $t > l/r$. This condition will be satisfied for our choice of l from Equation (6.7) if

$$t > \frac{n(r+1)}{2(L+1)} + \frac{kL}{2r} .$$

Hence Algorithm *Poly-Reconstruct* performs correct list-of-L decoding as long as the Condition (6.5) stated in the theorem is satisfied. □

Combinatorial implication for list decodability of Reed-Solomon codes: The above result implies that an $[n, k+1, n-k]_q$ Reed-Solomon code is (e, L)-list decodable as long as

$$e < n \cdot \left(1 - \frac{r+1}{2(L+1)} - \frac{L}{2r}\frac{k}{n}\right) , \tag{6.8}$$

(where r is defined to be the least integer in the range $1 \le r \le L+1$ for which $\frac{k}{n} \le \frac{r(r+1)}{L(L+1)}$). Note that in the limit of $L \to \infty$, this converges to $e/n < 1 - \sqrt{k/n}$ (since we will have $r/L \simeq \sqrt{k/n}$), which is the Johnson bound on list decoding radius from Corollary 3.3. But for a finite L, based on the bounds of Corollary 3.3 from Chapter 3, the Johnson radius for list-of-L decoding for MDS codes (like Reed-Solomon codes) is (roughly)

$$e/n = 1 - \sqrt{\frac{k}{n} + \left(1 - \frac{k}{n}\right) \cdot \frac{1}{L}} . \tag{6.9}$$

It can be verified that, at least for some range of parameters (especially for low values of k/n), the bound of Equation (6.8) is stronger than that of Equation (6.9). As a simple illustration, let us consider the case $L = 2$. For $L = 2$, the bound (6.8) implies that for every $k/n < 1/3$, there are at most two codewords in every Hamming ball of radius γn for some $\gamma > \frac{1}{2} \cdot (1 - k/n)$, i.e., greater than half the relative distance (in fact we have $\gamma = \frac{2}{3} - \frac{k}{n}$ in this case). The Johnson bound from Equation (6.9) on the other does not show that list-of-2 decoding is possible beyond half the minimum distance for any value of k/n.

Hence, for the special case of Reed-Solomon codes we are able to prove stronger combinatorial results than the generic Johnson bound on list decodability, and curiously the "improved" bound also comes with an "algorithmic" proof that demonstrates how to recover the list of codewords efficiently.

Decoding with Polynomial-sized lists

We now allow the list size to be a polynomial (actually, quadratic) in n, and pick parameters to squeeze out the maximum error-correction capability of the algorithm *Poly-Reconstruct*. In Step 0 of the algorithm, pick parameters r, l such that

$$r \stackrel{\text{def}}{=} 1 + \left\lfloor \frac{kn + \sqrt{k^2n^2 + 4(t^2 - kn)}}{2(t^2 - kn)} \right\rfloor \tag{6.10}$$

$$l \stackrel{\text{def}}{=} rt - 1 \tag{6.11}$$

Lemma 6.7 *If n, k, t satisfy $t^2 > kn$, then for the choice of r, l made in Equations (6.10) and (6.11) above, the conditions $n\binom{r+1}{2} < \frac{l(l+2)}{2k}$ and $rt > l$ both hold.*

Proof: Since $l \stackrel{\text{def}}{=} rt - 1$, $rt > l$ certainly holds. Using $l = rt - 1$, we now need to satisfy the constraint

$$n\binom{r+1}{2} < \frac{(rt-1)(rt+1)}{2k}$$

which simplifies to $r^2t^2 - 1 > kn(r^2 + r)$ or, equivalently,

$$r^2(t^2 - kn) - knr - 1 > 0.$$

Hence it suffices to pick r to be an integer greater than the larger root of the above quadratic, and therefore picking

$$r = 1 + \left\lfloor \frac{kn + \sqrt{k^2n^2 + 4(t^2 - kn)}}{2(t^2 - kn)} \right\rfloor$$

suffices, and this is exactly the choice made in Equation (6.10). □

Theorem 6.8 *Algorithm* Poly-Reconstruct *on inputs n, k, t and the points $\{(x_i, y_i) : 1 \le i \le n\}$, correctly solves the polynomial reconstruction problem provided $t > \sqrt{kn}$, for the choice of parameters r, l as in Equations (6.10) and (6.11). Moreover, the size of the list the algorithm outputs is at most $O(\sqrt{kn^3})$ (which is in turn $O(n^2)$).*

Proof: The correctness of the algorithm for this choice of parameters follows from Proposition 6.5 and Lemma 6.7. It remains to prove the claim about the number of codewords. By Lemma 6.3, the number M of such codewords is at most the degree $\deg_y(Q)$ of the bivariate polynomial Q in y. Since the $(1, k)$-weighted degree of Q is at most l, $\deg_y(Q) \le \lfloor l/k \rfloor$. Choosing $t = \lfloor\sqrt{kn}\rfloor + 1$ (which corresponds to the largest permissible value of the radius e), we have, by the choice of l, that

$$M = O(l/k) = O(rt/k) = O(knt/k) = O(\sqrt{kn^3}) \ ,$$

as desired. □

6.2.7 Runtime of the Algorithm

We now present a runtime analysis of algorithm *Poly-Reconstruct* and prove that it can be implemented to run efficiently (in time which is a reasonably slowly growing polynomial in n).

As was briefly discussed at the end of Section 6.2.3, the algorithm can definitely be implemented in polynomial time. However, we now give pointers to results which demonstrate much faster implementations of the decoding algorithm. In particular, these imply $\tilde{O}(n^2) \stackrel{\text{def}}{=} O(n^2 \log^{O(1)} n)$ time implementations of Algorithm *Poly-Reconstruct* for list decoding Reed-Solomon codes over poly(n)-sized fields with constant-sized lists (the constant in the big-Oh notation will depend polynomially on the size of the list).

Lemma 6.9 ([NH99b]) *For the algorithm Poly-Reconstruct with parameters r, l over a field $\mathbb{F}$, Step 1 can be implemented in $O(n^2r^5)$ operations over $\mathbb{F}$.*

We refer the reader to [NH99b, Section 6] for a description of the above implementation. Olshevsky and Shokrollahi [OS99] provide a different implementation using $O(n^2r^4 \log_q r \cdot l/k)$ field operations over $\mathbb{F}_q$.

Lemma 6.10 ([RR00]) *The root finding step (Step 2) of Algorithm Poly-Reconstruct over a field of size q can be implemented by a* randomized *algorithm that uses $O((nl + l^2/k \cdot \log q)\log^2(l/k))$ field operations over $\mathbb{F}_q$.*

We refer the reader to [RR00, Section 5] for a description of the implementation of the root-finding step. It is shown there that given a bivariate polynomial $Q(x, y)$ of y-degree at most b, one can find all degree k polynomials $p \in \mathbb{F}_q[x]$ that satisfy $Q(x, p(x)) \equiv 0$ using $O(kb\log^2 b(n + b\log q))$ operations over $\mathbb{F}_q$. For the application to Algorithm *Poly-Reconstruct*, we have $b = \lfloor l/k \rfloor$. Gao and Shokrollahi [GS00] give another algorithm to solve the problem in $O(k^2 b^3)$ time assuming $\log q \le k$. For the interesting setting of parameters, the complexity of Step 1 dominates the runtime of the algorithm. When the field size q is very large, the bound for Step 2 from Lemma 6.10 could dominate.

The interested reader can find the details in the above references. Below, we record the time bounds for the entire polynomial reconstruction algorithm. In the next section, we will record the results for list decoding Reed-Solomon codes along with an explicit runtime bound. Note that the result below simply presents the results of Theorem 6.6 and Theorem 6.8 with explicit runtime bounds that follow from Lemmas 6.9 and 6.10. Actually, slightly better bounds can be stated by carefully adding the bounds for Steps 1 and 2 of the algorithm, but we prefer to state simpler and reasonably tight bounds below.

Proposition 6.11 *The algorithm Poly-Reconstruct over $\mathbb{F}_q$ with inputs n, k, t can be implemented to run in* randomized*:*

(i) $O(n^2 L^5 \log^2 q \log^{O(1)} \log q)$ *time to find a list of size at most L that includes* all *polynomials p that satisfy $p(x_i) = y_i$ for at least t values of i, provided $t > \frac{r+1}{2(L+1)} \cdot n + \frac{L}{2r} \cdot k$. Here the parameter r is defined to be the smallest integer in the range $1 \le r \le L$ for which $\frac{k}{n} \le \frac{r(r+1)}{L(L+1)}$.*

(ii) $O(k^5 n^7 \log^2 q \log^{O(1)} \log q)$ *time to find a list of all polynomials p that satisfy $p(x_i) = y_i$ for at least t values of i, provided $t > \sqrt{kn}$.*

(iii) $O(n^2 \varepsilon^{-5} \log^2 q \log^{O(1)} \log q)$ *time to find a list of size at most $O(\frac{1}{\varepsilon}\sqrt{\frac{n}{k}})$ that includes all polynomials p that satisfy $p(x_i) = y_i$ for at least t values of i, provided $t \ge \sqrt{(1+\varepsilon)kn}$.*

Proof: For Part (i), for list-of-L decoding we have $r \le L + 1$,[5] and $l = O(kL)$. Applying Lemmas 6.9 and 6.10, the complexity of Step 1 is $O(n^2 L^5)$ field operations, and that of Step 2 is $O((nkL + kL^2 \log q)\log^2 L) = O(n^2 L^2 \log^2 L \log q)$ field operations. The overall complexity is therefore certainly at most $O(n^2 L^5 \log q)$ operations over $\mathbb{F}_q$.

For Part (ii), for decoding under the condition $t > \sqrt{kn}$, we have $r = O(kn)$ (from Equation 6.10), and $l = O(tr) = O(kn^2)$. The complexity of Step 1 is $O(n^7 k^5)$ field operations and that of Step 2 is at most $O(kn^4 \log^2 n \log q)$ field operations, for an overall complexity of $O(n^7 k^5 \log q)$ operations over $\mathbb{F}_q$.

For Part (iii), for decoding under the condition $t \ge \sqrt{(1+\varepsilon)kn}$, we have $r = O(1/\varepsilon)$ (again from Equation 6.10), and $l = O(rt) = O(\varepsilon^{-1}\sqrt{kn})$. Since the list size output by the polynomial reconstruction algorithm is at most $\lfloor l/k \rfloor = O(\varepsilon^{-1}\sqrt{n/k})$. The complexity

[5] For large L, we have $r/L \simeq \sqrt{k/n}$ and we can use this to get a slightly better runtime bound. But in later applications of this result we will be mostly interested in the situation when $k/n = \Omega(1)$, and for such a situation the improvement is only by a constant factor.

of Step 1 is $O(n^2\varepsilon^{-5})$ field operations and that of Step 2 is at most $O(n^2\varepsilon^{-2}\log q)$ field operations, for an overall complexity of $O(n^2\varepsilon^{-5}\log q)$ operations over $\mathbb{F}_q$.

Since field operations over $\mathbb{F}_q$ can be implemented in $O(\log q \log^{O(1)} \log q)$ time, the time bounds claimed in the proposition follow. □

A strongly polynomial time deterministic algorithm

Note two important features about the runtime bounds of Proposition 6.11 above: (a) it is randomized, and (b) it does *not* give a strongly polynomial time bound, since the number of field operations itself depends on q. The following result shows that one can have a deterministic strongly polynomial time algorithm algorithm that provides a weaker decoding guarantee. This result stated below is due to Augot and Pecquet [AP00], who prove it by providing a strongly polynomial time implementation of Sudan's earlier polynomial reconstruction algorithm [Sud97a]. We refer the reader to [AP00] for further details about the algorithm.

Proposition 6.12 ([Sud97a, AP00]) *There is a* deterministic, strongly polynomial time *algorithm for the polynomial reconstruction problem over $\mathbb{F}_q$ that solves it for parameters n, k, t using $O(n^2 \log n)$ operations over $\mathbb{F}_q$, provided $t > \sqrt{2kn}$. Moreover, the size of the list output by the algorithm is at most $\sqrt{2n/k}$.*

6.2.8 Main theorems about Reed-Solomon list decoding

So far we stated our results for the polynomial reconstruction problem. Below we state the result specialized to Reed-Solomon decoding.

Theorem 6.13 [Main Result on Reed-Solomon Decoding]

(i) For every integer $L \geq 1$, an $[n, k+1, n-k]_q$ Reed-Solomon code can be list decoded to a radius of e for $e < n - \frac{r+1}{2(L+1)}\cdot n - \frac{L}{2r}\cdot k$ using lists of size L in $O(n^2 L^5 \log^2 q \log^{O(1)} \log q)$ time. Here r is defined to be the smallest integer in the range $1 \leq r \leq L$ for which $\frac{k}{n} \leq \frac{r(r+1)}{L(L+1)}$.

(ii) An $[n, k+1, n-k]_q$ Reed-Solomon code can be list decoded to a radius of e for $e < n - \sqrt{kn}$ in $O(n^{12} \log^2 q \log^{O(1)} \log q)$ time using lists of size $O(n^2)$.

(iii) If $e \leq n - \sqrt{(1+\varepsilon)kn}$, then one can perform list decoding in $O(n^2\varepsilon^{-5}\log^2 q \log^{O(1)} \log q)$ time using lists of size $O(\varepsilon^{-1}\sqrt{n/k})$.

Proof: The statement (i) follows immediately from Proposition 6.11, Part (i), since the Reed-Solomon decoding problem is a special form of the polynomial reconstruction problem (cf. Proposition 6.1). Similarly, the statements (ii) and (iii) follow from Parts (ii) and (iii) of Proposition 6.11. □

"Tightness" of the performance bound

It is interesting to contrast the number of errors corrected by the above result with results in the spirit of Chapter 4 that show combinatorial limits to list decodability by demonstrating a received word with a large number of codewords within a certain Hamming distance from it. We will need such results specialized to the case of Reed-Solomon codes if we wish to understand the maximum number of errors for which a Reed-Solomon code can be list decoded using lists of a certain size. Since we decode up to the Johnson radius, the performance of our algorithm will be the best possible if the Johnson bound is tight for Reed-Solomon codes. However, we do not know this result. (Our results in Chapter 4 proved the tightness of the Johnson bound for *some* "contrived" binary code construction, but not for Reed-Solomon codes.)

Justesen and Hϕholdt [JH01] demonstrate that for certain Reed-Solomon codes, there exist Hamming balls of radius close to $(n-\sqrt{kn})$ with $\Omega(n)$ codewords. This provides strong evidence to the tightness of the Johnson bound (and the number of errors corrected by our algorithm) for list decoding Reed-Solomon codes with constant-sized lists. This result, however, applies only to certain values of the rate. They also obtain results that demonstrate the tightness of the exact bound of Theorem 6.13 for list-of-L decoding for certain values of the rate (namely the values $\frac{j(j+1)}{L(L+1)}$ for $0 \le j \le L$). Ruckenstein and Roth [RR01] extend this result to a wider range of rates — in particular, they demonstrate the tightness of Theorem 6.13 for all values of rate in the ranges $[0, \frac{2}{L(L+1)}]$ and $[\frac{L-1}{L+1}, 1]$. These bounds are of interest in that they hint at a potential limitation to further improvements to the list decoding approach, and they provide good evidence to the tightness of some of our performance bounds. Ruckenstein and Roth [RR01] also demonstrate that for certain setting of parameters, one *cannot* realize configurations exhibiting the tightness of the list-of-L decoding bound of Theorem 6.13, in a Reed-Solomon code. Thus a precise understanding of the combinatorics of list decoding Reed-Solomon codes is still lacking and the problem certainly deserves further study.

6.2.9 Some Further Consequences

We now describe some other easy consequences and extensions of the polynomial reconstruction algorithm of Section 6.2.3. The first three classes of results are just straightforward applications of the polynomial reconstruction algorithm. The fourth result, described in the next section, revisits the curve-fitting algorithm to get a solution to a weighted polynomial reconstruction problem.

Alternant codes

We now describe a family of codes called alternant codes. Alternant codes were first defined and studied by Helgert [Hel74]. A good discussion of alternant codes appears in [MS81, Chap. 12]. Alternant codes are a very broad class of codes which among other things include Reed-Solomon codes and BCH codes. BCH codes (short for Bose-Chaudhuri-Hocquenghem codes) are an extremely useful and well-studied family of codes, first introduced by [BR60, Hoc59] (see [MS81, Chap. 9] for a thorough discussion of BCH codes). They possess several of the nice algebraic properties of Reed-Solomon codes, but in addition have the advantage that codes with large blocklengths can be defined even over a fixed, small alphabet (unlike Reed-Solomon codes which require an alphabet size at least as large as the blocklength).

Definition 6.3 (Alternant Codes ([MS81], §12.2)) *For positive integers m, k_0, n, prime power q, the field $\mathbb{F} = \mathrm{GF}(q^m)$, a vector $\vec{\alpha}$ of distinct elements $\alpha_1, \ldots, \alpha_n \in \mathrm{GF}(q^m)$, and a vector $\vec{v}$ of nonzero elements $v_1, \ldots, v_n \in \mathrm{GF}(q^m)$, the q-ary alternant code $\mathcal{A}_{q,n,k_0,\vec{\alpha},\vec{v}}$ comprises of those codewords of the Generalized Reed-Solomon code $\mathrm{GRS}_{\mathbb{F},n,k_0,\vec{\alpha},\vec{v}}$ that have all components from $\mathrm{GF}(q)$.*

Since the Generalized Reed-Solomon code has distance exactly $n - k_0 + 1$, it follows that the respective alternant code, being a subcode of the Generalized Reed-Solomon code, has distance at least $n - k_0 + 1$. We term this the *designed distance* $d' = n - k_0 + 1$ of the alternant code. The actual rate and distance of the code are harder to determine. The rate lies somewhere between $n - m(n - k_0)$ and k_0, and the distance d lies between d' and md'.

The decoding algorithm of the previous section can be used to decode alternant codes as well. Given a received word $\langle r_1, \ldots, r_n \rangle \in \mathrm{GF}(q)^n$, we use as input to the polynomial reconstruction problem the pairs $\{(x_i, y_i)\}_{i=1}^n$, where $x_i = \alpha_i$ and $y_i = r_i/v_i$ are elements of $\mathrm{GF}(q^m)$ (here we are viewing each element $r_i \in \mathrm{GF}(q)$ as an element of $\mathrm{GF}(q^m)$). The list of polynomials output includes all possible codewords from the alternant code. Thus the decoding algorithm for the earlier section is really a decoding algorithm for alternant codes as well, with the caveat that its performance can only be compared with the designed distance, rather than the actual distance. The following theorem summarizes the scope of the decoding algorithm.

Theorem 6.14 *Let $\mathcal{A}$ be an $[n, k+1, d]_q$ alternant code with designed distance d' (and thus satisfying $\frac{d}{m} \le d' \le d$). Then there exists a polynomial time list decoding algorithm for $\mathcal{A}$ decoding up to $e < n - \sqrt{n(n-d')}$ errors.*

Discussion: We note that decoding algorithms for alternant codes given in classical texts like [MS81, Ber68] seem to correct only up to half the designed distance. Hence the above theorem improves upon those algorithms for every value of the designed distance. Since alternant codes include BCH codes as a special case, in particular we also have an algorithm to decode BCH codes up to the above radius. However, BCH codes can be defined over small alphabets, even over GF(2), and then the above bound $(n - \sqrt{n(n-d')})$ which does account for the alphabet size is quite far from the Johnson radius (which for binary codes is $\frac{n - \sqrt{n(n-2d')}}{2}$, from Theorem 3.2 of Chapter 3). But as noted by [KV00] it is possible to use the soft decoding algorithm that we will shortly discuss in Section 6.2.10 to decode q-ary BCH codes up to the q-ary Johnson radius. A similar situation arises with algebraic-geometric codes, and in Section 6.3.8 we will discuss how to use soft decoding to decode them up to the q-ary Johnson radius. The same argument will also apply to BCH codes, by using the soft decoding algorithm from Theorem 6.21 instead of the soft decoding algorithm for decoding AG-codes from Theorem 6.35.

Decoding with uncertain receptions

Consider the situation when, instead of receiving a single word $\mathbf{y} = \langle y_1, y_2, \ldots, y_n \rangle$, for each $i \in [n]$, we receive a list of ℓ possibilities $y_{i1}, y_{i2}, \ldots, y_{i\ell}$ such that one of them is the correct symbol (but we do not know which one). Once again, as in normal list decoding, we wish to find out all possible codewords which could have been possibly transmitted, except that now the guarantee given to us is not in terms of the number of errors effected, but in terms of the maximum number of uncertain possibilities at each position of the received word. Let us call this problem *decoding from uncertain receptions*. In this situation, we have the following result.

Theorem 6.15 *List decoding from uncertain receptions on a $[n, k+1, d = n-k]_q$ Reed-Solomon code can be done in polynomial time provided the number of "uncertain possibilities" l at each position $i \in [n]$ is (strictly) less than n/k.*

Proof: Apply the solution to the polynomial reconstruction problem with input being the set of pairs $\{(x_i, y_{ij}) : 1 \le i \le n,\ 1 \le j \le \ell\}$, the goal being to find all degree k polynomials p such that $p(x_i) \in \{y_{i1}, \ldots, y_{i\ell}\}$ for every i, $1 \le i \le n$. Applying Proposition 6.11, Part (ii), with the total number of pairs $N = n\ell$ and $t = n$, we get that we can solve this problem in polynomial time provided $n > \sqrt{kn\ell}$, or if $\ell < n/k$. $\square$

We can also consider the situation when given lists L_i of size ℓ at each position of the code, we wish to find all Reed-Solomon codewords that agree with an element of the list L_i for at least a fraction α of the positions i.[6] The polynomial reconstruction problem still captures this problem, and below we state a version of the result that follows from Part (iii) of Proposition 6.11.

Theorem 6.16 *Let C be an $[n, k+1, n-k]_q$ Reed-Solomon code. Suppose we are given lists $L_i \subseteq \mathbb{F}_q$ such that the average list size equals ℓ, i.e., $\sum_i |L_i| = \ell n$. Then, there are at most $O(\gamma^{-1}\sqrt{n\ell/k})$ codewords of C which agree with an element of L_i for at least αn values of i, provided $\alpha > \sqrt{(1+\gamma)k\ell/n}$. Moreover, the list of all such codewords can be found in* randomized *$O(n^2\ell^2\gamma^{-5}\log^2 q \log^{O(1)} \log q)$ time.*

We can obtain a strongly polynomial time version of the above result using the result of Proposition 6.12. This is very useful for certain later applications where we consider Reed-Solomon codes over very large alphabets (compared to the blocklength).

Theorem 6.17 *Let C be an $[n, k+1, n-k]_q$ Reed-Solomon code of rate r. Suppose we are given lists $L_i \subseteq \mathbb{F}_q$ such that the average list size equals ℓ, i.e., $\sum_i |L_i| = \ell n$. Then, there are at most $\sqrt{2n\ell/k}$ codewords of C which agree with an element of L_i for at least αn values of i, provided $\alpha > \sqrt{2k\ell/n}$. Moreover, the list of all such codewords can be found* deterministically *using $O(n^2\ell^2 r^{-O(1)}\log(n\ell))$ operations over $\mathbb{F}_q$, or in $O(n^2\ell^2 r^{-O(1)}\log(n\ell)\log q \log^{O(1)}\log q)$ time. (For constant rate, the runtime equals $O((n\ell)^2\log(n\ell)\log q\log^{O(1)}\log q)$.)*

The above versions of the result and the soft decoding algorithm from Proposition 6.21 to be discussed shortly are in fact the main forms of our decoding results for Reed-Solomon codes that will be used repeatedly in later chapters.

Errors and Erasures decoding

The algorithm of Section 6.2.3 is also capable of dealing with other notions of corruption of information. A much weaker notion of corruption than an error in data transmission is that of an "erasure". Here a transmitted symbol is either simply "lost" or received in obviously corrupted shape (so we might as well declare it erased). We now note that the decoding algorithm of Section 6.2.3 naturally handles the case when there are both errors and erasures. Suppose n symbols were transmitted and $n' \le n$ were received and $s = n - n'$ symbols got erased. (We stress that the problem definition specifies that the receiver knows which symbols are erased.) This problem just reduces to a polynomial reconstruction problem on n' points. An application of Proposition 6.11 yields that e errors can be corrected provided $e < n' - \sqrt{n'k}$. Thus we get:

[6] This model of decoding will be formally defined and called *list recovering* in Chapter 9, where we will study combinatorial aspects of codes with good list recoverability, and will also make use of the list recovering property of Reed-Solomon codes.

Theorem 6.18 *The list decoding problem for $[n, k+1, n-k]_q$ Reed-Solomon codes allowing for e errors and s erasures can be solved in polynomial time, provided $e+s < n-\sqrt{(n-s)k}$.*

The classical results of this nature show that one can solve the decoding problem if $2e + s < n - k$. To compare the two results we restate them. The classical result can be rephrased as

$$n - (s + e) > \frac{n - s + k}{2},$$

while our result requires that

$$n - (s + e) > \sqrt{(n - s)k}.$$

By the Arithmetic mean vs. Geometric mean inequality it is clear that the second condition holds whenever the first one holds.

6.2.10 Weighted polynomial reconstruction and soft decoding of RS codes

Soft decoding: The context

Moving on to a more general situation that includes both the errors-and-erasures model and the model of decoding with uncertain receptions from the previous sections, we now consider the case where the received symbol looks like a "combination" of several (or even, all) possible field elements, with varying "degrees of confidence". This model is referred to in the literature as *soft-decision decoding* (or simply, *soft decoding*), as opposed to "hard decoding" with a *fixed* received word which has been the object of study so far. Under soft decoding the received "word" is really an $n \times q$-matrix $R = \{r_{ij}\}$ of non-negative rational numbers, with an entry corresponding to each codeword position and each symbol of the alphabet (we take the alphabet to be $[q]$). The entry r_{ij} indicates the *weight* with which the i'th transmitted symbol may be taken to be j.[7] Exactly how to set these weights can itself be a non-trivial task depending on the specific code and channel under question. We do not address this issue here, and just provide a "back-end" soft decoder for Reed-Solomon codes that makes good use of the weights *given* the weight matrix R as input. Koetter and Vardy [KV00] address the question of how to assign weights in order to get good performance from soft decoding for several channel models. We also point to their work for further discussion and pointers relating to soft decoding.

The soft decoding model is quite general and it captures all the previously discussed noise models. The case of decoding from uncertain receptions is captured by the case when $r_{ij} = 1$ for each field element in the list of "uncertain possibilities", and $r_{ij} = 0$ for each element not on this list. Similarly, the i'th symbol being declared an erasure is captured by setting $r_{ij} = 0$ for every field element j. The "errors only" (or hard-decision) case with received word $\mathbf{y} = \langle y_1, \ldots, y_n \rangle$ is captured by $r_{i,y_i} = 1$ and $r_{ij} = 0$ for $j \in [q] \setminus \{y_i\}$.

Given an input reliability matrix R, the goal of soft decoding for Reed-Solomon codes would be to find all polynomials p that are "close" to this R matrix. It is not totally clear what the best measure of "closeness" is. One natural measure is to measure the closeness of a polynomial p to R by the quantity $S(p, R) = \sum_{i=1}^{n} r_{i,p(x_i)}$ (here $x_1, x_2, \ldots, x_n$ are n

[7]In its most typical use, the soft information can be used to model situations where the "received word" is itself returned by an earlier decoding stage — for example that performed by a source decoder or an inner decoder in a concatenation scheme. In such a case, the decoder can set the weights r_{ij} to somehow quantify the confidence it has in its "vote" for the i'th symbol being equal to j.

distinct field elements that define the Reed-Solomon encoding of a polynomial p). The holy grail in this setting is maximum likelihood soft-decision decoding, where the goal is to find the polynomial p that maximizes $S(p,R)$. This turns out to be a prohibitively difficult algorithmic task, so one studies the *bounded distance soft decoding* problem. This problem reduces to the problem of finding a list of all polynomials p such that $S(p,R)$ is at least a certain threshold.

In what follows, we provide a polynomial time bounded distance soft decoding algorithm for Reed-Solomon codes. For a matrix R as input, the algorithm finds all degree k polynomials p such that $S(p,R)$ is at least $\sqrt{k}$ times the L_2 norm of the R matrix. We obtain our soft decoding algorithm via a solution to a weighted generalization of the polynomial reconstruction problem, which we define next.

Weighted polynomial reconstruction

We first formally define the problem.

Problem 3 (Weighted polynomial reconstruction)
INPUT: *N distinct pairs $\{(x_1,y_1),\ldots,(x_N,y_N)\}$, where each $x_i,y_i \in \mathbb{F}$ for some field $\mathbb{F}$;*
N non-negative weights $w_1,\ldots,w_N$;
degree parameter k and agreement parameter W.
OUTPUT: *All polynomials p of degree at most k which satisfy*

$$\sum_{i:p(x_i)=y_i} w_i \geq W \ .$$

Note: There is **no** requirement of distinctness on the various x_i's in the above specification. This will be crucial to the application to soft decoding algorithms for Reed-Solomon codes.

We now discuss how the algorithm *Poly-Reconstruct* can be modified to solve the weighted polynomial reconstruction problem for the case when the weights are non-negative integers. The basic idea is to use multiplicities at the various points (x_i,y_i) in proportion to their weights, in the interpolation step (Step 1 of the algorithm). Specifically, in Step 1, we would find a polynomial Q which has a singularity of order $w_i\rho$ at the point (x_i,y_i), for a suitably large integer parameter ρ. Thus we would now have $\sum_{i=1}^{n}\binom{\rho w_i+1}{2}$ constraints. If a polynomial p passes through the points (x_i,y_i) for every $i \in S$ for some $S \subseteq [n]$, then $y-p(x)$ will appear as a factor of $Q(x,y)$ provided $\sum_{i\in S}\rho w_i$ is greater than $(1,k)$-wt-deg(Q). Also $(1,k)$-wt-deg(Q) must be picked large enough such that Q has more coefficients than the number of linear constraints it must satisfy (so that a non-zero Q with the required properties will be guaranteed to exist). Optimizing over the weighted degree of Q yields the following theorem – the proof is similar to Theorem 6.8.

Theorem 6.19 *If the weights w_i are* non-negative integers, *the weighted polynomial reconstruction problem with parameters k,W over $\mathbb{F}_q$ can be solved deterministically in time polynomial in the sum of w_i's and q, provided $W > \sqrt{k\sum_{i=1}^{N} w_i^2}$.*

We now remove the requirement of integer weights and the pseudo-polynomial dependence of the runtime on the w_i's (ideally we would like the algorithm to run in time polynomial in the logarithm of the w_i's). Below we note that with an ϵ degradation in performance, the algorithm can be implemented to run in poly$(N,1/\epsilon)$ time, even when the weights are arbitrary rational numbers.

Lemma 6.20 *For any tolerance parameter $\epsilon > 0$, the weighted polynomial reconstruction problem for N pairs $\{(x_i, y_i) \in \mathbb{F}_q^2\}$ with associated non-negative rational weights w_i, degree parameter k and agreement parameter W, can be solved deterministically in time polynomial in N, q and $1/\epsilon$, provided*

$$W > \sqrt{k \sum_{i=1}^{N} w_i^2} + \epsilon w_{\max}.$$

Proof: Assume without loss of generality that $w_1 \le w_2 \le \cdots \le w_N$. Pick any large integer $L \ge \frac{N}{\epsilon}$, and form the integer weights $w'_i = \lfloor L w_i / w_N \rfloor$. Since $w_i \le w_N$ for all i, the weights w'_i are all at most L. Therefore Theorem 6.19 implies that one can find, in $\mathrm{poly}(N, L)$ time, a list of all polynomials p of degree less at most k that satisfy

$$\sum_{i:p(x_i)=y_i} w'_i > \sqrt{k \sum_{i=1}^{N} {w'}_i^2}.$$

But since $Lw_i/w_N \ge w'_i > Lw_i/w_N - 1$, this implies that in $\mathrm{poly}(N, L, q) = \mathrm{poly}(N, q, 1/\epsilon)$ time, we can find all polynomials p of degree at most k that satisfy the condition

$$\begin{aligned}
& \sum_{i:p(x_i)=y_i} \left(\frac{Lw_i}{w_N} - 1 \right) && \ge && \sqrt{k \sum_{i=1}^{N} \left(\frac{Lw_i}{w_N} \right)^2} \\
\Longleftarrow \quad & \sum_{i:p(x_i)=y_i} w_i && \ge && \sqrt{k \sum_{i=1}^{N} w_i^2} + \frac{N w_N}{L} \\
\Longleftarrow \quad & \sum_{i:p(x_i)=y_i} w_i && \ge && \sqrt{k \sum_{i=1}^{N} w_i^2} + \epsilon w_N
\end{aligned}$$

(the last step follows since $L \ge N/\epsilon$). □

Reed-Solomon soft decoding: Main result

We also record the following consequence for soft decoding of Reed-Solomon codes. Its proof follows immediately from the above lemma. We will appeal to this result several times when we discuss decoding algorithms for concatenated codes in Chapter 8 of the thesis.

Theorem 6.21 (Soft list decoding of Reed-Solomon codes) *Consider an $[n, k+1, n-k]_q$ Reed-Solomon code with messages being polynomials r over $\mathbb{F}_q$ of degree at most k. Let the encoding function be $r \mapsto \langle r(x_1), r(x_2), \ldots, r(x_n) \rangle$ where $x_1, \ldots, x_n$ are distinct elements of $\mathbb{F}_q$. Let $\epsilon > 0$ be an arbitrary constant. For each $i \in [n]$ and $\alpha \in \mathbb{F}_q$, let $w_{i,\alpha}$ be a non-negative rational number. Then, there exists a deterministic algorithm with runtime $\mathrm{poly}(n, q, 1/\epsilon)$ that, when given as input the weights $w_{i,\alpha}$ for $i \in [n]$ and $\alpha \in \mathbb{F}_q$, finds a list of* all *polynomials $p \in \mathbb{F}_q[x]$ of degree at most k that satisfy*

$$\sum_{i=1}^{n} w_{i,p(x_i)} \ge \sqrt{k \sum_{i=1}^{n} \sum_{\alpha \in \mathbb{F}_q} w_{i,\alpha}^2} + \epsilon \max_{i,\alpha} w_{i,\alpha} \ . \tag{6.12}$$

Finally, we would like to point out that, once again, the error-correction performance of the above result approaches what is indicated to be possible by the combinatorial bounds on list decodability, specifically the "weighted Johnson bound" from Corollary 3.7 of Chapter 3.

Significance of weighted polynomial reconstruction

The weighted polynomial reconstruction problem is at the heart of soft decoding algorithms for Reed-Solomon codes (cf. [KV00]). It also plays a crucial role in decoding certain concatenated codes, where the weights for Reed-Solomon decoding are passed by the inner decoder [GS00, Nie00, GHSZ00, Gur01b]. This will be discussed in detail in Chapter 8 on concatenated codes. Thus, the weighted polynomial reconstruction is a very useful subroutine that has found several applications to list decoding, and we view it as one of the key contributions of this thesis.

Also, the basic algebraic technique behind weighted polynomial reconstruction extends to more general codes than Reed-Solomon codes. In the next section, we will present a list decoding algorithm for algebraic-geometric codes, which then also generalizes to a weighted version (Section 6.3.7). In the next chapter, we will present a soft list decoding algorithm for an even broader class of codes called "ideal-based" codes.

6.3 Algebraic-Geometric Codes

We now describe the extension of our algorithm to the case of algebraic-geometric codes. Our extension shows that the algebra of the previous section extends to the case of algebraic function fields, yielding an approach to the list decoding problem for algebraic-geometric codes. In particular it reduces the decoding problem to some basis computations in an algebraic function field and to a root-finding problem over the algebraic function field. However neither of these tasks is known to be solvable efficiently given only the generator matrix of the algebraic-geometric code (or some such standard "minimal" representation of a linear code). But we will show that by precomputing a polynomial amount of additional information about the linear code and the underlying algebraic structures, one can solve both parts efficiently.

6.3.1 Overview

Algebraic-geometric codes (henceforth AG-codes) are defined by evaluations of "regular" functions at a set of points on a "nice" algebraic curve. These were first defined by Goppa [Gop81] in a seminal work, and are hence sometimes also referred to as geometric Goppa codes. Their properties are proved using some deep facts from the theory of algebraic function fields. Before we describe the generalization of the Reed-Solomon decoding algorithm to AG-codes, in Section 6.3.2 we first develop the necessary definitions and preliminaries on algebraic function fields, and formally define algebraic-geometric codes.

We then present our algorithm for list decoding modulo some algorithmic assumptions about the underlying structures. Under these assumptions, our algorithm yields an algorithm for list decoding which corrects up to $e < n - \sqrt{n(n-d^*)}$ errors in a code of blocklength n and designed distance d^*. (The earlier best result for list decoding AG-codes, due to [SW99], could correct up to about $n - \sqrt{2n(n-d^*)}$ errors.)

Obtaining an efficient implementation of our decoding algorithm raises some fundamental questions about how elements of an algebraic function field are represented and manipulated. We next discuss these issues in detail and demonstrate a representation of AG-codes under which the list decoding algorithm can be implemented to run in polynomial time. This turns out to be a non-standard representation (i.e., we do not how to implement the algorithm in polynomial time given only the generator matrix or some such "standard" representation of the code). However, the necessary representation of the code is succinct and is of size polynomial in the blocklength of the code.

6.3.2 Algebraic-geometric codes: Preliminaries

We now discuss the main notions associated with the theory of algebraic function fields that will be necessary for defining and studying algebraic-geometric codes. The interested reader may find further details in [Sti93, GS95a]. In the presentation below, we will assume familiarity with the basic notions of field extensions, which can be found in any standard algebra text, eg., [Art91]. We note that the definition of AG-codes presented here is in line with that of the standard texts (in particular, we closely follow the presentation in Stichtenoth's book [Sti93]) — a different presentation which is somewhat less heavy on algebraic terminology appears in the paper by the author and Sudan [GS99] where the decoding algorithm discussed here first appeared.

An extension field K of a field k, denoted K/k, is an *algebraic function field* (or simply, *function field*) over k if the following conditions are satisfied: (i) There is an element $x \in K$ that is transcendental over k such that K is a finite extension of $k(x)$, and (ii) k is algebraically closed in K, that is, the only elements in K that are algebraic over k are those in k.

For our applications to AG-codes, we will be interested in the case when k is a finite field, i.e., $k = \mathbb{F}_q$ for some prime power q. A function field $K/\mathbb{F}_q$ can be obtained as $K = \mathbb{F}_q(X)[y_1, y_2, \ldots, y_m]$ where each y_i satisfies some polynomial equation over $\mathbb{F}_q(X)[y_1, \ldots, y_{i-1}]$. For the rest of this section, K will denote the function field in question.

Places and Valuations: A function field $K/\mathbb{F}_q$ has a set of *places* $\mathbb{P}_K$ and the associated set of *valuations*, given by a valuation map $v : \mathbb{P}_K \times K \to \mathbb{Z} \cup \{\infty\}$. The exact definition of these notions can be found, for instance, in [Sti93]; we only abstract some properties relevant to us below.

Intuitively the places correspond to "points" on the algebraic curve associated with the function field K, and the valuation map tells us how many poles or zeroes a function in K has at a specific place in $\mathbb{P}_K$. It has the property that for any $f \in K$, there are only finitely many places $P \in \mathbb{P}_K$ such that $v(P, f) \neq 0$. As is normal practice, for each $P \in \mathbb{P}_K$, we denote by $v_P : K \to \mathbb{Z} \cup \{\infty\}$, the map $v_P(\cdot) = v(P, \cdot)$ which tells how many zeroes or poles a given function has at P (with the convention $v_P(0) = \infty$ for any place P). If $v_P(x) < 0$, we say x has a pole at P, and $-v_P(x)$ is called the *pole order* of x at P. Similarly, if $v_P(x) > 0$, we say that x has a zero at P, and in such a case $v_P(x)$ is the *zero order* of x at P. The valuation v_P at any place satisfies the following properties:

(a) $v_P(a) = \infty$ iff $a = 0$ and $v_P(a) = 0$ for all $a \in \mathbb{F}_q \setminus \{0\}$.

(b) $v_P(ab) = v_P(a) + v_P(b)$ for all $a, b \in K \setminus \{0\}$.

(c) $v_P(a + b) \geq \min\{v_P(a), v_P(b)\}$ for all $a, b \in K$.

For those familiar with some commutative algebra terminology, we just recap how places are formally defined from their corresponding valuations. For every valuation v_P of K, the ring of regular functions at P, denoted $\mathcal{O}_P$, is defined to be

$$\mathcal{O}_P = \{x \in K : v_P(x) \geq 0\} .$$

This ring is a "discrete valuation ring" and in particular is a local ring with a unique maximal ideal. This unique maximal ideal of $\mathcal{O}_P$ is *defined* to be the "place" P associated with v_P, and is given by

$$P = \{x \in K : v_P(x) > 0\} .$$

Intuitively, $\mathcal{O}_P$ is the ring of functions in K that do not have any poles at a certain "point" on the curve; the place P corresponds to this point, and is algebraically defined as the ideal of functions in K that vanish at that "point".

Degree of a place: Associated with every place is a *degree* abstracted via the map deg : $\mathbb{P}_K \to \mathbb{Z}^+$. The degree, $\deg(P)$, of any place P is a positive integer and intuitively means the following: when we pick a function $f \in K$ which has no poles at P and "evaluate" it at P, we get a value in the field $\mathbb{F}_{q^{\deg(P)}}$. Places of degree one correspond to *rational points* on the curve. More formally, the notion of degree means the following: for every place P, the quotient ring $\mathcal{O}_P/P$ is a finite field of size $q^{\deg(P)}$.

Evaluations of functions at places: We can abstract the notion of *evaluation* of elements of the function field at the places by a map $\mathsf{eval} : K \times \mathbb{P}_K \to \bar{\mathbb{F}}_q \cup \{\infty\}$ (here $\bar{\mathbb{F}}_q = \bigcup_{i \geq 1} \mathbb{F}_{q^i}$ is the algebraic closure of $\mathbb{F}_q$). This map has the following properties:

(i) For every $P \in \mathbb{P}_K$ and $f \in K$, $\mathsf{eval}(f, P) = \infty$ iff $v_P(f) < 0$, and $\mathsf{eval}(f, P) = 0$ iff $v_P(f) > 0$.

(ii) If $f \in K$, $P \in \mathbb{P}_K$ and $v_P(f) \geq 0$, then $\mathsf{eval}(f, P) \in \mathbb{F}_{q^{\deg(P)}}$.

(iii) The map eval respects field operations; in other words, if $v_P(f_1) \geq 0$ and $v_P(f_2) \geq 0$, then $\mathsf{eval}(f_1 + f_2, P) = \mathsf{eval}(f_1, P) + \mathsf{eval}(f_2, P)$, and $\mathsf{eval}(f_1 * f_2, P) = \mathsf{eval}(f_1, P) * \mathsf{eval}(f_2, P)$ (where we have used $(+, *)$ to denote the addition and multiplication operations in both K and $\bar{\mathbb{F}}_q$).

Divisors: The *divisor group* $\mathcal{D}_K$ of the function field K is a free abelian group on $\mathbb{P}_K$. An element D of $\mathcal{D}_K$ is thus represented by the formal sum $\sum_{P \in \mathbb{P}_K} a_P P$ where each $a_P \in \mathbb{Z}$, and $a_P = 0$ for all but finitely many P. We say $D \succeq 0$ if $a_P \geq 0$ for all $P \in \mathbb{P}_K$. The support of a divisor D, denoted $\mathrm{supp}(D)$, is the (finite) set $\{P \in \mathbb{P}_K : a_P \neq 0\}$. The degree map extends naturally to the divisor group $\mathcal{D}_K$ and deg : $\mathcal{D}_K \to \mathbb{Z}$ is defined as $\deg(\sum_P a_P P) = \sum_P a_P \deg(P)$.

For every $f \in K \setminus \{0\}$, there is an associated divisor, called the principal divisor and denoted (f), which is defined by $(f) = \sum_P v_P(f) P$. The following result states that degree of any principal divisor equals 0. It is a well-known result and just states that every non-zero function in the function field has an equal number of zeroes and poles.

Proposition 6.22 *For any function field $K/\mathbb{F}_q$ and any $f \in K \setminus \{0\}$, $\deg((f)) = 0$.*

The Riemann-Roch Theorem: For every divisor $D \in \mathcal{D}_K$, one can define the linear space of functions $\mathcal{L}(D)$ as

$$\mathcal{L}(D) = \{g \in K : (g) + D \succeq 0\}.$$

For example for a divisor $D = aQ - bP$ where $P, Q \in \mathbb{P}_K$ and $a, b > 0$, $\mathcal{L}(D)$ is the space of all functions that have at least b zeroes at P and at most a poles at Q. It is known that for any divisor $D \succeq 0$, $\mathcal{L}(D)$ is a finite-dimensional vector space over $\mathbb{F}_q$ and

$\dim(\mathcal{L}(D)) \le 1 + \deg(D)$ (see [Sti93] for a proof). A lower bound on $\dim(\mathcal{L}(D))$ is given by the celebrated Riemann-Roch theorem for function fields, which is stated below. The theorem statement also introduces the "genus" of a function field $K/\mathbb{F}_q$, which in some sense measures the "complexity" of the underlying algebraic curve. (A genus equal to zero corresponds to the simplest case when $K = \mathbb{F}_q(X)$ is the field of all rational functions in one variable.)

Theorem 6.23 [Riemann-Roch]: *Let $K/\mathbb{F}_q$ be any function field. There is a non-negative integer g, called the* genus *of $K/\mathbb{F}_q$, such that*

(a) For any divisor $D \in \mathcal{D}_K$, $\dim(\mathcal{L}(D)) \ge \deg(D) - g + 1$.

(b) There is an integer c, depending only on $K/\mathbb{F}_q$, such that $\dim(\mathcal{L}(D)) = \deg(D) - g + 1$ whenever $\deg(D) \ge c$. Furthermore, $c \le 2g - 1$.

Algebraic-geometric codes: We are now ready to define the notion of an AG-code (also known as *geometric Goppa code*). Let $K/\mathbb{F}_q$ be an algebraic function field of genus g, let $P_0, P_1, P_2, \ldots, P_n$ be *distinct* places of degree one in $\mathbb{P}_K$, and let $G = P_1 + P_2 + \cdots + P_n$ and $D = \alpha P_0$ be divisors of $K/\mathbb{F}_q$ (note that $\mathrm{supp}(G) \cap \mathrm{supp}(D) = \emptyset$).

The algebraic-geometric code $\mathcal{C}_{\mathcal{L}}(G, D) = \mathcal{C}_{\mathcal{L}}(G, \alpha, P_0)$ is defined by

$$\mathcal{C}_{\mathcal{L}}(G, \alpha, P_0) := \{(\mathsf{eval}(f, P_1), \ldots, \mathsf{eval}(f, P_n)) : f \in \mathcal{L}(\alpha P_0)\} \subseteq \mathbb{F}_q^n \ .$$

(Note that $\mathsf{eval}(f, P_i) \in \mathbb{F}_q$ since $v_{P_i}(f) \ge 0$ and $\deg(P_i) = 1$.) It is clear that the defined space is a linear space, since $\mathcal{L}(\alpha P_0)$ is an $\mathbb{F}_q$-linear vector space. The following Proposition follows from the Riemann-Roch theorem and quantifies the parameters of these codes. [8]

Proposition 6.24 *Let $K/\mathbb{F}_q$ be a function field of genus g, and let α, n be positive integers with $\alpha < n$. Let $P_0, P_1, P_2, \ldots, P_n$ be distinct places of degree one in $\mathbb{P}_K$, and let G be the divisor $G = P_1 + P_2 + \ldots + P_n$. Then $\mathcal{C}_{\mathcal{L}}(G, \alpha, P_0)$ is an $[n, k, d]_q$ code with $k = \dim(\mathcal{L}(\alpha P_0)) \ge \alpha - g + 1$ and $d \ge d^* \stackrel{\mathrm{def}}{=} n - \alpha$. The quantity $d^* = n - \alpha$ is called the* designed distance *of the code. Moreover, if $\alpha \ge 2g - 1$, then $k = \alpha - g + 1$.*

Proof: The claims about the dimension follow from the Riemann-Roch theorem. For the distance property, let $f_1 \ne f_2 \in L(\alpha P_0)$ be two distinct messages of $\mathcal{C}_{\mathcal{L}}(G, \alpha, P_0)$. Then $f_1 - f_2 \in L(\alpha P_0)$ as well, and hence $f_1 - f_2$ has at most $\alpha \deg(P_0) = \alpha$ poles in all. By Proposition 6.22, $f_1 - f_2$ has at most α zeroes, and hence $\mathsf{eval}(f_1 - f_2, P_i) = 0$ for at most α values of i, $1 \le i \le n$. Therefore, the encodings of f_1 and f_2 agree on at most α places among $P_1, P_2, \ldots, P_n$, which proves that the distance of the code is at least $n - \alpha$. □

Significance of AG-codes: Codes constructed as above and achieving $d/n, k/n > 0$ (in the limit of large n) are known for constant alphabet size q. In fact, such codes achieving bounds better than those known by probabilistic constructions are known for $q \ge 49$ [TVZ82]. Such a situation, where an explicit construction is better than the best probabilistic construction,

[8] The AG-codes defined here are the so called "one-point divisor codes" since the space of functions are allowed to have poles at only one place. While one can also study AG-codes defined using more general divisors (i.e., allow poles at multiple places), one-point divisor codes are the most widely studied AG-codes, and there is no clear quantitative advantage of using more general divisors in defining the code. Therefore, in this thesis we will focus only on one-point divisor codes. An explicit generalization of the decoding algorithm presented in this thesis to the case of AG-codes based on general divisors appears, for instance, in [Pec01].

is quite rare in combinatorics. This is one of the primary reasons for the importance of and enormous interest in algebraic-geometric codes. Moreover, AG-codes have a very rich algebraic structure which can be exploited to design efficient decoding algorithms, as we do in this section. Recently, Elkies [Elk01] defined a more general algebraic family of codes than the AG-codes discussed here. His codes are non-linear and are based on some deep algebraic properties of certain modular curves. For certain setting of parameters his codes provide an asymptotic improvement over what is possible using "conventional" AG-codes. Thus the underlying idea of defining codes by evaluations of "nice" functions on a suitably picked set of points on a "nice" algebraic curve or variety is a very powerful one. Our decoding algorithm gives a powerful and quite general method of dealing with such algebraic codes.

6.3.3 List decoding algorithm for algebraic-geometric codes

We now describe the extension of our Reed-Solomon list decoding algorithm to the case of algebraic-geometric codes. We begin with a formal description of the problem – this is the generalization of the polynomial reconstruction problem discussed in the decoding of Reed-Solomon codes to the case of algebraic function fields.

Problem 4 (Function Reconstruction)
INPUT: *Integers n, α, t; n distinct pairs $\{(P_i, y_i)\}_{i=1}^{n}$ where each P_i is a place of degree one of the function field $K/\mathbb{F}_q$, and each $y_i \in \mathbb{F}_q$; and a place P_0 of degree one with $P_0 \notin \{P_1, \ldots, P_n\}$.*
OUTPUT: *All functions h in $\mathcal{L}(\alpha P_0)$ that satisfy $\mathsf{eval}(h, P_i) = y_i$ for at least t values of i, $1 \le i \le n$.*

As before, it is easily seen the list decoding problem for the AG-code $\mathcal{C}_{\mathcal{L}}(G, \alpha, P_0)$ from $(n-t)$ errors (with divisor $G = P_1 + P_2 + \ldots + P_n$) reduces to the above reconstruction problem. While in the AG-codes case the places P_i are distinct, note that this is not required in the above specification, and indeed as with the case of polynomial reconstruction, we will solve the above problem without assuming that the P_i's are distinct.

Solution Idea

As with the Reed-Solomon case, we will first try to describe the data points $\{(P_i, y_i)\}$ by some polynomial Q. We follow [SW99] and let Q be a polynomial in a formal variable y with coefficients from K (i.e., $Q \in K[y]$). Now given a value of $y_i \in \mathbb{F}_q$, $Q(y_i)$ will yield an element of K. By definition such an element of K can be evaluated at the place $P_i \in \mathbb{P}_K$. We will require that Q have the property that $Q(P_i, y_i) \stackrel{\text{def}}{=} \mathsf{eval}(Q(y_i), P_i)$ equal zero, for every $i \in [n]$. We will actually require more and insist that (P_i, y_i) "behave" like a zero of multiplicity r of Q; since $P_i \in \mathbb{P}_K$ and $y_i \in \mathbb{F}_q$, we need to be careful in specifying the conditions to achieve this, and we will return to this shortly. We will also insist that $Q(y)$ has a small number of poles (say, at most l) at P_0 for any substitution of y with a function in $\mathcal{L}(\alpha P_0)$. Having found such a Q, we then look for *roots* $h \in \mathcal{L}(\alpha P_0)$ of Q, and winnow out those that do not satisfy $\mathsf{eval}(h, P_i) = y_i$ for at least t values of i, $1 \le i \le n$.

Description of the algorithm

Algorithm *Function-Reconstruct*$(n, \alpha, t; P_0, P_1, \ldots, P_n \in \mathbb{P}_K)$
Input: $y_1, y_2, \ldots, y_n \in \mathbb{F}_q$

Output: All functions $h \in \mathcal{L}(\alpha P_0)$ such that $\mathsf{eval}(h, P_i) = y_i$ for at least t values of $i \in \{1, \ldots, n\}$.

1. Pick parameters r and l suitably (as in the algorithm *Poly-Reconstruct*)

2. "Fit" the pairs (y_i, P_i) by a "suitable" non-zero polynomial $Q \in K[y]$. Specifically find $Q \in K[y]$, $Q \neq 0$, such that

   (i) $Q(f) \in \mathcal{L}(lP_0)$ for every $f \in \mathcal{L}(\alpha P_0)$, and

   (ii) for every $i \in \{1, 2, \ldots, n\}$ and $h \in K$, if $\mathsf{eval}(h, P_i) = y_i$ then $v_{P_i}(Q(h)) \geq r$.

3. Find all roots $h \in \mathcal{L}(\alpha P_0)$ of $Q \in K[y]$. For each of them check if $\mathsf{eval}(h, P_i) = y_i$ for at least t values of i, and if so, output h.

What remains to be done is to explicitly express the Conditions (i) and (ii) above in a manner that allows for an algorithmic solution. To ensure requirement (i), it suffices if the coefficient of the y^j term in $Q \in K[y]$ belongs to $\mathcal{L}((l - \alpha j)P_0)$. To require so, we assume that we are explicitly given basis functions $\phi_1, \ldots, \phi_{l-g+1}$ for $\mathcal{L}(lP_0)$ which satisfy $v_{P_0}(\phi_j, x_0) \geq -(j+g-1)$ (i.e., ϕ_j has at most $(j+g-1)$ poles at p_0) and $v_{P_0}(\phi_j) > v_{P_0}(\phi_{j+1})$ for $1 \leq j < l - g + 1$ (i.e., the pole orders at P_0 of ϕ_j increase with j). Let $s \stackrel{\text{def}}{=} \left\lfloor \frac{l-g}{\alpha} \right\rfloor$. We will then look for coefficients $q_{j_1, j_2} \in \mathbb{F}_q$ such that the polynomial $Q \in K[y]$ can be written of the form:

$$Q(y) = \sum_{j_2=0}^{s} \sum_{j_1=1}^{l-g+1-\alpha j_2} q_{j_1 j_2} \phi_{j_1} y^{j_2} \ . \tag{6.13}$$

By explicitly setting up Q as above, we impose the Condition (i) in the algorithm above. To enforce Condition (ii), we need to "shift" our basis. This is done exactly as in the Reed-Solomon case with respect to the y_i's; however, $P_i \in \mathbb{P}_K$ and hence a different method is required to handle it. The lemmas below show how this may be achieved.

Lemma 6.25 *For every non-zero $f, g \in K$ and $P \in \mathbb{P}_K$ such that $v_P(f) = v_P(g)$, there exist $\alpha_0, \beta_0 \in \mathbb{F}_q \setminus \{0\}$, such that $v_P(\alpha_0 f + \beta_0 g) > v_P(f)$.*

Proof: Let $v_P(f) = v_P(g) = m$ and f^{-1} be the multiplicative inverse of f in K. Then $v_P(f * f^{-1}) = 0$ and $v_P(f^{-1}) = -v_P(f) = -m$. Therefore, $v_P(g * f^{-1}) = v_P(g) + v_P(f^{-1}) = 0$. Let $\mathsf{eval}(f * f^{-1}, P) = \alpha$ and $\mathsf{eval}(g * f^{-1}, P) = \beta$. We have $\alpha, \beta \notin \{0, \infty\}$, and since P is of degree one, we have $\alpha, \beta \in \mathbb{F}_q$. Thus we find that $\mathsf{eval}(\beta f * f^{-1} - \alpha g * f^{-1}, P) = 0$. Thus $v_P(\beta f * f^{-1} - \alpha g * f^{-1}) > 0$ and so $v_P(\beta f - \alpha g) > m$, as required. □

Lemma 6.26 *Given non-zero functions $\phi_1, \ldots, \phi_p$ of distinct pole orders at P_0 satisfying $\phi_j \in \mathcal{L}((j + g - 1)P_0)$, and a place $P_i \neq P_0$, there exist non-zero functions $\psi_1, \ldots, \psi_p \in K$ that satisfy:*

(i) $v_{P_i}(\psi_j) \geq j - 1$ (i.e., ψ_j has at least $(j - 1)$ zeroes at P_i)

(ii) There exist $\alpha_{P_i, j_1, j_3} \in \mathbb{F}_q$ for $1 \leq j_1, j_3 \leq p$ such that each function ϕ_{j_1} can be expressed as a linear combination of the ψ_j's of the form: $\phi_{j_1} = \sum_{j_3=1}^{p} \alpha_{P_i, j_1, j_3} \psi_{j_3}$.

Proof: We prove a stronger statement by induction on p: If $\phi_1, \ldots, \phi_p$ are linearly independent functions (over $\mathbb{F}_q$) that satisfy $v_{P_i}(\phi_j) \geq m$ for each $j = 1, 2, \ldots, p$ for some $m \geq 0$, then there exist functions $\psi_1, \ldots, \psi_p$ with $v_{P_i}(\psi_j) \geq (m + j - 1)$ that generate the ϕ_j's over $\mathbb{F}_q$. Note that this will imply our lemma since the fact that ϕ_j's have distinct pole orders at P_0 implies that $\phi_1, \phi_2, \ldots, \phi_p$ are linearly independent (this follows easily using the fact that $v_{P_0}(\alpha f + \beta g) = \min\{v_{P_0}(f), v_{P_0}(g)\}$ if $v_{P_0}(f) \neq v_{P_0}(g)$).

The statement claimed is obviously true for $p = 1$, with the choice $\psi_1 = \phi_1$. Now let $p > 1$. Assume without loss of generality that ϕ_1 is a function with the least zero order at P_i. By assumption, ϕ_1 has at least m zeroes at P_i, i.e., $v_{P_i}(\phi_1) \geq m$. We let $\psi_1 = \phi_1$. Now, for $2 \leq j \leq p$, set $\phi'_j = \phi_j$ if $v_{P_i}(\phi_j) > v_{P_i}(\phi_1)$. Otherwise, if $v_{P_i}(\phi_j) = v_{P_i}(\phi_1)$, using Lemma 6.25 to the pair (ϕ_1, ϕ_j) of functions, we get $\alpha_j, \beta_j \in \mathbb{F}_q - \{0\}$ such that the function $\phi'_j = \alpha_j \phi_1 + \beta_j \phi_j$ satisfies $v_{P_i}(\phi'_j) > v_{P_i}(\phi_1) \geq m$. Since $\phi_j = \beta_j^{-1}\phi'_j - \alpha_j \beta_j^{-1} \phi_1$ in this case, we conclude that in any case, for $2 \leq j \leq p$, $\psi_1 = \phi_1$ and ϕ'_j generate ϕ_j. Now $\phi'_2, \phi'_3, \ldots, \phi'_p$ are linearly independent (since $\phi_1, \phi_2, \ldots, \phi_p$ are) and $v_{P_i}(\phi'_j) \geq m + 1$ for $2 \leq j \leq p$. Therefore the induction hypothesis applied to the functions $\phi'_2, \ldots, \phi'_p$ now yields $\psi_2, \ldots, \psi_p$ as required. □

We are now ready to express Condition (ii) of Algorithm *Function-Reconstruct* which requires that

$$v_{P_i}(Q(h)) \geq r \text{ for all } h \in K \text{ and } i \in \{1, 2, \ldots, n\} \text{ such that } \mathsf{eval}(h, P_i) = y_i\ . \qquad (6.14)$$

Informally, we say that the above requirement forces (P_i, y_i) to be a "zero" of multiplicity r of the polynomial Q. Using Lemma 6.26 and Equation (6.13), we know that Q has the form

$$Q(y) = \sum_{j_2=0}^{s} \sum_{j_3=1}^{l-g+1} \sum_{j_1=1}^{l-g+1-j_2\alpha} q_{j_1,j_2} \alpha_{P_i,j_1,j_3}\ \psi_{j_3,P_i}\ y^{j_2}.$$

The shifting to y_i is achieved by defining

$$Q^{(i)}(y) \stackrel{\text{def}}{=} Q(y + y_i)\ . \qquad (6.15)$$

The requirement (6.14) on Q, P_i now becomes

$$v_{P_i}(Q^{(i)}(h)) \geq r \text{ for all } h \in K \text{ such that } \mathsf{eval}(h, P_i) = 0, \text{ i.e., } \forall\ h \text{ s.t. } v_{P_i}(h) \geq 1\ . \quad (6.16)$$

Now

$$Q^{(i)}(y) = \sum_{j_4=0}^{s} \sum_{j_3=1}^{l-g+1} q^{(i)}_{j_3,j_4} \psi_{j_3,P_i} y^{j_4}\ , \qquad (6.17)$$

where

$$q^{(i)}_{j_3,j_4} \stackrel{\text{def}}{=} \sum_{j_2=j_4}^{s} \sum_{j_1=1}^{l-g+1-\alpha j_2} \binom{j_2}{j_4} y_i^{j_2-j_4} \cdot q_{j_1,j_2} \alpha_{P_i,j_1,j_3}\ . \qquad (6.18)$$

The terms in $Q^{(i)}(y)$ that are divisible by y^p contribute p towards the multiplicity of $(P_i, 0)$ as a "zero" of $Q^{(i)}$, or, equivalently, the multiplicity of (P_i, y_i) as a zero of Q. Since $v_{P_i}(\psi_{j_3,P_i}) \geq (j_3 - 1)$ by Lemma 6.26, we can achieve the required Condition (6.16) on $Q^{(i)}$, or equivalently the required Condition (6.14) on Q, by insisting that $q^{(i)}_{j_3,j_4} = 0$ for all $j_3 \geq 1$, $j_4 \geq 0$ such that $j_4 + j_3 - 1 < r$, i.e. $j_3 + j_4 \leq r$ (there are $\binom{r+1}{2}$ such constraints for each $i \in \{1, 2, \ldots, n\}$).

The above discussion shows that it is possible to solve Step 1 in Algorithm *Function-Reconstruct* by finding a non-zero solution to a homogeneous linear system (with unknowns being the coefficients q_{j_1,j_2} from the expansion (6.13) of the polynomial Q), modulo the assumption that we "know" a certain basis $\phi_1, \phi_2, \ldots, \phi_{l-g+1}$ of $\mathcal{L}(lP_0)$, and also the "basis change coefficients" α_{P_i,j_1,j_3} (from Lemma 6.26). The coefficients of the polynomial will also be found and expressed in terms of their representation in terms of the basis $\langle \phi_1, \ldots, \phi_{l-g+1} \rangle$. To solve Step 2, we need a subroutine to find roots of univariate polynomials over function fields, to which we turn next.

6.3.4 Root finding over algebraic function fields

Before discussing the root finding algorithm, we discuss some issues concerning the representation of elements of a function field that deserve attention at this point. The decoding algorithm discussed above relies strongly on the algebraic properties of the underlying function field. In particular, it relies on the ability to perform certain operations over these fields. These operations include basic field operations such as addition and multiplication, but also some non-trivial operations such as evaluating functions at "places", and finding roots of polynomials over these fields.

One of the essential bottlenecks towards the unified presentation of our algorithm for *all* AG-codes is that one needs a generic method to represent the elements of an algebraic function field so that (a) these representations are short for the elements of interest in the construction of the algebraic-geometric codes; and (b) basic operations are efficient under this representation. By carefully picking the representation of the algebraic-geometric code, we are able to meet both the requirements above. By doing so, we are able to present a compact and general theorem about list decoding of AG-codes.

As mentioned earlier, algorithms that involve function fields, and in particular decoding algorithms for AG-codes, raise several issues on how to represent elements from the function field K and the places $\mathbb{P}_K$.

In order to implement Algorithm *Function-Reconstruct* efficiently over a function field K, we would like to perform the following basic operations efficiently: (i) Given two elements $x, y \in K$, compute their sum and product in K; (ii) Given $f \in K$ and $P \in \mathbb{P}_K$, compute $v_P(f)$ and $\mathsf{eval}(f, P)$; and (iii) Given a divisor $D \succeq 0$, compute a basis for the vector space $\mathcal{L}(D)$ over $\mathbb{F}_q$ (it suffices to solve this for one-point divisors $D = \alpha P_0$).

It is a priori unclear that these operations can performed in polynomial time for *every* function field K. First of all, the function field K is an infinite set, so one cannot assume that operations in K (like sum and product) are unit operations. Instead one must fix a representation for the elements and give explicit algorithms to perform these operations that are efficient with respect to the size of the representation of an element. A natural representation to consider is to express elements of K as ratio of two homogeneous multivariate polynomials. For this representation, the field operations in K can be done in time polynomial in the sum of degrees of the respective polynomials. However, for question (iii) above, it is *not* known whether for general function fields there always exists a basis for $\mathcal{L}(D)$ over $\mathbb{F}_q$ with a succinct representation (i.e., one of size polynomial in $\deg(D)$) as the ratio of polynomials (see [SW99] for a discussion concerning this point).

For applications to decoding, one does not need to work with all of K, but instead can focus attention only on elements in $\mathcal{L}(D)$ for some divisor $D \succeq 0$ (in fact for $D = lP_0$ for some place P_0). This allows us the option of representing elements of $\mathcal{L}(D)$ as vectors in $\mathbb{F}_q^{\dim(\mathcal{L}(D))}$ which represent their coordinates with respect to some *fixed* basis of $\mathcal{L}(D)$

over $\mathbb{F}_q$. Since it is known that $\dim(\mathcal{L}(D)) \leq \deg(D)+1$, this representation will be small provided $\deg(D)$ is small. Indeed, this is what we exploited already in our "solution" to Step 1 of Algorithm *Function-Reconstruct*. In order to be able to perform the root-finding step also efficiently, we augment this representation suitably. Before explaining this, we formally describe the basic root-finding task that we wish to solve, and describe an algebraic algorithm to solve it (this is along the lines of the algorithms of [NH99a, GS00]). We then discuss the representation issues this algorithm motivates, and in the next section give a full list decoding algorithm with the required representation explicitly spelled out.

Procedure ROOT-FIND$_{(K,D)}(Q)$

Input: A degree m polynomial $Q = \sum_{i=0}^{m} a_i y^i \in K[y]$ where each $a_i \in \mathcal{L}(D)$ for some divisor $D \succeq 0$.

Output: All roots of Q that lie in $\mathcal{L}(D)$.

1. Let $R \in \mathbb{P}_K$ be a place (outside $\mathrm{supp}(D)$) that has degree $r > \deg(D)$. "Reduce" H modulo the place R — namely, compute $b_i = \mathsf{eval}(a_i, R)$ for $0 \leq i \leq m$ and consider the polynomial $P = \sum_{i=0}^{m} b_i Y^i \in \mathbb{F}_{q^r}[Y]$.

2. Compute the roots, say $\alpha_1, \ldots, \alpha_t$, of P that lie in $\mathbb{F}_{q^r}$ using a root-finding algorithm for finite fields. (This can be accomplished in deterministic $\mathrm{poly}(q,r)$ time by an algorithm due to Berlekamp [Ber70].)

3. For each α_j, $1 \leq j \leq t$, "find" $\beta_j \in \mathcal{L}(D)$ such that $\mathsf{eval}(\beta_j, R) = \alpha_j$, if any such β_j exists.

The tricky issue in the above algorithm is that we need to find a place R of large enough degree and then be able to evaluate functions $f \in \mathcal{L}(D)$ at R. To aid this we "represent" a place R by the values $\mathsf{eval}(\phi_i, R)$, for $1 \leq i \leq p$ where $p = \dim(\mathcal{L}(D))$ and $\phi_1, \ldots, \phi_p$ is a basis of $\mathcal{L}(D)$ over $\mathbb{F}_q$. Together with the representation of any element of $\mathcal{L}(D)$ as a linear combination of the ϕ_i's this clearly enables us to evaluate any element of $\mathcal{L}(D)$ at R. Since each $\mathsf{eval}(\phi_i, R) \in \mathbb{F}_{q^r}$ where $r = \deg(R)$, for purposes of evaluation by members of $\mathcal{L}(D)$, one can represent R as a vector of length p with entries in $\mathbb{F}_{q^r}$ and present it as auxiliary "advice" input to the root-finding algorithm. Given this table of evaluations of the basis functions at R, Step 3 just amounts to solving a linear system of equations over $\mathbb{F}_q$, and therefore can also be performed efficiently. (We will shortly argue that for each α_j, there can exist at most one $\beta_j \in \mathcal{L}(D)$ such that $\mathsf{eval}(\beta_j, R) = \alpha_j$. Therefore, the operation of Step 3, which "lifts" roots in $\mathbb{F}_{q^r}$ to roots in $\mathcal{L}(D)$, is well-defined.)

It should be clear that given this representation, the above algorithm can in fact be implemented efficiently. We next argue the correctness of the above root-finding procedure. Then we will integrate it with the solution to the interpolation step (Step 1) of *Function-Reconstruct* to describe the full list decoding algorithm for AG-codes, with all the representation details explicitly spelled out.

The two simple lemmas below are necessary for the correctness of the algorithm.

Lemma 6.27 *For any function field K, there exists a place of degree m in $\mathbb{P}_K$ for every large enough integer m.*

Proof: By the Hasse-Weil Bound (see, for example, [Sti93, Theorem V.2.3]), the number N_m of places of degree m in $\mathbb{P}_K$ satisfies $|N_m - q^m - 1| \leq 2gq^{m/2}$ where g is the genus of the function field K. Hence if $m \geq m_0$ where m_0 is the smallest integer that satisfies $\frac{q^{m_0}-1}{2q^{m_0/2}} > g$, then $N_m \geq 1$. □

Lemma 6.28 *If $f_1, f_2 \in \mathcal{L}(A)$ for some divisor $A \succeq 0$ and $\mathsf{eval}(f_1, R) = \mathsf{eval}(f_2, R)$ for some place R with $\deg(R) > \deg(A)$, then $f_1 = f_2$.*

Proof: Suppose not, so that $f_1 - f_2 \neq 0$. Then, by Proposition 6.22, $\deg((f_1 - f_2)) = 0$. But $f_1 - f_2 \in \mathcal{L}(A)$ and $v_R(f_1 - f_2) \geq 1$, so that $\deg((f_1 - f_2)) \geq \deg(R) - \deg(A) > 0$, a contradiction. Hence $f_1 = f_2$. □

The correctness of *ROOT-FIND* follows easily from the above two lemmas. The first lemma implies that a place R as required in the first step of *ROOT-FIND* exists. The second lemma implies that reducing the polynomial modulo R (of degree greater than $\deg(D)$) "preserves" all its roots that lie in $\mathcal{L}(D)$, since "lifting" the root from $\mathbb{F}_{q^r}$ to $\mathcal{L}(D)$ is an "injective" operation.

It is clear that given the place R as advice in the form of the table of values of $\mathsf{eval}(\phi, R)$ for ϕ ranging over a set B of basis functions for $\mathcal{L}(D)$, the root-finding algorithm can be implemented in polynomial time. (We also assume the coefficients of the polynomial Q, which are elements of $\mathcal{L}(D)$, are input in the form of the coefficients of their expansion in terms of the same basis B.) We can thus record the following which states that the root-finding step of Algorithm *Function-Reconstruct* can be implemented efficiently.

Theorem 6.29 *There is an efficient root-finding algorithm that, for any function field K and any divisor $D \succeq 0$, given an "advice" that depends only on D and is of size polynomial in $\deg(D)$, finds, in $\mathrm{poly}(m, \deg(D))$ time, all roots in $\mathcal{L}(D)$ of any input degree m polynomial in $K[y]$ all of whose coefficients lie in $\mathcal{L}(D)$.*

6.3.5 An explicit list decoding algorithm

With the individual efficient implementations of both the interpolation and root-finding steps of *Function-Reconstruct* in place, we now move on to the full description of an explicit polynomial time list decoding algorithm for AG-codes, together with all parameter choices. The algorithm works with a polynomial size advice (or "non-uniform" input) that depends only the code (and not on the received word which is being decoded). This will imply that AG-codes admit a succinct representation given which *Function-Reconstruct* can be implemented to run in polynomial time. In the next section, we will formally establish the correctness of the algorithm and analyze its error-correction performance for our specific parameter choices.

We only discuss the version of *Function-Reconstruct* for decoding with polynomial-sized lists (i.e., the result of Theorem 6.8 generalized to AG-codes). The analogous generalization of Theorem 6.6 for decoding with constant-sized lists to AG-codes follows similarly — we omit the details. Recall that the object of *Function-Reconstruct* is to find, for input pairs (P_i, y_i), $1 \leq i \leq n$, where each $P_i \in \mathbb{P}_K$ is a place of $K/\mathbb{F}_q$ and each $y_i \in \mathbb{F}_q$, a list of all $h \in \mathcal{L}(\alpha P_0)$ such that $\mathsf{eval}(h, P_i) = y_i$ for at least t values of i.

Algorithm *Function-Reconstruct*$(n, \alpha, t; P_0, P_1, \ldots, P_n \in \mathbb{P}_K)$

Input: $y_1, y_2, \ldots, y_n \in \mathbb{F}_q$

Output: All functions $h \in \mathcal{L}(\alpha P_0)$ which satisfy $\mathsf{eval}(h, P_i) = y_i$ for at least t values of $i \in \{1, \ldots, n\}$.

Parameters: n, α, t; the genus g of K. Based on these the algorithm computes parameters r, l.

Non-uniform input: Fix a set of linearly independent functions $\{\phi_{j_1} : 1 \le j_1 \le l - g + 1\} \subseteq \mathcal{L}(lP_0)$ such that $v_{P_0}(\phi_{j_1}) \ge -(j_1 + g - 1)$ (i.e., ϕ_{j_1} has at most $(j_1 + g - 1)$ poles at P_0). Note that these functions span a subspace W of $\mathcal{L}(lP_0)$ of dimension $(l - g + 1)$, and by the Riemann-Roch theorem, if $l \ge 2g - 1$, they span the entire space $\mathcal{L}(lP_0)$. As shown in Lemma 6.26, for each i, $1 \le i \le n$, there exists a basis $\{\psi_{j_3,P_i} : 1 \le j_3 \le l - g + 1\}$ of the subspace W of $\mathcal{L}(lP_0)$ such that $v_{P_i}(\psi_{j_3,P_i}) \ge j_3 - 1$. The explicit information which the decoding algorithm needs as advice information (or non-uniform input) is the following:

(a) The values $\mathsf{eval}(\phi_{j_1}, P_i) \in \mathbb{F}_q$ for $1 \le i \le n$ and $1 \le j_1 \le l - g + 1$.

(b) The "basis change" coefficients $\{\alpha_{P_i,j_1,j_3} \in \mathbb{F}_q : 1 \le i \le n, 1 \le j_1, j_3 \le l - g + 1\}$ such that for every i and every j_1, we have $\phi_{j_1} = \sum_{j_3} \alpha_{P_i,j_1,j_3} \psi_{j_3,P_i}$ (as elements in K).

(c) A place $R \in \mathbb{P}_K$ with $\deg(R) = s > l$ represented through the table of values $\mathsf{eval}(\phi_{j_1}, R)$ for $1 \le j_1 \le l - g + 1$ (note that each such evaluation lies in $\mathbb{F}_{q^s}$).

(We stress here that the above information depends on the specific set of places $P_0, P_1, \ldots, P_n$, but *not* on the y_i's, and is moreover of size polynomial in n — indeed it is of size $O(nl^2 \log q)$, and l is at most cubic in n for the choice made in the algorithm. In our application to decoding AG-codes, this means that the polynomial amount of advice information necessary for our algorithm depends only on the code and not on the actual received word that is being decoded. In other words, we can simply view the above information as comprising a non-standard, but succinct, representation of the underlying AG-code.)

Step 0: Compute parameters r, l which satisfy

$$rt > l \text{ and } \frac{(l-g)(l-g+1)}{2\alpha} > n\binom{r+1}{2} . \tag{6.19}$$

In particular, set

$$r \stackrel{\text{def}}{=} 1 + \left\lfloor \frac{2gt + \alpha n + \sqrt{(2gt+\alpha n)^2 - 4(g^2-1)(t^2-\alpha n)}}{2(t^2 - \alpha n)} \right\rfloor , \tag{6.20}$$

$$l \stackrel{\text{def}}{=} rt - 1 . \tag{6.21}$$

Step 1: ***(Interpolation Step)*** Find $Q \in \mathcal{L}(lP_0)[y]$ of the form

$$Q(y) = \sum_{j_2=0}^{s} \sum_{j_1=1}^{l-g+1-\alpha j_2} q_{j_1 j_2} \phi_{j_1} y^{j_2} ,$$

for $s \stackrel{\text{def}}{=} \left\lfloor \frac{l-g}{\alpha} \right\rfloor$; i.e., find values of the coefficients $\{q_{j_1,j_2} \in \mathbb{F}_q\}$ such that the following conditions hold:

1. At least one q_{j_1,j_2} is non-zero (so that Q is a non-zero polynomial in $K[y]$).
2. For every $i \in [n]$, $\forall j_3, j_4$, $j_3 \ge 1$, $j_4 \ge 0$ such that $j_3 + j_4 \le r$,

$$q^{(i)}_{j_3,j_4} \stackrel{\text{def}}{=} \sum_{j_2=j_4}^{s} \sum_{j_1=1}^{l-g+1-\alpha j_2} \binom{j_2}{j_4} y_i^{j_2-j_4} \cdot q_{j_1,j_2} \alpha_{P_i,j_1,j_3} = 0.$$

Step 2: ***(Root-finding step)*** Using the root-finding algorithm *ROOT-FIND* from Section 6.3.4 together with the place R which is supplied to the algorithm, "find" all roots $h \in \mathcal{L}(\alpha P_0) \subseteq \mathcal{L}(lP_0)$ of the polynomial $Q \in K[y]$. For each such h, check if $\mathsf{eval}(h, P_i) = y_i$ for at least t values of i, and if so, include h in output list. (Since h is "found" by finding its coefficients with respect to the basis functions ϕ_{j_1} and the algorithm is given the values $\mathsf{eval}(\phi_{j_1}, P_i)$ for $1 \le i \le n$ and $1 \le j_1 \le l - g + 1$, each $\mathsf{eval}(h, P_i)$ can be computed efficiently.)

Step 3: Output the list of all functions $h \in \mathcal{L}(\alpha P_0)$ found in Step 2.

Since Step 1 just involves solving a homogeneous linear system of equations and Step 2 involves root-finding for which we gave an efficient algorithm in Section 6.3.4, it is clear that the above algorithm runs in polynomial time given the advice information it takes as input. Since the list decoding problem for AG-codes reduces to the Function Reconstruction problem, we thus have a polynomial time list decoding algorithm for AG-codes (assuming the required representation of the code). We next analyze the error-correction performance of the algorithm for the specific choice of parameters made in the algorithm.

6.3.6 Analysis of the Algorithm

We now analyze the performance of *Function-Reconstruct*. We first verify that the choice of r, l made in the algorithm satisfy the required Condition (6.19).

Lemma 6.30 *If n, α, t satisfy $t^2 > \alpha n$, then for the choice of r, l made in the algorithm (in Equations (6.20) and (6.21)), the conditions $\frac{(l-g)(l-g+2)}{2\alpha} > n\binom{r+1}{2}$ and $rt > l$ both hold.*

Proof: The proof parallels that of Lemma 6.7. The condition $rt > l$ certainly holds since we pick $l \stackrel{\text{def}}{=} rt - 1$. Using $l = rt - 1$, the other constraint becomes

$$\frac{(rt-g)^2 - 1}{2\alpha} > n\binom{r+1}{2}$$

which simplifies to

$$r^2(t^2 - \alpha n) - (2gt + \alpha n)r + (g^2 - 1) > 0.$$

If $t^2 - \alpha n > 0$, it suffices to pick r to be an integer greater than the larger root of the above quadratic, and therefore picking

$$r \stackrel{\text{def}}{=} 1 + \left\lfloor \frac{2gt+\alpha n+\sqrt{(2gt+\alpha n)^2-4(g^2-1)(t^2-\alpha n)}}{2(t^2-\alpha n)} \right\rfloor$$

suffices, and this is exactly the choice made in the algorithm. □

Lemma 6.31 *If $n\binom{r+1}{2} < \frac{(l-g)(l-g+2)}{2\alpha}$, then $Q(y)$ as sought in Step 1 does exist (and can be found in polynomial time by solving a linear system).*

Proof: The proof follows that of Lemma 6.4. The computational task in Step 1 is once again that of solving a homogeneous linear system. A non-trivial solution exists as long as the number of unknowns exceeds the number of constraints. The number of constraints in the linear system is $n\binom{r+1}{2}$, while the number of unknowns equals

$$\sum_{j_2=0}^{s}(l-g+1-\alpha j_2) \geq \frac{(l-g)(l-g+2)}{2\alpha}. \qquad \Box$$

We next prove that any Q found in the interpolation step will have all the required functions h, namely those that satisfy $\mathsf{eval}(h, P_i) = y_i$ for at least t values of i, as roots.

Lemma 6.32 *For $i \in \{1, 2, \ldots, n\}$, if $h \in K$ satisfies $\mathsf{eval}(h, P_i) = y_i$, then $v_{P_i}(Q(h)) \geq r$.*

Proof: Using Equation (6.17), we have the following for every place P:

$$\mathsf{eval}(Q(h), P) = \sum_{j_4=0}^{s}\sum_{j_3=1}^{l-g+1} q_{j_3,j_4}^{(i)} \mathsf{eval}(\psi_{j_3,P_i}, P)(\mathsf{eval}(h, P) - y_i)^{j_4} .$$

Now if $\mathsf{eval}(h, P_i) = y_i$, we get

$$\mathsf{eval}(Q(h), P) = \sum_{j_4=0}^{s}\sum_{j_3=1}^{l-g+1} q_{j_3,j_4}^{(i)} \mathsf{eval}(\psi_{j_3,P_i}, P)(\mathsf{eval}(h, P) - \mathsf{eval}(h, P_i))^{j_4} . \tag{6.22}$$

By our choice of Q, $q_{j_3,j_4}^{(i)} = 0$ for $j_3 + j_4 \leq r$. Also, $v_{P_i}(\psi_{j_3,P_i}) \geq j_3 - 1$, and if $h^{(i)}$ is defined by its value on places in $\mathbb{P}_K$ as $\mathsf{eval}(h^{(i)}, P) \stackrel{\text{def}}{=} \mathsf{eval}(h, P) - \mathsf{eval}(h, P_i)$, then $v_{P_i}((h^{(i)})^{j_4}) \geq j_4$. It then follows from Equation (6.22) that $v_{P_i}(Q(h)) \geq r$. $\Box$

Lemma 6.33 *If $h \in \mathcal{L}(\alpha P_0)$ is such that $\mathsf{eval}(h, P_i) = y_i$ for at least t values of $i \in \{1, 2, \ldots, n\}$ and $rt > l$, then $Q(h) \equiv 0$; i.e. h is a root of $Q \in K[y]$.*

Proof: By our choice of Q in the interpolation step, we have $Q(h) \in \mathcal{L}(lP_0)$ for all $h \in \mathcal{L}(\alpha P_0)$. Hence $v_{P_i}(Q(h)) \geq 0$ for each $i \in [n]$. If $\mathsf{eval}(h, P_i) = y_i$ for at least t values of i, using Lemma 6.32, we get $\sum_{i \in [n]} v_{P_i}(Q(h)) \geq rt > l$, and hence the zero order of $Q(h)$ is greater than l. Since $Q(h) \in \mathcal{L}(lP_0)$, the pole order of $Q(h)$ is at most l. Since there are more zeroes than poles for $Q(h)$, appealing to Proposition 6.22, we therefore conclude that we must have $Q(h) \equiv 0$. Thus h is a root of Q. $\Box$

Our main theorem on list decoding AG-codes now follows from Lemmas 6.30-6.33 and the polynomial runtime claimed in the previous section.

Theorem 6.34 *Let $\mathcal{C} = \mathcal{C}_{\mathcal{L}}(G, \alpha, P_0)$ be an AG-code of blocklength n and designed minimum distance $d^* = n - \alpha$. Then there exists a representation of the code of size polynomial in n, given which there exists a polynomial time list decoding algorithm for $\mathcal{C}$ that decodes up to $e < n - \sqrt{\alpha n} = n - \sqrt{n(n - d^*)}$ errors.*

6.3.7 Weighted list decoding of AG-codes

We now state the generalization of the result of Theorem 6.34 to an algorithm that can exploit soft information (weights), similar to the soft decoding algorithm for Reed-Solomon codes from Section 6.2.10. The proof method is similar to the Reed-Solomon case and involves using multiplicities in proportion to the weights of the various pairs in the interpolation step. We omit the details.

Theorem 6.35 *For every q-ary AG-code $\mathcal{C}$ of blocklength n and designed minimum distance $d^* = n - \alpha$, there exists a representation of the code of size polynomial in n under which the following holds. Let $\epsilon > 0$ be an arbitrary constant. For $1 \le i \le n$ and $\gamma \in \mathbb{F}_q$, let $w_{i,\gamma}$ be a non-negative real. Then one can find in* $\mathrm{poly}(n, q, 1/\epsilon)$ *time, a list of* all *codewords* $\mathbf{c} = \langle c_1, c_2, \ldots, c_n \rangle$ *of $\mathcal{C}$ that satisfy*

$$\sum_{i=1}^{n} w_{i,c_i} \ge \sqrt{(n - d^*) \sum_{i=1}^{n} \sum_{\gamma \in \mathbb{F}_q} w_{i,\gamma}^2} + \epsilon \max_{i,\gamma} w_{i,\gamma} \ . \tag{6.23}$$

6.3.8 Decoding up to the "q-ary Johnson radius"

For Reed-Solomon codes of blocklength n and minimum distance d, the quantity $n - \sqrt{n(n-d)}$ closely approximates the Johnson radius since the alphabet size is very large (it is at least n). Similarly, for AG-codes over very large alphabets, the result of Theorem 6.34 decodes almost up to the Johnson bound on list decoding radius of the concerned AG-code. However, as discussed earlier, AG-codes of growing blocklength can also be defined over a fixed alphabet, say q. When $q \ll n$, the quantity $(n - \sqrt{n(n-d)})$, while always greater than $d/2$ so that we are still decoding beyond half the designed minimum distance, is no longer an accurate estimate of the Johnson radius for q-ary codes (which the reader might recall from Theorem 3.2 of Chapter 3 is $n(1 - \frac{1}{q}) \cdot \left(1 - \sqrt{1 - \frac{d/n}{1-1/q}}\right)$).

However, as was insightfully noted by Koetter and Vardy [KV00], it is possible to improve the number of errors corrected by our result from Theorem 6.34, using the soft decoding algorithm from Theorem 6.35. This is quite surprising since the result of Theorem 6.34 is for hard decoding where the channel just outputs one of the q symbols as the received symbol at each position, and thus (seemingly) provides no soft information whatsoever. In fact, using the soft decoding algorithm with the right choice of weights, one can decode up to (exactly) the q-ary Johnson bound on list decoding radius! Thus, as with Reed-Solomon codes, algebraic-geometric codes can also be efficiently decoded up to their "a priori combinatorial list decoding potential", namely the *q-ary Johnson radius*.

The actual setting of weights which allows for decoding up to the q-ary Johnson radius is the following: Let $\delta = d^*/n$ be the relative designed distance of the AG-code, and let

$$\tau \stackrel{\text{def}}{=} \left(1 - \frac{1}{q}\right)\left(1 - \sqrt{1 - \frac{q\delta}{q-1}}\right) \tag{6.24}$$

be the desired fraction of errors to be list decoded (this is just the q-ary Johnson radius normalized by the blocklength). It is a straightforward calculation to check that such a τ satisfies

$$\frac{\tau^2}{q-1} + (1-\tau)^2 = 1 - \delta \ . \tag{6.25}$$

If the i'th received symbol of the received word $\mathbf{y}$ is $y_i \in \mathbb{F}_q$, then we set $w_{i,y_i} = 1 - \tau$ and $w_{i,\gamma} = \tau/(q-1)$ for $\gamma \in \mathbb{F}_q \setminus \{y_i\}$, and then apply Theorem 6.35 to this choice of weights. Assuming there are e errors, Condition (6.23) of Theorem 6.35 implies that the algorithm successfully finds all codewords within Hamming distance e from $\mathbf{y}$ provided

$$(n-e)\cdot(1-\tau) + e\cdot\frac{\tau}{q-1} > \sqrt{(n-d^*)n\Big(\frac{\tau^2}{q-1} + (1-\tau)^2\Big)} \ . \tag{6.26}$$

We now prove that for τ defined as in Equation (6.24), the above condition is satisfied as long as $e < \tau n$. (We have ignored the ϵ slack term needed in Condition (6.23) for convenience, but this has a negligible effect on the performance, which is explicitly accounted for in the formal statement below.) Indeed, setting $e = \gamma n$ (for some $\gamma < \tau$) and using Equation (6.25), Condition (6.26) above is satisfied as long as

$$\begin{aligned} (1-\gamma)(1-\tau) + \frac{\gamma\tau}{q-1} &> 1-\delta \\ \Longleftarrow 1-\tau-\gamma\Big(1-\frac{q\tau}{q-1}\Big) &> 1-\delta \ . \end{aligned} \tag{6.27}$$

Since $\tau < (1-1/q)$ and $\gamma < \tau$, the left hand side of the above inequality is greater than

$$1-\tau-\tau\Big(1-\frac{q\tau}{q-1}\Big) = \frac{\tau^2}{q-1} + (1-\tau)^2 = 1-\delta \ ,$$

as desired, where the last step follows from Equation (6.25).

Hence, one can efficiently list decode q-ary AG-codes of relative designed distance δ up to a fraction τ of errors, for τ as defined in Equation (6.24), or, in other words, up to the q-ary Johnson bound on list decoding radius. For easy reference, we state the formal result below.

Theorem 6.36 ([GS99, KV00]) *For every $\epsilon > 0$ and for every q-ary AG-code $\mathbf{C}$ of block-length n and designed relative distance δ, there exists a representation of the code of size polynomial in n, given which there exists a polynomial time list decoding algorithm for $\mathbf{C}$ that decodes up to*

$$n\Big(1-\frac{1}{q}\Big)\Big(1-\sqrt{1-\frac{q\delta}{q-1}}-\epsilon\Big)$$

errors.

6.3.9 List decodability offered by the best-known AG-codes

The results of the previous sections imply that for AG-codes, assuming they are suitably represented, there are efficient list decoding algorithms to decode up to the Johnson radius (and hence beyond half the designed distance). We now apply these results to the best known AG-codes (in terms of the asymptotic rate vs. distance trade-off) in order to infer the existence of very good list decodable codes. We next present a discussion of the various best known constructions of algebraic-geometric codes.

Let $K/\mathbb{F}_q$ be a function field that has at least $n+1$ places $P_0, P_1, \ldots, P_n$ of degree one. Let C be a q-ary algebraic-geometric code $\mathcal{C}_{\mathcal{L}}(G, \alpha, P_0)$ over K of blocklength n and designed distance $d^* = n - \alpha$. Let $g = g(K)$ be the genus of K. By Proposition 6.24, the dimension k of C is at least $\alpha - g + 1$, and distance d is at least the designed distance $d^* = n - \alpha$. Hence the rate R and relative distance δ of C satisfy $R + \delta \geq 1 - g/n$.

In order to obtain the best trade-off between R and δ for an asymptotically good family of AG-codes, it is therefore desirable to find a sequence of function fields $K_i/\mathbb{F}_q$, $i \geq 1$, such that each K_i has at least $n_i + 1$ places of degree one where $n_i \to \infty$ as $i \to \infty$, and K_i has genus g_i, with $\limsup_i g_i/n_i < 1$ and as small as possible. Constructions of such a sequence of function fields is a non-trivial task. In fact the limiting value of ratio g_i/n_i for a sequence of function fields $K_i/\mathbb{F}_q$ with $n_i \to \infty$ cannot be smaller than $1/(\sqrt{q} - 1)$ — this is the so called *Drinfeld-Vlădut bound* [DV83] (see also [Sti93, Section V.3]). The amazing fact, which is a major accomplishment in the theory of algebraic function fields, is that for every q which is an even power of a prime, there are known constructions of towers of function fields that meet the Drinfeld-Vlădut bound, i.e., they achieve $\limsup_i g_i/n_i = 1/(\sqrt{q} - 1)$. The first such constructions were due to Ihara [Iha81] and Tsfasman, Vlădut and Zink [TVZ82]. The authors of [TVZ82] combined the construction of such function fields together with Goppa's idea of obtaining codes from algebraic curves, to obtain a major breakthrough result in coding theory. Specifically, they obtained the trade-off $R + \delta \geq 1 - \frac{1}{\sqrt{q}-1}$ between the rate R and relative distance δ for a family of linear codes over $\mathbb{F}_q$ when q was a square. This gives an improvement over the Gilbert-Varshamov bound $R \geq 1 - H_q(\delta)$ (which is the trade-off achieved by random linear codes) for a certain range of δ for $q \geq 49$. The construction of [TVZ82] was, however, very complicated and their proofs required deep results from algebraic geometry and the theory of modular curves. The concerned modular curves were not explicitly specified and it was extremely hard to obtain algorithms of reasonable time complexity to compute a representation of the concerned AG-code (even though Manin and Vlădut [MV85] gave an algorithm — of high complexity — showing how to construct the corresponding codes; see also [KTV84, L9́8] for a discussion of algorithms for code construction on modular curves, and the work of Elkies [Elk97] for a discussion of explicit equations for certain modular curves).

In a major step forward in 1995, Garcia and Stichtenoth [GS95b, GS96] (see also the survey [GS95a]) presented two explicitly described towers of function fields that attain the Drinfeld-Vlădut bound for every square prime power q. These constructions are a lot simpler than the constructions due to Ihara or Tsfasman *et al* [TVZ82]. In a significant recent development, Shum *et al* [SAK+01] (see also [Shu00]) present a near-cubic time algorithm to compute the generator matrix of the codes corresponding to the algebraic curves from [GS96]. We record the above discussion in the following statement:

Fact 6.1 *For every q which is an even power of a prime, there is a polynomial time construction of algebraic-geometric codes whose rate R and relative distance δ satisfy*

$$R + \delta \geq 1 - \frac{1}{\sqrt{q} - 1} \ .$$

The above trade-off is often referred to as the "TVZ bound".

As a corollary we have the following which gives good constructions of codes with fractional list decoding radius $(1 - \varepsilon)$.

Corollary 6.37 *For every $\varepsilon > 0$, there exist a polynomial time constructible family $\mathcal{C}_\varepsilon$ of AG-codes with rate $R(\mathcal{C}_\varepsilon) = \Omega(\varepsilon^2)$, relative designed distance $\delta(\mathcal{C}_\varepsilon) \geq (1 - O(\varepsilon^2))$ over an alphabet of size $q(\mathcal{C}_\varepsilon) = O(1/\varepsilon^4)$. Applying the Johnson bound from Theorem 3.2, we have $\mathsf{LDR}_L(\mathcal{C}_\varepsilon) \geq 1 - \varepsilon$ for $L = O(1/\varepsilon^2)$.*

By Theorem 6.34, we have a polynomial time algorithm to list decode an AG-code of relative designed distance δ up to a fractional radius $(1 - \sqrt{1-\delta})$. Using this result on the codes from the above corollary, we get the following:

Theorem 6.38 *For every $\varepsilon > 0$, there exists a polynomial time constructible family of AG-codes with the following properties:*

(i) It is defined over an alphabet of size $O(1/\varepsilon^4)$.

(ii) It has rate $\Omega(\varepsilon^2)$ and relative distance at least $(1 - O(\varepsilon^2))$.

(iii) There exists a representation of each code of the family, of size polynomial in its blocklength, given which there is a polynomial time decoding algorithm to list decode the code up to a fraction $(1-\varepsilon)$ of errors, using lists of size $O(1/\varepsilon^2)$.

The above result gives codes of very good list decodability (list decoding radius $(1-\varepsilon)$) and reasonable (namely, $\Omega(\varepsilon^2)$) rate. The result of Theorem 5.1 implies that the best possible rate for codes with such list decodability is $\Theta(\varepsilon)$ (for an alphabet size of $1/\varepsilon^{O(1)}$). Hence the above result, while providing a non-trivial and interesting trade-off between list decodability and rate for codes over a large alphabet, is not optimal. Moreover, the decoding complexity of the codes is quite high due to the corresponding situation for algebraic-geometric codes. It is also not known (at least so far) if the representation of the code necessary for decoding can be found in polynomial time (we only know that it is succinct).[9] In light of these limitations of the result of Theorem 6.38, in Chapter 9, we will return to the question of alternate lower complexity constructions of codes that are efficiently list decodable up to a fraction $(1-\varepsilon)$ of errors.

6.4 Concluding Remarks and Open Questions

We have given a polynomial time algorithm to decode up to a fraction $(1-\sqrt{1-\delta})$ of errors for Reed-Solomon codes of relative distance δ. We also generalized the algorithm for the broader class of algebraic-geometric codes. Our algorithm is able to correct a number of errors exceeding half the minimum (designed) distance for any rate. We also presented soft decoding algorithms for these codes. The main results of this chapter are Theorems 6.13, 6.16, and 6.21 for Reed-Solomon decoding, and Theorems 6.34 and 6.35 for list decoding AG-codes.

The Reed-Solomon list decoding algorithm, in addition to its obvious importance to coding theory and practice, is also at the core of several complexity-theoretic applications. The main common theme of these applications is to deduce average-case hardness results for certain functions based on worst-case hardness assumptions. List decoding provides a way to "recover" the codeword even when several symbols are in error, and this (roughly) corresponds, in the complexity theory applications, to being able to compute the function on every input given, say, a circuit to compute it (or a related function) on a small fraction of the inputs. More details on these applications can be found in Chapter 12.

The Reed-Solomon decoding algorithm is also used in list decoding algorithms for Reed-Muller codes, using clever reductions of the multivariate polynomial reconstruction problem to the univariate polynomial reconstruction problem (cf. [AS97, STV99]).

[9] Though, judging by the recent progress made by [SAK$^+$01]) on the question of the generator matrices of such codes, we believe an answer in the affirmative to this question will be found soon.

Some natural questions left open regarding the material of this chapter are mentioned below. The first question concerns the true limit on the number of efficiently correctable errors for Reed-Solomon codes.

Open Question 6.1 *Can one efficiently list decode a family of Reed-Solomon codes of rate r beyond a fraction $(1-\sqrt{r})$ of errors? As a first step, what is the true list decoding radius (for polynomial-sized lists) for Reed-Solomon codes?*

Partial progress on the latter question above for decoding with constant-sized lists appears in [JH01, RR01]. (As mentioned earlier, these results provide good evidence that the performance of Theorem 6.6 is tight for decoding with constant-sized lists.) We conjecture that asymptotically, $(1-\sqrt{r})$ is the largest fraction of errors that can be decoded with polynomial-sized lists for Reed-Solomon codes of rate r, for every value of the rate r. If true, this will make our decoding algorithm for Reed-Solomon codes optimal in terms of the fraction of errors corrected. A resolution of this conjecture appears rather difficult, though. The next question concerns the computational complexity of list decoding Reed-Solomon codes.

Open Question 6.2 *Is there a near-linear time (i.e $O(n^{1+o(1)})$ time) list decoding algorithm for decoding an $[n, k+1, n-k]$ Reed-Solomon code up to a radius $n-(1+\varepsilon)\sqrt{kn}$ (for $\varepsilon > 0$ a fixed, but arbitrarily small constant)?*

Recall that the best runtime known so far is quadratic in the blocklength.

Open Question 6.3 *For a family of AG-codes that meet the Drinfeld-Vlădut bound (for example the codes based on the Garcia-Stichtenoth tower of function fields), can one compute the non-standard representation necessary for our list decoding algorithm from Section 6.3.5 in polynomial time?*

6.5 Bibliographic notes

The first time Reed-Solomon codes appeared as codes was in 1960 in the work of Reed and Solomon [RS60], though they had already been explicitly constructed by Bush [Bus52] in 1952, using the language of orthogonal arrays. Though their importance was not immediately realized, Reed-Solomon codes have since received a lot of attention and study. The Reed-Solomon decoding problem itself has a long history and is one of the central problems in all of coding theory. The first polynomial time algorithm to decode up to half the distance was discovered by Peterson [Pet60], even before the notion of polynomial time was formalized as a metric of feasibility of an algorithm! Owing to the importance of the problem, several works have investigated more efficient implementations of the algorithm: here we mention the Berlekamp-Massey algorithm [Ber68, Section 7.4], [Mas69] and the Euclid-based algorithm [Bla83, Chapter 7], [SKHN75]. These are "syndrome computation" based algorithms, and have quadratic runtimes which can be improved to near-linear time (specifically, $O(n \log^{O(1)} n)$ field operations) using Fast Fourier Transform based methods (cf. [Jus76]). Other algorithms with quadratic runtimes for decoding Reed-Solomon codes, and which involve no explicit "syndrome computation", are the Berlekamp-Welch algorithm [WB86] and Blahut's time-domain decoder [Bla83, Chapter 9].

However, these algorithms are all limited by the combinatorial barrier of half the distance of the code. Despite several years of research, there were no known efficient algorithms to correct significantly more errors by resorting to the list decoding approach. The only improvements over the algorithm of Peterson [Pet60] (in terms of number of errors corrected) were decoding algorithms due to Sidelnikov [Sid94] and Dumer [Dum89] which corrected $\frac{d}{2} + \Theta(\log n)$ errors in polynomial time. The first significant breakthrough in terms of errors corrected came when Sudan [Sud97a, Sud97b] gave an algorithm to correct (about) $n - \sqrt{2n(n-d)}$ errors. His algorithm improved over the classical $d/2$ bound for all rates less than 1/3, and for low-rates corrected almost twice as many errors than the previous algorithms. Sudan's algorithm builds upon ideas from earlier work by Ar, Lipton, Rubinfeld and Sudan [ALRS99], and an elegant presentation of the Berlekamp-Welch decoding algorithm due to Gemmell and Sudan [GS92]. (We should mention that it is non-trivial to ferret out this particular view of the Berlekamp-Welch algorithm from the original paper [WB86].) Despite the very good performance for low rates, for rates larger than 1/3, Sudan's algorithm did not give any improvement over the classical algorithms, and even for lower rates fell short of decoding up to the Johnson bound on list decoding radius.

Following the result of [Sud97a], Roth and Ruckenstein [RR00] investigated efficient implementations of the algorithm, and obtained a near-quadratic time bound (for decoding with constant-sized lists). Portions of this result were used in Section 6.2.7. Also, Shokrollahi and Wasserman [SW99] generalized the algorithm to algebraic-geometric codes. However, the error-correction capabilities of these algorithms were all limited to half the distance for large rates.

The decoding algorithm discussed in this chapter, which decodes both Reed-Solomon and algebraic-geometric codes up to the Johnson radius and hence beyond half the distance for every value of the rate, appears in [GS99]. This result sparked a renewed interest in decoding Reed-Solomon and AG-codes.

Several works investigated questions about the efficient implementation of the polynomial time algorithm for Reed-Solomon codes from [GS99]. Nielsen and Hϕholdt [NH99b] presented a fast implementation of the interpolation step of the decoding algorithm, which we referred to in Section 6.2.7, though they did not present an explicit runtime analysis. Independently, Olshevsky and Shokrollahi [OS99] gave efficient algorithms for the solving the interpolation step based on a general "displacement method" applied to find non-zero elements in the kernel of certain structured matrices. Gao and Shokrollahi [GS00] presented efficient algorithms for the root-finding step. The work of [RR00] and [GS00] both needed to find roots of univariate polynomials over $\mathbb{F}_q$ to solve the second step of the decoding algorithm. This was avoided by Augot and Pecquet [AP00], who presented an efficient implementation of the second step based on Hensel lifting. Their result gives the only known strongly polynomial time implementation of the list decoding algorithm, though it applies only to the earlier algorithm of Sudan [Sud97a], and not the general "multiplicity" based algorithm discussed in this chapter.

The soft (or weighted) decoding algorithm for Reed-Solomon codes, and its counterpart for AG-codes, have also sparked a lot of interest. Prior to the result of Theorem 6.21, the only general, provable soft decoding algorithms for Reed-Solomon codes appear to be the Generalized Minimum Distance (GMD) decoding algorithm due to Forney [For66a], which gave a (unique) soft decoding algorithm that worked under a fairly general condition, and the improvement by Berlekamp [Ber96] who gave a very efficient soft-decision decoding algorithm that expanded the error-correction radius by 1 (compared to GMD decoding). Koetter and Vardy [KV00] investigate the best setting of weights for use in the soft list decoding algorithm when decoding under fairly general probabilistic channels. The soft decoding algorithms are also exploited in decoding concatenated codes [GS00, Nie00, GHSZ00], and this will be discussed in detail in Chapter 8.

The unique decoding problem for AG-codes has been considered by several authors for over a decade. Some of the notable works are [JLJ$^+$89, SV90, Pel89, JLJH92, FR93], and these gave decoding algorithms to unambiguously decode an AG-code of designed distance d^* up to $\lfloor (d^* - 1 - r)/2 \rfloor$ errors where r is some integer between 0 and the genus g. The last of these works could in fact efficiently decode up to half the designed distance, i.e., up to $\lfloor (d^* - 1)/2 \rfloor$ errors. The first list decoding algorithm for AG-codes, that could decode well beyond half the designed distance at least for low rates, appeared in the work of Shokrollahi and Wasserman [SW99]. They generalized Sudan's list decoding algorithm for Reed-Solomon codes [Sud97a, Sud97b] to AG-codes. Their algorithm was improved in [GS99], and the resulting algorithm, which was discussed in this chapter, can decode beyond half the designed distance for every value of the rate.

The algorithm presented in [GS99] for AG-codes actually only gave a polynomial time reduction of the list decoding problem to certain algorithmic tasks over the underlying function field, including the task of finding roots of univariate polynomials over the function field. It was, however, not clear how to implement the corresponding steps efficiently for *every* function field.

Accordingly, the complexity of these steps has been studied by several authors, including Gao and Shokrollahi [GS00], Hϕholdt and Nielsen [NH99a], Wu and Siegel [WS00], and Augot and Pecquet [AP00]. However, none of the results provide a general polynomial time algorithm for all function fields. This is due to the following two reasons: (a) Either these algorithms work only for specific function fields; for example the algorithms in [NH99a] work for function fields of Hermitian curves, and those in [GS00] work for function fields of nonsingular plane algebraic curves, or (b) as in [AP00, WS00], the algorithms reduce the concerned questions to certain "more basic" algorithmic tasks on function fields, and it is not clear how to perform even these "basic" tasks efficiently for *every* function field.

The approach taken in this chapter was to build upon some of the above-mentioned works to show that there is a polynomial amount of precomputed information, given which we can have a completely general solution to the list decoding problem for AG-codes. This approach was taken in the paper [GS01b]. Independently of our work, Matsumoto [Mat99] also gave a completely general implementation of the list decoding algorithm for AG-codes discussed in Section 6.3.3, and his algorithms also require only a polynomial amount of precomputed information as advice.

Much of the technical content discussed in this chapter appears in the papers [GS99, GS01b], though our presentation here is a lot more integrated and elaborate in nature.

Chapter 7

A Unified Framework for List Decoding of Algebraic Codes

Be wise! Generalize!
Piccayune [sic] Sentinel

7.1 Introduction

In the previous chapter we presented list decoding algorithms for two widely-studied families of algebraic codes: Reed-Solomon codes and AG-codes. Owing to the importance of these codes, these results can be viewed as providing strong evidence to the general utility of list decoding as an algorithmic notion. Indeed, as we shall see in future chapters, they set the stage for a whole body of results about list decoding.

The reader might have already noticed a great deal of similarity between the general structure of the decoding algorithms for Reed-Solomon codes and AG-codes. Since Reed-Solomon codes are a special instance of AG-codes, the decoding algorithm for AG-codes is just a generalization of the Reed-Solomon decoding algorithm, and this should explain the great deal of similarity between the algorithms. In this chapter, we will present a further generalization of the decoding algorithm by presenting a unified algorithm for soft decoding a general family of algebraic codes (which we call *ideal-based codes*). The decoding algorithms for Reed-Solomon and AG-codes are then just special cases of this general paradigm. Such a unified framework for list decoding is important for two reasons. Firstly, such unifications are elegant and highlight the essence of the idea without any vagaries that might result from a specific situation. Secondly, it reduces the list decoding problem for specific instantiations of ideal-based codes, including the Reed-Solomon and AG-codes we studied in the previous chapter, to the efficient implementation of certain core algorithmic steps when applied to the specific context in question. To illustrate this point, after developing the general list decoding algorithm, we will apply it to a "new" situation, namely to list decoding Chinese Remainder codes (henceforth, CRT codes).

Recall that CRT codes, also called Redundant Residue codes, are the number-theoretic analogue of Reed-Solomon codes. They are defined by picking n relatively prime integers $p_1 < p_2 < \cdots < p_n$. The messages m of the code are integers in the range $0 \le m < \prod_{i=1}^{k} p_i$ for some k, $1 \le k < n$. A message m is encoded by its residues modulo all the p_i's, i.e., $m \mapsto \langle m \bmod p_1, m \bmod p_2, \ldots, m \bmod p_n \rangle$. By the Chinese Remainder theorem, the message m is uniquely specified by *any* k of its residues modulo $p_1, p_2, \ldots, p_n$, and hence the above forms a redundant encoding of the message m. Indeed, this argument shows that two codewords (corresponding to encodings of m_1, m_2 with $m_1 \neq m_2$) differ in at least $(n-k+1)$ positions. Hence, the distance of the code can be shown to equal $(n-k+1)$.

There has been a lot of interest in decoding CRT codes [Man76, Man78, GRS00, Bon00], but all these works fall short of list decoding CRT codes up to the Johnson radius, and in fact even fall short of decoding to half the minimum distance in general.[1]

Our general weighted list decoding algorithm for ideal-based codes, when applied to the case of CRT codes with a specific choice of weights (the exact choice ends up being a non-trivial guess), almost immediately gives an improvement to the prior results and decodes up to close to the Johnson bound. In fact, by choosing the parameters in the algorithm appropriately, the algorithm can decode up to the corresponding "weighted" Johnson bound (see Theorem 7.9) for *every* choice of weights. We also give a more efficient algorithm based on the Generalized Minimum Distance (GMD) decoding, to decode CRT codes up to half the minimum distance. GMD decoding was first discovered by Forney [For66b], who applied it to the soft decoding of Reed-Solomon codes.

We should mention here that by the very nature of the topic, the contents of this chapter are somewhat heavy on algebra. The results of this chapter put the algorithms from the previous chapter in a unified context and thus elucidate them better, but they are not necessary to the understanding of the results in the following chapters.

7.1.1 Overview

We begin in the next section by discussing the necessary preliminaries and terminology from commutative algebra concerning rings and ideals. These will be necessary for the definition of ideal-based codes and in the development of the list decoding algorithm for ideal-based codes. In Section 7.3 we give a formal definition of ideal-based codes and explain how Reed-Solomon codes, AG-codes and CRT codes can all be obtained as specific examples of ideal-based codes. In Section 7.4 we enlist some basic assumptions about the underlying rings and ideals, and prove the basic distance property of ideal-based codes. We add some further assumptions and develop a general weighted (soft) list decoding algorithm for ideal-based codes in Section 7.5. We then apply the results to the specific context of CRT codes in Section 7.6 and obtain a polynomial time soft decoding algorithm for CRT codes. We then apply it to specific interesting choices of weights to deduce results for CRT codes that decode up to the Johnson bound. Finally, in Section 7.7, we discuss the GMD decoding algorithm to decode CRT codes up to half the minimum distance.

7.2 Preliminaries

We quickly recall the basic algebraic definitions necessary for this chapter. If necessary, the reader can find further details and examples in any of the standard algebra texts (eg. [Art91]).

Rings: A *ring* is an algebraic structure $(R, +, \cdot)$ consisting of a set R together with two binary operations $(+, \cdot)$, normally called addition and multiplication respectively, which satisfy the following axioms:

[1]This limitation is for the Hamming metric of measuring distance between the received word and the codewords. Indeed, the result of [GRS00] provides a list decoding algorithm to decode up to the Johnson bound for a certain "natural" weighting of codeword positions of the CRT code.

- R is an abelian group under the operation $+$, with identity denoted by 0. This abelian group is denoted by R^+.
- R is closed under the operation $\cdot$, and $\forall\, x, y, z \in R$ we have
  - $x \cdot y = y \cdot x$ (Commutativity)
  - $x \cdot (y \cdot z) = (x \cdot y) \cdot z$ (Associativity)
  - $x \cdot (y + z) = x \cdot y + x \cdot z$ (Distributive property of $\cdot$ over $+$)
- There exists an identity element for multiplication, denoted by 1, which satisfies $1 \cdot x = x \cdot 1 = x$ for every $x \in R$.

The terminology relating to rings is not completely standardized. In some texts, rings are defined without the requirement of the commutativity of multiplication and/or the existence of the multiplicative identity 1. In their terminology, the above definition will correspond to a subclass of rings called *commutative rings with identity.* We will work exclusively with commutative rings with identity, and hence we included these axioms in our definition of rings.

A ring is said to be an *integral domain* if $a \cdot b = 0$ implies that either $a = 0$ or $b = 0$ or both. All rings we deal with will be integral domains.

A *field* is a ring together with the additional property that for every non-zero element $x \in R$, there exists a unique inverse $x^{-1} \in R$ such that $x \cdot x^{-1} = x^{-1} \cdot x = 1$. In other words, a field is a ring whose non-zero elements form an abelian group under the multiplication operation.

Ideals:

An *ideal* I of a ring R is, by definition, a subset of R with the following properties:

(i) I is a subgroup of R^+.

(ii) If $a \in I$ and $r \in R$, then $r \cdot a \in I$.

In any ring, the set of multiples of a particular element a forms an ideal called the *principal ideal* generated by a, and is denoted (a). The set consisting of 0 alone is always an ideal called the *zero ideal*, and is denoted (0). Likewise, the whole ring R is also an ideal (generated by the element 1), called the *unit ideal*, and is denoted (1).

One can define sum, product and intersection operations on ideals as follows. The intersection of ideals I, J is simply their intersection as subsets of R. The sum of I, J is defined as $I + J = \{a + b : a \in I \text{ and } b \in J\}$. The product of I and J, denoted $I \cdot J$ (or, IJ), is defined to be all finite linear combinations of the form $a_1b_1 + a_2b_2 + \ldots + a_mb_m$ where each $a_i \in I$ and each $b_i \in J$. In other words, IJ is the smallest ideal which contains all elements of the form ab where $a \in I$ and $b \in J$. It is easily checked that if I, J are ideals of R, then so are $I \cap J$, $I + J$ and IJ. Note that for every pair of ideals I and J, $IJ \subseteq I \cap J$. For an ideal I, the power ideal I^n, for $n \geq 1$, is defined in the obvious way as: $I^n = I$ if $n = 1$, and $I^n = I \cdot I^{n-1}$ if $n > 1$.

Quotient rings: Let I be an ideal of a ring R. Consider the relation on R defined by $a \sim b$ if $a - b \in I$. It is easily checked that $\sim$ is an equivalence relation, and therefore it partitions R into equivalence classes. These equivalence classes are called the *cosets* of the ideal I. For $a \in R$, we denote by a/I the coset to which a belongs. The set of cosets of I themselves form a ring, denoted R/I, by inheriting the addition and multiplication operations from R.

Specifically, one defines $(+,\cdot)$ for R/I by: $a/I + b/I \stackrel{\text{def}}{=} (a+b)/I$ and $a/I \ \cdot b/I \stackrel{\text{def}}{=} (a\cdot b)/I$. It is easy to check that these operations are well-defined and that R/I forms a ring under these operations. The ideals of R/I are in one-one correspondence with the ideals of R that contain I.

As an example, if $R = \mathbb{Z}$ and $I = (n)$ is the ideal generated by n, then $R/I = \mathbb{Z}/(n)$ is the ring of integers modulo n.

Prime and Maximal Ideals:

An ideal I of a ring R is a *prime ideal* if $a \cdot b \in I$ implies that at least one of a, b belongs to I. This is equivalent to the condition that the quotient ring R/I is an integral domain. The terminology "prime ideal" comes from the fact that if R is the ring of integers $\mathbb{Z}$ and $I = (m)$ is the ideal generated by an integer m, then I is a prime ideal if and only if m is a prime number.

An ideal I is a *maximal ideal* if $I \neq R$ and $I \not\subseteq J$ for any ideal $J \neq I, R$. An equivalent definition is that I is maximal iff the quotient ring R/I is a field.

Two ideals I, J of R are said to be *coprime* if $I + J = R$ (i.e., if $1 \in I + J$). The terminology comes from the fact that if the ring $R = \mathbb{Z}$ and $I = (m)$ and $J = (n)$ for integers m, n, then I, J are coprime ideals if and only if m, n are coprime integers. For coprime ideals I, J, we have $IJ = I \cap J$.[2]

7.3 Ideal-based codes

We now describe the basic principle that underlies the construction of several families of algebraic error-correcting codes, including Reed-Solomon codes, Algebraic-geometric codes, Chinese Remainder codes (and also Number field codes [Len86, Gur01c], though we do not discuss them in this thesis).

An algebraic error-correcting code is defined based on an underlying ring R (assume it is an integral domain), whose elements r come equipped with an appropriate notion of "size", denoted $\mathsf{size}(r)$. For example, for Reed-Solomon codes, the ring is the polynomial ring $\mathbb{F}[X]$ over a (large enough) finite field $\mathbb{F}$, and the "size" of $f \in \mathbb{F}[X]$ is related to its degree as a polynomial in X. Similarly, for the CRT code, the ring is $\mathbb{Z}$, and the "size" is the usual absolute value.

The messages of the code are the elements of the ring R whose size is at most a parameter B (this parameter governs the rate of the code). The encoding of a message $m \in R$ is given by

$$m \mapsto \mathsf{Enc}(m) = \langle m/I_1, m/I_2, \cdots, m/I_n \rangle \ ,$$

where I_j, $1 \le j \le n$ are n pairwise coprime ideals of R (we will assume that each of the quotient rings R/I_j is finite). Here m/I_j denotes the residue of m modulo the ideal I_j, and will belong to a finite alphabet whose size equals $|R/I_j|$. The formal definition follows:

Definition 7.1 *Let R be an integral domain; let $I_1, I_2, \ldots, I_n$ be n pairwise coprime ideals in R such that each R/I_j is finite, and let B be an arbitrary positive real. Further assume that there is a non-negative function* $\mathsf{size} : R \to \mathbb{R}^+$ *that associates a non-negative size*

[2]The easy proof of this fact goes as follows. Since $IJ \subseteq I \cap J$, we only have to prove that if $f \in I \cap J$ and $I + J = R$, then $f \in IJ$. Let $a \in I$ and $b \in J$ be such that $a + b = 1$. Now, $f = f \cdot (a+b) = f \cdot a + f \cdot b$. Now, clearly both $f \cdot a$ and $f \cdot b$ belong to IJ. Hence $f \in IJ$, as desired.

with each element of the ring R. Then, the "ideal-based" code $\mathbf{C}[R; I_1, I_2, \ldots, I_n; \mathsf{size}, B]$ *is defined to be the set of codewords*

$$\{\langle m/I_1, m/I_2, \ldots, m/I_n\rangle : m \in R \ \wedge \ \mathsf{size}(m) \leq B\} \tag{7.1}$$

7.3.1 Examples of ideal-based codes

Chinese Remainder codes (CRT codes): Taking $R = \mathbb{Z}$; $I_j = (p_j)$, the principal ideal generated by the n mutually coprime integers $p_1, p_2, \ldots, p_n$; and $\mathsf{size}(m) = |m|$, the absolute value of m, we get the definition of CRT codes from the above definition.

Reed-Solomon codes: We get the Reed-Solomon code from the above definition by taking $R = \mathbb{F}_q[X]$ where $\mathbb{F}_q$ is a finite field with at least n elements (i.e. $q \geq n$), and $I_j = (X - \alpha_j)$ — the ideal generated by the polynomial $(X - \alpha_j)$ — for $1 \leq j \leq n$, where $\alpha_1, \ldots, \alpha_n$ are *distinct* elements of $\mathbb{F}_q$. The notion of size is defined by $\mathsf{size}(p) = q^{\deg(p)}$. In other words, the messages are polynomials in $\mathbb{F}_q[X]$ of degree at most k, for some parameter k.

Algebraic-geometric codes: We now describe how the AG-codes from the previous chapter can also be obtained as a special case of ideal-based codes. Let $K/\mathbb{F}_q$ be a function field and P_0 be any fixed place of $K/\mathbb{F}_q$. For $i \geq 0$, let $\mathcal{L}(iP_0)$ be the set of functions in K which have no poles outside P_0 and have at most i poles at P_0. To specify an AG-code in the above ideal-theoretic language, we take the ring $R = \bigcup_{i \geq 0} \mathcal{L}(iP_0)$, and the ideal I_j to be a place P_j such that $P_1, P_2, \ldots, P_n$ and P_0 are all distinct places. (Recall from the previous chapter that a place P is by definition the unique maximal ideal of the ring $\mathcal{O}_P$ of regular functions at P, and since clearly $\mathcal{O}_P \subseteq R$ if $P \neq P_0$, such a place can also be viewed as an ideal of R.) The notion of size we use is related to the pole order at the place P_0; specifically we set $\mathsf{size}(x) = q^{-v_{P_0}(x)}$. Hence the set $\{x \in R : \mathsf{size}(x) \leq q^{\alpha}\}$ equals $\mathcal{L}(\alpha P_0)$, as with the usual definition of AG-codes.

7.4 Properties of Ideal-based codes

We now develop a set of axioms/assumptions about the ring R which will allow us to quantify the distance properties of the ideal-based code defined in Equation (7.1) above. We will later add a few further assumptions which will allow us to specify a unified list decoding algorithm for ideal-based codes and perform a quantitative analysis of its error-correction capabilities.

7.4.1 Axioms and assumptions

Let R be an integral domain (a commutative ring where $a \cdot b = 0$ implies either $a = 0$ or $b = 0$). We assume the following properties for the ring R:

1. [Size of Elements]: There exists a function $\mathsf{size} : R \to \mathbb{R}$ such that for all $x, y \in R$:
   - (S1) $\mathsf{size}(x) \geq 0$, and $\mathsf{size}(x) = 0 \Leftrightarrow x = 0$, and $\mathsf{size}(1) = \mathsf{size}(-1) = 1$.
   - (S2) There exists an integer $1 \leq a \leq 2$ such that $\mathsf{size}(x+y) \leq a \cdot \max\{\mathsf{size}(x), \mathsf{size}(y)\}$; in other words, size satisfies a certain kind of "triangle" inequality.[3]

[3]We point out that it is a well-known fact that if the stated inequality holds for some $a \leq 2$, then the "regular" archimedean triangle inequality $\mathsf{size}(x+y) \leq \mathsf{size}(x) + \mathsf{size}(y)$ also holds. Hence the name "triangle inequality" for this property.

(S3) $\mathsf{size}(xy) \le \mathsf{size}(x)\mathsf{size}(y)$

2. [Size of Ideals]: There exists a function Δ that maps each non-zero ideal I of R to a positive real number $\Delta(I)$ such that

   (I1) If x is a non-zero element of an ideal I, then $\Delta(I) \le \mathsf{size}(x)$.

   (I2) For every pair of coprime ideals I, J, $\Delta(IJ) \ge \Delta(I)\Delta(J)$.

The above axioms suffice to define a code and state the distance property that the code will satisfy.

7.4.2 Distance property of ideal-based codes

Lemma 7.1 *Assume that the assumptions (S1-S3) and (I1, I2) hold. Consider the code* $\mathbf{C}[R; I_1, \ldots, I_n; \mathsf{size}, B]$ *where the ring* R *satisfies the above assumptions (S1-S3) and (I1, I2). Assume further that the ideals* I_j *are ordered so that* $\Delta(I_1) \le \Delta(I_2) \le \cdots \le \Delta(I_n)$. *Then the minimum (Hamming) distance of this code is at least* $(n - t + 1)$ *where* t *is the smallest integer satisfying:*

$$\prod_{i=1}^{t} \Delta(I_i) > a \cdot B \ .$$

Proof: Let two distinct codewords in $\mathbf{C}$ corresponding to messages x, y agree on s residues, and let t be as in the statement of the lemma. We will show that $s < t$. Since $\mathsf{size}(x) \le B$ and $\mathsf{size}(y) \le B$, we have $\mathsf{size}(x - y) \le a \cdot B$ by axiom (S2). On the other hand, $(x - y)$ belongs to at least s ideals, and since the I_j's are pairwise coprime, $(x - y)$ belongs to the product of at least s ideals, say that of $I_{j_1}, \ldots, I_{j_s}$. Then, using axioms (I1) and (I2), we have

$$\mathsf{size}(x - y) \ge \Delta(\prod_{i=1}^{s} I_{j_i}) \ge \prod_{i=1}^{s} \Delta(I_{j_i}) \ge \prod_{i=1}^{s} \Delta(I_i) \ .$$

Together with $\mathsf{size}(x - y) \le aB$, this implies that

$$\prod_{i=1}^{s} \Delta(I_i) \le aB < \prod_{i=1}^{t} \Delta(I_i) \ ,$$

which shows that $s < t$ and completes the proof. □

To quantify the rate of these codes, we need a lower bound on the number of elements of R that have size at most B. We will later add axioms that guarantee this and further properties about the size of ideals that we will need to argue about the performance of our list decoding algorithm. We now turn to the specification of our list decoding algorithm.

7.5 List decoding ideal-based codes

We directly tackle the general "weighted" list decoding problem which is described below. We use the notation from the previous section and focus on list decoding an ideal-based code $\mathbf{C}[R; I_1 \ldots, I_n; \mathsf{size}, B]$ with message space $\mathcal{M} = \{x \in R : \mathsf{size}(x) \le B\}$.

Input: A vector $\mathbf{r} = \langle r_1, \ldots, r_n \rangle$ where $r_i \in R/I_i$ for $1 \le i \le n$, non-negative real weights $w_1, w_2, \ldots, w_n$, and agreement parameter W.

Required Output: A list of all $m \in \mathcal{M}$ such that $\sum_{i=1}^{n} w_i a_i > W$ where a_i is defined to be equal to 1 if $m/I_i = r_i$ and 0 otherwise.

To describe our list decoding algorithm, we assume the weights are some appropriate integers $z_1, z_2, \ldots, z_n$. Our algorithm will then output all codewords that satisfy a certain weighted condition in terms of the z_i's. The description of how to pick the z_i's to get useful results for specific input weights $w_1, w_2, \ldots, w_n$ will be described later when we apply the general algorithm to the case of the CRT code.

7.5.1 High level structure of the decoding algorithm

Before formally describing the algorithm, we first give some intuition on how it is designed based on the earlier Reed-Solomon decoding algorithm. Recall that our goal is to efficiently find a list of all $m \in R$ with $\mathsf{size}(m) \le B$ such that $\mathbf{C}(m)$ and the received word $\mathbf{r}$ have sufficient weighted agreement.

Following the Reed-Solomon and AG-codes case, the basic idea will be to "interpolate" a polynomial $c \in R[y]$ (based on the received word $\mathbf{r}$) with the property that every m for which $\mathbf{C}(m)$ has sufficient weighted agreement with the received word must be a *root* of the polynomial $c(y)$ (this polynomial c was called Q in the algorithms of the previous chapter). Then, by finding the roots of $c(y)$ and pruning out the spurious roots, we can recover all the codewords with sufficient weighted agreement with $\mathbf{r}$.

We are able to construct such a polynomial c by pursuing two objectives, which are in turn adaptations of the objectives from the case of decoding Reed-Solomon and AG-codes:

1. To ensure that the polynomial c has the property that for any $m \in R$ that satisfies $m/I_i = r_i$, we have $c(m) \in M_i$, for some suitable sequence of coprime ideals M_i, $i = 1, 2, \ldots, n$. This in turn implies that for any $m \in R$ we have $c(m) \in \prod_i M_i^{a_i}$, where $a_i = 1$ if $m/I_i = r_i$, and $a_i = 0$ otherwise.
2. To ensure that the coefficients c_j of $c(y) = \sum_{j=0}^{\ell} c_j y^j$ are small, i.e., each $\mathsf{size}(c_j)$ is sufficiently small. The aim of this step is to ensure that $\mathsf{size}(c(m))$ is small, say $\mathsf{size}(c(m)) < F$, for every m with $\mathsf{size}(m) \le B$.

By combining Objectives 1 and 2, we see that for any $m \in R$ with $\mathsf{size}(m) \le B$, $c(m)$ on the one hand has size less than F, and on the other hand belongs to $\prod_i M_i^{a_i}$. Hence if, $c(m) \neq 0$, we must have

$$F > \mathsf{size}(c(m)) \ge |R/M_i|^{a_i} \ , \tag{7.2}$$

where the second step uses axioms (I1), (I5). Therefore, if the boolean "agreement" vector $\mathbf{a} = \langle a_1, a_2, \ldots, a_n \rangle$ between $\mathbf{C}(m)$ and $\mathbf{r}$ satisfies the weighted condition

$$\sum_i a_i \log |R/M_i| > \log F \ ,$$

then Condition (7.2) cannot hold, and hence we must have $c(m) = 0$. Naturally, the performance of the algorithm depends on the choices of the ideals M_i and the parameter F. Our algorithm will pick $M_i = I_i^{z_i}$ (where z_i's are the input integer weights), and F to be a sufficiently large integer for which a polynomial $c \in R[y]$ meeting Objectives 1 and 2 exists. Precise details follow in the next section.

(Weighted) List-decoding algorithm:

Input: A vector $\mathbf{r} = \langle r_1, \ldots, r_n \rangle$ where $r_i \in R/I_i$ for $1 \le i \le n$, non-negative integers $z_1, z_2, \ldots, z_n$ and parameter Z.

Required Output: A list of all $m \in \mathcal{M}$ such that $\sum_{i=1}^{n} z_i a_i > Z$ (where a_i is defined to be equal to 1 if $m/I_i = r_i$ and 0 otherwise).

1. Pick parameters ℓ, F appropriately.
2. Find a non-zero polynomial $c \in \prod_{i=1}^{n} J_i^{z_i}$ of degree at most ℓ with the property that $\mathsf{size}(c(m)) \le F$ for every $m \in R$ with $\mathsf{size}(m) \le B$.
3. Find all roots of c that lie in R and report those roots ζ such that $\mathsf{size}(\zeta) \le B$ and the condition $\sum_{i=1}^{n} z_i a_i > Z$ is satisfied (where a_i is defined to be equal to 1 if $\zeta/I_i = r_i$ and 0 otherwise).

Figure 7-1: A general list decoding algorithm for ideal-based codes

7.5.2 Formal description of the decoding algorithm

Before describing the algorithm we need some auxiliary definitions and notation.

- Let $R[y]$ be the ring of polynomials in y with coefficients from R.
- For $1 \le i \le n$, let J_i be the ideal in $R[y]$ defined as $\{a(y)(y - r_i) + b(y) \cdot p | a, b \in R[y] \text{ and } p \in I_i\}$. It is readily checked that J_i is an ideal in $R[y]$ and further that if $m \in R$ satisfies $m/I_i = r_i$, then $c(m) \in I_i$ for every $c \in J_i$.

The algorithm is formally described in Figure 7-1. We stress that we do not know efficient implementations of all the steps in the algorithm for a general ideal-based code, but for specific codes like Reed-Solomon codes and AG-codes these do have efficient implementations. We will later show how with a moderate "slack" they can also be implemented in polynomial time for CRT codes.

7.5.3 Further assumptions on the underlying ring and ideals

In order to analyze the error-correction capability of the algorithm above, we add some further axiomatic assumptions. The following assumptions need to apply only to the ideals $I_1, \ldots, I_n$ specified in the construction of the code.

(I3) For each i, we have $\Delta(I_i^k) \ge \Delta(I_i)^k$ for all positive integers k.

(I4) For each i, we have that $|R/I_i^k| \le |R/I_i|^k$ for all positive integers k.

(I5) For each i, we have that $\Delta(I_i) \ge |R/I_i|$.

We also add the following assumption on the number of elements in R with bounded size. This is not only critical in order to quantify the rate of the code, but is also used in the analysis of the list decoding algorithm.

(S4) There exists a positive constant α depending only on the ring R such that for all positive integers F, the number of elements x of R with $\mathsf{size}(x) < F$ is at least αF.

Note that for the CRT code ($R = \mathbb{Z}$), we have $\alpha \simeq 2$, while for Reed-Solomon and AG-codes we have $\alpha = 1$.

7.5.4 Analysis of the list decoding algorithm

We now specify the parameter choices in the above algorithm for it to output all the "relevant" codewords, and determine the exact condition (specifically the value of the agreement parameter Z) for which the algorithm will succeed in finding all codewords that satisfy $\sum_i a_i z_i > Z$.

The following sequence of lemmas will be used in the analysis.

Lemma 7.2 *If $c \in J_i^{z_i}$, then for every $m \in R$ with $m/I_i = r_i$, we have $c(m) \in I_i^{z_i}$.*

Proof: Every $c \in J_i^{z_i}$ is the sum of a finite number of terms each of the form

$$\prod_{s=1}^{z_i} (a_s(y)(y - r_i) + b_s(y)p_s) \ ,$$

where $a_s, b_s \in R[y]$ and $p_s \in I_i$ for $1 \le s \le z_i$. Substituting $y = m$ where $m/I_i = r_i$, we have each of the s terms in the product belongs to I_i, and hence the entire term belongs to $I_i^{z_i}$. Since this is true for each term of $c(m)$, it follows that $c(m)$ itself is in $I_i^{z_i}$, as desired. □

Lemma 7.3 *For each i, $1 \le i \le n$, $|R[y]/J_i^{z_i}| \le |R/I_i|^{\binom{z_i+1}{2}}$.*

Proof: We need to estimate the number of different residues that polynomials in $R[y]$ can have modulo $J_i^{z_i}$. Let $c \in R[y]$ be any polynomial. Expand $c(y)$ in terms of sums of powers of $(y - r_i)$ (i.e., use the change of variable $y' = y - r_i$, and write down $c(y' + r_i)$). Since $(y - r_i)^m \in J_i^{z_i}$ for $m \ge z_i$, to compute the residue of c modulo $J_i^{z_i}$, we can ignore all terms of degree at least z_i. Thus we can assume that

$$c/J_i^{z_i} = \sum_{s=0}^{z_i - 1} \alpha_s (y - r_i)^s \ , \tag{7.3}$$

for suitable coefficients α_s. Now since $\alpha_s(y - r_i)^s \in J_i^{z_i}$ if $\alpha_s \in I_i^{z_i - s}$, it follows that we may assume that α_s is reduced modulo $I_i^{z_i - s}$ in the above, or in other words that $\alpha_s \in R/I_i^{z_i - s}$. Hence the number of possibilities for α_s is at most $|R/I_i^{z_i - s}| \le |R/I_i|^{z_i - s}$ using assumption (I4). Combining with Equation (7.3), we obtain that the total number of possible residues modulo $J_i^{z_i}$, in other words $|R[y]/J_i^{z_i}|$, is at most

$$\prod_{s=0}^{z_i - 1} |R/I_i|^{z_i - s} = |R/I_i|^{\binom{z_i+1}{2}} \ ,$$

as claimed. □

Corollary 7.4 *We have*

$$|R[y]/\prod_{i=1}^{n} J_i^{z_i}| \le \prod_{i=1}^{n} |R/I_i|^{\binom{z_i+1}{2}} .$$

Proof: First of all, note that since the I_i's are all coprime (i.e., $I_i + I_j = R$ for $i \neq j$), we also have the J_i's to be pairwise coprime. This in turn implies that the ideals $J_i^{z_i}$ are all pairwise coprime. Therefore,

$$|R[y]/\prod_{i=1}^{n} J_i^{z_i}| = \prod_{i=1}^{n} |R[y]/J_i^{z_i}| \le \prod_{i=1}^{n} |R/I_i|^{\binom{z_i+1}{2}}$$

where the second step follows from Lemma 7.3. □

Before stating the next lemma, we need the following notation. Let b_k be the least integer such that for all $x_1, x_2, \ldots x_k \in R$, we have $\mathsf{size}(x_1+x_2+\cdots+x_k) \le b_k \max\{\mathsf{size}(x_1), \ldots, \mathsf{size}(x_k)\}$. We clearly have $b_1 = 1$, $b_2 \le a$ (recall that a was the parameter used in the "triangle" inequality (S2)). Of course if $a = 1$, then each $b_k = 1$, and one can show that as long as $a \le 2$, $b_k \le k$. (This follows because it is a standard exercise to show that $a \le 2$ implies size satisfies the "familiar" triangle inequality $\mathsf{size}(x+y) \le \mathsf{size}(x) + \mathsf{size}(y)$, from which of course $b_k \le k$ follows easily.) Thus, for Reed-Solomon and algebraic-geometric codes, we have $b_k = 1$ for all $k \ge 1$, while for CRT codes we have $b_k = k$ for all $k \ge 1$.

Lemma 7.5 *For positive integers B, F', the number of polynomials $c \in R[y]$ of degree at most ℓ with the property that $\mathsf{size}(c(m)) < F'$ whenever $\mathsf{size}(m) \le B$ is at least*

$$\left(\frac{\alpha F'}{b_{\ell+1} B^{\ell/2}}\right)^{\ell+1} .$$

Proof: Consider polynomial $c(y) = c_0 + c_1 y + \ldots + c_\ell y^\ell$ where each $c_j \in R$ for $0 \le j \le \ell$. We will pick coefficients so that for any m with $\mathsf{size}(m) \le B$, we will have $\mathsf{size}(c_j m^j) < F'/b_{\ell+1}$. Note that this will imply that $\mathsf{size}(c(m)) < F'$ whenever $\mathsf{size}(m) \le B$. This requirement on c_j will be satisfied if $\mathsf{size}(c_j) < F' \cdot B^{-j}/b_{\ell+1}$ (here we are using (S3)). Also, by assumption (S4) there at least $\frac{\alpha F'}{B^j b_{\ell+1}}$ such choices for c_j. Hence the total number of polynomials $c \in R[y]$ with the required property is at least

$$\left(\frac{\alpha F'}{b_{\ell+1}}\right)^{\ell+1} \cdot \prod_{j=0}^{\ell} B^{-j} = \left(\frac{\alpha F'}{b_{\ell+1} B^{\ell/2}}\right)^{\ell+1} ,$$

as claimed. □

We are now ready to prove that for suitable choices of ℓ, F a non-zero polynomial with the desired properties as in Step 2 of the list decoding algorithm exists in $\prod_{i=1}^{n} J_i^{z_i}$.

Lemma 7.6 *Let ℓ, B, F be positive integers which satisfy the following condition:*

$$F \ge B^{\ell/2} \cdot \left(\frac{a \cdot b_{\ell+1}}{\alpha}\right) \left(\prod_{i=1}^{n} |R/I_i|^{\binom{z_i+1}{2}}\right)^{1/(\ell+1)} . \tag{7.4}$$

Then there exists a non-zero $c \in \prod_{i=1}^{n} J_i^{z_i}$ which satisfies the property that $\mathsf{size}(c(m)) < F$ for every $m \in R$ with $\mathsf{size}(m) \le B$.

Proof: The proof follows from Corollary 7.4, Lemma 7.5, and the pigeonhole principle. Specifically, if Condition (7.4) is satisfied, then we have

$$\left(\frac{\alpha \cdot F/a}{b_{\ell+1}B^{\ell/2}}\right)^{\ell+1} > \prod_{i=1}^{n} |R/I_i|^{\binom{z_i+1}{2}} ,$$

and thus the number of degree ℓ polynomials $c \in R[y]$ with $\mathsf{size}(c(m)) < F/a$ whenever $\mathsf{size}(m) \leq B$ is greater than the total number of residues of polynomials modulo $\prod_{i=1}^{n} J_i^{z_i}$. Hence, by the pigeonhole principle there must exist two distinct polynomials $c_1, c_2 \in R[y]$ of degree at most ℓ such that $(c_1 - c_2) \in \prod_{i=1}^{n} J_i^{z_i}$. Since $\mathsf{size}(c_1(m)) < F/a$ and $\mathsf{size}(c_2(m)) < F/a$ for every m with $\mathsf{size}(m) \leq B$, we have by assumption (S2) that $\mathsf{size}((c_1 - c_2)(m)) < F$ for each such m. Thus the claim of the lemma is satisfied with $c \stackrel{\text{def}}{=} (c_1 - c_2)$. □

Lemma 7.7 *Let $c \in \prod_{i=1}^{z_i} J_i^{z_i}$ be such that $\mathsf{size}(c(x)) < F$ for every $x \in R$ with $\mathsf{size}(x) \leq B$. Then, any $m \in R$ that satisfies $\mathsf{size}(m) \leq B$ and*

$$\prod_{i:m/I_i=r_i} |R/I_i|^{z_i} \geq F \tag{7.5}$$

must be a root of c, i.e., must satisfy $c(m) = 0$.

Proof: Let m be any such element of R. Since $\mathsf{size}(m) \leq B$, by the property of c, we have

$$\mathsf{size}(c(m)) < F \ . \tag{7.6}$$

Since $c \in J_i^{z_i}$ for each i, $1 \leq i \leq n$, by Lemma 7.2, we have $c(m) \in I_i^{z_i}$ for each i such that $m/I_i = r_i$. Hence

$$c(m) \in \prod_{i:m/I_i=r_i} I_i^{z_i} \ .$$

Now, using assumptions (I1), (I2), (I3) and (I5), we have that if $c(m) \neq 0$, then

$$\mathsf{size}(c(m)) \geq \prod_{i:m/I_i=r_i} \Delta(I_i^{z_i}) \geq \prod_{i:m/I_i=r_i} \Delta(I_i)^{z_i} \geq \prod_{i:m/I_i=r_i} |R/I_i|^{z_i} \ . \tag{7.7}$$

From (7.7) and (7.6) it follows that if Condition (7.5) is satisfied, we have a contradiction and therefore must have $c(m) = 0$, as desired. □

7.5.5 Performance of the list decoding algorithm

We are now ready to state and prove the main result of this section on the performance of our list decoding algorithm from Section 7.5.1.

Theorem 7.8 *For every set of non-negative integers $z_1, z_2, \ldots, z_n$, for a suitable choice of parameters ℓ, F, the list decoding algorithm on receiving as input a word $\mathbf{r} = \langle r_1, \ldots, r_n \rangle$ with $r_i \in R/I_i$, can find a list of size at most ℓ which includes all messages $m \in R$ with $\mathsf{size}(m) \leq B$ that satisfy*

$$\sum_{i=1}^{n} a_i z_i \log q_i > \frac{1}{\ell+1} \sum_{i=1}^{n} \binom{z_i+1}{2} \log q_i + \log(a/\alpha) + \frac{\ell}{2} \log B + \log b_{\ell+1} \ . \tag{7.8}$$

where we use the shorthand $q_i = |R/I_i|$, and a_i is an indicator variable defined to be 1 if $m/I_i = r_i$ and 0 otherwise.

Proof: The proof follows easily from the statements of Lemma 7.6 and Lemma 7.7. Indeed, one can choose

$$F = \left\lceil B^{\ell/2} \cdot \Big(\frac{a \cdot b_{\ell+1}}{\alpha}\Big)\Big(\prod_{i=1}^{n} |R/I_i|^{\binom{z_i+1}{2}}\Big)^{1/(\ell+1)} \right\rceil , \tag{7.9}$$

and for this choice of F, the algorithm can find a non-zero $c \in \prod_{i=1}^{n} J_i^{z_i}$ with $\mathsf{size}(c(m)) < F$ whenever $\mathsf{size}(m) \le B$ (since by Lemma 7.6 such a c exists). By Lemma 7.7, the algorithm will output a list of all $m \in R$ with $\mathsf{size}(m) \le B$ such that

$$\prod_{i=1}^{n} q_i^{a_i z_i} \ge F \ .$$

Note that the number of solutions the algorithm outputs is at most ℓ, since it only outputs a subset of the roots of a degree ℓ polynomial over the integers. Also, since both the terms on the right and left hand sides of the above condition are integers, taking logarithms we note that the above condition is implied by the decoding Condition (7.8) stated in the theorem. □

Remark: There is a natural notion of an "approximate solution" for Step 2 in the list decoding algorithm. We know that for F defined as in Equation (7.9), there exists a non-zero polynomial $c \in \prod_{i=1}^{n} J_i^{z_i}$ that satisfies $\mathsf{size}(c(m)) < F$ whenever $\mathsf{size}(m) \le B$. It is conceivable that, in certain contexts, finding such a c for this "optimum" choice of F might be difficult to accomplish efficiently. In such a case, suppose the algorithm only manages to find a non-zero polynomial $c \in \prod_{i=1}^{n} J_i^{z_i}$ with a factor β slack in the size guarantee, namely a polynomial c such that $\mathsf{size}(c(m)) < F'$ for every m with $\mathsf{size}(m) \le B$, where $F' = \beta F$. Then, it is easy to check that such an algorithm can decode under a condition similar to (7.8) with an additional $\log \beta$ term on the right hand side. We will make use of this fact when considering an efficient implementation of the decoding algorithm for the specific context of decoding CRT codes in Section 7.6.2.

7.5.6 Obtaining algorithms for Reed-Solomon and AG-codes

We now briefly indicate how the Reed-Solomon and AG-code list decoding algorithms from the previous chapter can be obtained from Theorem 7.8 above. Note that the list decoding algorithm of Figure 7-1 is really only a general algorithmic schema, and one needs to implement each of its steps efficiently in order to apply it and get polynomial time list decoding algorithms for specific families of ideal-based codes. Hence, our aim below is only to show that this algorithm gives (more or less) the same parameters as the specific polynomial time algorithms discussed in the previous chapter.

For Reed-Solomon codes over a field $\mathbb{F}_q$, each $q_i = q$, $\alpha = 1$, and $a = 1$ in assumption (S2) (and hence $b_{\ell+1} = 1$ as well). If the code is defined by evaluations of polynomials of degree at most k, then since $\mathsf{size}(p) = q^{\deg(p)}$, we have $B = q^k$. Substituting these we get the algorithm finds all codewords that have "z-weighted" agreement with $\mathbf{r}$ more than

$$\frac{1}{\ell+1}\sum_{i=1}^{n}\binom{z_i+1}{2} + \frac{\ell}{2}k$$

which for large ℓ, is approximately $\sqrt{k\sum_i z_i(z_i+1)}$ which approaches the performance of the soft decoding algorithm for Reed-Solomon codes from Section 6.2.10 (by taking the z_i's to be large multiples of the weights w_i).

For AG-codes over $\mathbb{F}_q$, once again each $q_i = q$, $a = 1$ and $b_{\ell+1} = 1$. If the underlying function field has genus g, then $\alpha = q^{-g}$. Also, $B = q^{\alpha^*}$ if the message space of the AG-code is $\mathcal{L}(\alpha^* P_0)$. Hence Theorem 7.8 implies that one can find all codewords that have "z-weighted" agreement with $\mathbf{r}$ more than

$$\frac{1}{\ell+1}\sum_{i=1}^{n}\binom{z_i+1}{2} + \frac{\ell}{2}\alpha^* + g$$

which for large ℓ again approaches the performance of the soft decoding algorithms for AG-codes from Chapter 6.

We already knew the decoding algorithms for Reed-Solomon codes and AG-codes from the previous chapter, but the above indicates the generality of our decoding algorithm for ideal-based codes. In the next section, we will exploit the generality of our algorithm from Figure 7-1 to devise a list decoding algorithm for Chinese Remainder (CRT) codes. Indeed, it was the design of a good decoding algorithm for CRT codes that motivated us to dig deeper into the algebra underlying the list decoding algorithms and unveil the unified decoding algorithm for ideal-based codes described in Figure 7-1.

7.6 Decoding algorithms for CRT codes

In this section, we discuss efficient decoding algorithms for the CRT code. Recall that a CRT code is specified by a sequence $p_1 < p_2 < \ldots < p_n$ of relatively prime integers and an integer $k < n$. Let $K = \prod_{i=1}^{k} p_i$; $N = \prod_{i=1}^{n} p_i$. For easy reference, we say such a CRT codes as being specified by parameters $(p_1, p_2, \ldots, p_n; K)$. We associate to each integer $m \in \{0, 1, \ldots, K-1\}$ the codeword $\langle m_1, m_2, \ldots, m_n\rangle$, where $m_i = m \bmod p_i$. We will abuse notation and refer to both this sequence and m as a *codeword*. We consider a *received word* to be a sequence $\mathbf{r} = \langle r_1, r_2, \ldots, r_n\rangle$ of integers with $0 \le r_i < p_i$ for each $i \in [n]$. For a given sequence of weights $\vec{w} = \langle w_1, \ldots, w_n\rangle$, the *$\vec{w}$-weighted agreement* (or simply *weighted agreement*, when the weighting we are referring to is clear) between a codeword $m < K$ and a received word $\mathbf{r}$ is the defined to be the quantity $\sum_i a_i w_i$, where $a_i = 1$ if $m_i = r_i$, and $a_i = 0$ otherwise.

Our goal in this section is to efficiently find a list of all non-negative integers $m < K$ such that the encoding of m and the received word $\mathbf{r}$ have sufficient weighted agreement. We note that a simple transformation makes it equivalent for us to find integers m where $|m| \le K/2$, whose encodings have sufficient agreement with $\mathbf{r}$. It is this version of the problem that we focus on for describing our decoding algorithms.

In this section, we present two efficient decoding algorithms for the CRT code. In the first (which is our main) decoding algorithm, the goal is to efficiently find a list of *all* codewords m such that m and the received word $\mathbf{r}$ have sufficient weighted agreement. In particular, we are able to give an efficient list decoding algorithm which outputs all m with $|m| \le K/2$ such that $m \bmod p_i = r_i$ for at least $\sqrt{k(n+\varepsilon)}$ values of i (for any ε, with the running time of the algorithm depending polynomially on $1/\varepsilon$). Thus, we are able to efficiently list decode the CRT code up to (essentially) the Johnson bound on list decoding radius (from Corollary 3.3 with distance $d = n - k + 1$). This improves the earlier works of [GRS00, Bon00] which could only find the codewords which agreed with the received word in at least $\Omega(\sqrt{kn \log p_n / \log p_1})$ positions. Our algorithm is obtained by efficient implementations of the steps of the general decoding algorithm of Figure 7-1, specialized for the case of the CRT code. This gives a general weighted decoding algorithm which successfully list decodes as long as a certain "weighted" condition is satisfied. The above claimed bound is then obtained by an appropriate choice of weights in the weighted algorithm (the exact setting of weights turns out to be a non-trivial guess).

For any sequence of positive weights $\vec{\beta}$, our second decoding algorithm efficiently (in near-quadratic time) recovers the *unique* codeword m with highest $\vec{\beta}$-weighted agreement with a received word r, as long as there is a codeword whose $\vec{\beta}$-weighted distance from r is less than half the $\vec{\beta}$-weighted minimum distance of the code. This is accomplished by adapting the GMD decoding algorithm due to Forney, introduced for Reed-Solomon codes in [For66b], to CRT codes in Section 7.7. Note that in particular this result gives the *first* polynomial time algorithm to correct up to $(n-k)/2$ errors (i.e., decode up to half the minimum distance) for the CRT code. In view of our more powerful list decoding algorithm, the main role of this result can be viewed as highlighting the role of GMD decoding in the task of decoding the CRT code, plus achieving a simpler, faster algorithm for unique decoding of CRT codes.

7.6.1 Combinatorial Bounds on List Decoding

Before delving into the decoding algorithms, we first state a generalized Johnson-type bound which specifies a fairly general condition under which list decoding using "small" lists can be performed. This result will indicate the kind of performance that we can hope for from our list decoding algorithms for the CRT codes, since in order to efficiently output a list of codewords as possible answers, we need an a priori guarantee that the list size will be small.

The result below gives a generalization of the weighted Johnson bound from Chapter 3 (specifically the result of Corollary 3.7) to the case when the various codeword positions have different contributions towards the distance of the code.

Theorem 7.9 *Let C be a code of blocklength n with the i'th symbol coming from an alphabet of size q_i, for $1 \le i \le n$. Let the distance D_α of the code be measured according to a weighting vector $\vec{\alpha} = \langle \alpha_1, \ldots, \alpha_n \rangle$. In other words, for any two distinct codewords $c_1, c_2 \in C$, we have $\sum_{i: c_{1i} \ne c_{2i}} \alpha_i \ge D_\alpha$ (assume each $\alpha_i \ge 1$ without loss of generality). For a weighting vector $\vec{\beta} = \langle \beta_1, \ldots, \beta_n \rangle$ and a received word $\mathbf{r} = \langle r_1, \ldots, r_n \rangle \in [q_1] \times \cdots \times [q_n]$, define the set $S_{\vec{\beta}}(\mathbf{r}, W)$ to consist of all strings z (in the space $[q_1] \times [q_2] \times \cdots \times [q_n]$) with weighted $\vec{\beta}$-weighted agreement with $\mathbf{r}$ at least W, i.e., which satisfy $\sum_{i: r_i = z_i} \beta_i \ge W$. Then, for all $\mathbf{r}$, the set $S_{\vec{\beta}}(\mathbf{r}, W_\beta)$ has at most $(2 \sum_{i=1}^n q_i)$ codewords from C, provided that:*

$$W_\beta \ge \left[\Big(\sum_{i=1}^n \alpha_i - D_\alpha \Big) \sum_{i=1}^n \frac{\beta_i^2}{\alpha_i} \right]^{1/2} , \tag{7.10}$$

and has at most L codewords from C, provided that

$$W_\beta \ge \left[\Big(\sum_{i=1}^n \alpha_i - D_\alpha + \frac{D_\alpha}{L} \Big) \sum_{i=1}^n \frac{\beta_i^2}{\alpha_i} \right]^{1/2} . \tag{7.11}$$

Remark: A more complicated and stronger bound than the above theorem can be proved by taking into account the size of the alphabets q_i's (akin to the weighted Johnson bound of Theorem 3.6 that took into account the alphabet size). This is, however, not very important for us since we want to use the above bound to only informally indicate the "near-tightness" of the error-correction performance of our list decoding algorithms for the CRT code, and the above bound suffices for this purpose. Moreover, for the CRT code the q_i's are typically large primes, and for large alphabets the difference between the stronger bound and the above bound becomes negligible.

Proof of Theorem 7.9: The proof follows along the lines of Theorem 3.1. Let $\vec{\beta}$ be a weighting vector and W an agreement parameter. Let $\mathbf{c_1}, \ldots, \mathbf{c_m}$ be *all* the codewords in $S_{\vec{\beta}}(\mathbf{r}, W_\beta)$, where $\mathbf{r} \in [q_1] \times \cdots \times [q_n]$ is the "received word".

We will embed elements of $[q_1] \times \cdots \times [q_n]$ as vectors in $\mathbb{R}^Q$ where $Q = \sum_{i=1}^n q_i$, with the i'th block being a vector of length q_i corresponding to the i'th symbol. For the received word $\mathbf{r}$, we will let the i'th block (which is of length q_i) have a value of $\beta_i/\sqrt{\alpha_i}$ at position number r_i, and 0's elsewhere. By abuse of notation, we denote the resulting vector in $\mathbb{R}^Q$ also by $\mathbf{r}$. For each of the codewords $\mathbf{c_j}$, $1 \le j \le m$, we will let the i'th block have a value of $\sqrt{\alpha_i}$ at position number $c_{j,i}$ (i.e., the i'th symbol of the codeword $\mathbf{c_j}$), and 0's elsewhere. Once again, since it is convenient to do so, we denote the resulting vectors in $\mathbb{R}^Q$ also by $\mathbf{c_1}, \ldots, \mathbf{c_m}$.

It is easy to see from the above choices that $\langle \mathbf{c_j}, \mathbf{r} \rangle$ equals the $\vec{\beta}$-weighted agreement between $\mathbf{c_j}$ and $\mathbf{r}$ (i.e., $\langle \mathbf{c_j}, \mathbf{r} \rangle = \sum_{i: c_{j,i} = r_i} \beta_i$), and that $\langle \mathbf{c_j}, \mathbf{c_k} \rangle$ equals the $\vec{\alpha}$-weighted agreement between $\mathbf{c_j}$ and $\mathbf{c_k}$. Therefore, we have, for every $1 \le j < k \le m$,

$$\langle \mathbf{c_j}, \mathbf{r} \rangle \;\ge\; W_\beta \tag{7.12}$$

$$\langle \mathbf{c_j}, \mathbf{c_k} \rangle \;\le\; A_\alpha \stackrel{\text{def}}{=} \Big(\sum_{i=1}^n \alpha_i - D_\alpha \Big) \tag{7.13}$$

The idea now is to pick a suitable parameter $\gamma > 0$ such that the pairwise dot products between the vectors $(\mathbf{c_j} - \gamma \mathbf{r})$ are all non-positive. This is similar to the idea used in the proof of Theorem 3.1, and the details are in fact simpler in this case (since the conditions under which we want to show a small list size do not depend on the alphabet sizes q_i).

Equations (7.12) and (7.13) together with the facts that $\langle \mathbf{r}, \mathbf{r} \rangle = \sum_{i=1}^n \beta_i^2/\alpha_i$, implies, for $j \ne k$,

$$\langle \mathbf{c_j} - \gamma \mathbf{r}, \mathbf{c_k} - \gamma \mathbf{r} \rangle \le A_\alpha - 2\gamma W_\beta + \gamma^2 \sum_{i=1}^n \frac{\beta_i^2}{\alpha_i} \ . \tag{7.14}$$

We will therefore have $\langle \mathbf{c_j} - \gamma \mathbf{r}, \mathbf{c_k} - \gamma \mathbf{r} \rangle \le 0$, provided

$$W_\beta \ge \frac{\gamma}{2} \sum_{i=1}^n \frac{\beta_i^2}{\alpha_i} + \frac{A_\alpha}{2\gamma} \ . \tag{7.15}$$

The right hand side is minimized for $\gamma = (A_\alpha)^{1/2} \cdot (\sum_i \frac{\beta_i^2}{\alpha_i})^{-1/2}$, and for this choice of γ, Condition (7.15) becomes

$$W_\beta \ge \left[A_\alpha \sum_{i=1}^n \frac{\beta_i^2}{\alpha_i} \right]^{1/2} \ . \tag{7.16}$$

Now appealing to the geometric Lemma 3.4, Part (i), we get that the number of codewords m is at most $2Q$ (since the pairwise dot products of the Q-dimensional real vectors $(\mathbf{c_j} - \gamma \mathbf{r})$ are all non-positive). Thus, the number of codewords which lie in $S_{\vec{\beta}}(\mathbf{r}, W_\beta)$ if Condition (7.16) holds is at most $2Q$, which proves the first assertion of the theorem. The second assertion also follows similarly, by picking the parameter γ such that the pairwise dot products of the unit vectors along $(\mathbf{c_j} - \gamma\mathbf{r})$ is at most $-1/(L-1)$, and then appealing to geometric Lemma 3.5. We omit the details. $\square$

We now apply the result of Theorem 7.9 to specific choices of $\vec{\alpha}$ and $\vec{\beta}$ for the CRT code. For a CRT code we have, in the ideal-based language, $R = \mathbb{Z}$ and $I_i = (p_i)$ for relatively prime integers $p_1 < p_2 < \cdots < p_n$. Hence $q_i = p_i$ for $1 \le i \le n$. Furthermore the "size" of the ideals satisfy $\Delta(I_i) = |R/I_i| = p_i$. If we consider messages to be integers m in the range $-K/2 < m \le K/2$ where $K = \prod_{i=1}^{k} p_i$, then it is easy to prove using ideas in Lemma 7.1 that the $\vec{\alpha}$-weighted distance of the CRT code is

- at least $(n-k+1)$ for the all-ones weight vector (the case when $\alpha_i = 1$ for every i), and

- at least $\log(N/K)$ for case when $\alpha_i = \log p_i$.

Using these in the bound of Theorem 7.9 we get (roughly) the following conditions under which CRT list decoding is feasible (combinatorially):

$$\sum_i a_i \log p_i \;>\; \sqrt{(\log K + \varepsilon)\log N} \tag{7.17}$$

$$\sum_i a_i \;>\; \sqrt{(k+\varepsilon)n} \tag{7.18}$$

$$\sum_i a_i\beta_i \;>\; \left((\log K + \varepsilon)\sum_i \frac{\beta_i^2}{\log p_i}\right)^{1/2} \tag{7.19}$$

(Note that the third condition above implies the first with the choice of weights $\beta_i = \log p_i$.) In fact, the algorithm of Goldreich, Ron, and Sudan [GRS00] could list decode under the first condition (7.17). In some sense the case $\alpha_i = \beta_i$ is the most natural one for the CRT code. However, neither the algorithm of [GRS00] nor the improvement due to Boneh [Bon00], could work as well for other weightings (including the case $\beta_i = \alpha_i = 1$ from the second condition above). In the next section, we apply our general decoding algorithm for ideal-based codes to the case of CRT codes to remedy this defect of earlier algorithms, and give a weighted list decoding algorithm which, by appropriately choosing the weights, can decode under each of the above conditions.

7.6.2 Weighted list decoding algorithm

We now apply Theorem 7.8 to the case of CRT codes and get the following.

Theorem 7.10 *For a CRT code with parameters $(p_1, p_2, \ldots, p_n; K)$, given a received word $\mathbf{r} = (r_1, r_2, \ldots, r_n)$ with $0 \le r_i < p_i$, and non-negative integers ℓ and z_i for $1 \le i \le n$, we can find in time polynomial in $n, \ell, \sum_i \log p_i$ and $\sum_i z_i$, a list of size at most ℓ which includes all codewords m that satisfy*

$$\sum_{i=1}^{n} a_i z_i \log p_i > \log(\ell+1) + \frac{\ell}{2}\log K + \frac{1}{\ell+1}\sum_{i=1}^{n}\binom{z_i+1}{2}\log p_i \ , \tag{7.20}$$

where as usual we define $a_i = 1$ if $m_i = r_i$ and $a_i = 0$ otherwise.

Proof: We will show that the above condition implies the condition under which the decoding algorithm of Figure 7-1, when applied to the CRT case, successfully list decodes the received word. It will then remain to argue that for the CRT code each of the steps of the algorithm can be implemented in polynomial time.

For the CRT code, we have $|R/I_i| = |\mathbb{Z}/(p_i)| = p_i$ for $1 \le i \le n$, $a = 2$, and $b_{\ell+1} = \ell+1$ (since the integers satisfy the "familiar" archimedean triangle inequality). Furthermore, since there are $(2F-1)$ integers of absolute value less than F for any positive integer F, we can assume that $\alpha \ge (2-\gamma)$ for some small $\gamma > 0$ (in fact we can take $\gamma = o(1)$ in the parameters involved). Also since the messages are integers of absolute value at most $K/2$, we have $B = K/2$. Plugging these parameters into the general bound of Equation (7.8) we get the condition

$$\sum_{i=1}^{n} a_i z_i \log p_i > \log(\ell+1) + \frac{\ell}{2}\log(K/2) + \frac{1}{\ell+1}\sum_{i=1}^{n}\binom{z_i+1}{2}\log p_i + \log(2/(2-\gamma)) \ . \quad (7.21)$$

Note that in fact the above condition poses a weaker requirement than that of Condition 7.20 stated in the theorem, since we have an $\frac{\ell}{2}\log(K/2)$ term on the right hand side instead of $\frac{\ell}{2}\log K$ as stated in the theorem — the additional $\log(2/(2-\gamma))$ term is of course negligible in comparison. Hence the general algorithm can also decode under the condition stated in the theorem. The reason for the slack in Condition (7.20) is that we now also want a polynomial time implementation of its various steps, and hence can only find an "approximation" to the best polynomial $c \in \mathbb{Z}[y]$ in Step 2 of the algorithm. As discussed in the remark following Theorem 7.8, this necessitates a slight weakening of the error-correction performance. We discuss the details next.

The two non-trivial steps in the algorithm of Figure 7-1, when applied to the CRT code, are (i) finding a non-zero degree ℓ polynomial c with integer coefficients in the ideal $\prod_i J_i^{z_i}$ such that $|c(m)| < F$ for all m with $|m| \le K/2$, and (ii) finding the roots of c and looking for candidate codewords among its roots. The second task can be done in polynomial time using, for instance, the algorithm for factoring polynomials with integer coefficients due to Lenstra, Lenstra and Lovász [LLL82].[4] For the first task, Lemma 7.6 applied to the CRT case implies that for

$$F = \lceil F^* \rceil \quad \text{where} \quad F^* \stackrel{\text{def}}{=} (\ell+1)(K/2)^{\ell/2}\Big(\prod_i p_i^{\binom{z_i+1}{2}}\Big)^{1/(\ell+1)} ,$$

there exists a non-zero $c \in \prod_i J_i^{z_i}$ with $|c(m)| < F$ whenever $|m| \le K/2$ (in fact the coefficients c_j of c will satisfy $|c_j| < \frac{F}{(\ell+1)(K/2)^j}$ for $0 \le j \le \ell$). We will now prove that for F' which $2^{\ell/2}$ times larger than F, we can find, *in polynomial time*, a $c \in \mathbb{Z}[y]$ that satisfies $|c(m)| < F'$ for every m with $|m| \le K/2$. We do this by reducing this problem to that of finding a short lattice vector in a suitably defined lattice, and then appealing to the well-known approximate shortest lattice vector algorithms due to [LLL82].

We can view degree ℓ polynomials as vectors in $\mathbb{Z}^{\ell+1}$ in the obvious way. Note that the ideal $J = \prod_i J_i^{z_i}$, when restricted to polynomials of degree at most ℓ, can be viewed as an integer lattice, say L, of dimension $(\ell+1)$. Therefore, finding a suitable non-zero polynomial $c \in J$ with small coefficients amounts to finding a short non-zero lattice vector

[4] Since the root finding task is easier than a general factorization task, there are faster ways to solve the root finding problem. A brief discussion about this appears in [GRS00].

in L. This can be accomplished using the LLL algorithm, provided we can compute a basis for the lattice L. We now demonstrate how this can be done efficiently.

Note that $L = \cap_i L_i$ where L_i is the lattice corresponding to degree ℓ polynomials in $J_i^{z_i}$, for $1 \le i \le n$. Explicit bases for the individual lattices L_i are easily obtained by considering the generating polynomials for $J_i^{z_i}$ restricted to polynomials of degree at most ℓ. Let $\tilde{z}_i = \min\{z_i, \ell\}$. The first $\tilde{z}_i + 1$ vectors in our basis correspond to the generating polynomials $\{p_i^{(z_i-a)}(y-r_i)^a : 0 \le a \le \tilde{z}_i\}$ from the ideal $I_i^{z_i}$. For example, corresponding to $p_i^{z_i-2}(y-r_i)^2$, we add the vector $(r_i^2 \cdot p_i^{z_i-2}, \quad -2r_i \cdot p_i^{z_i-2}, \quad p_i^{z_i-2}, \quad 0, \quad \ldots, \quad 0)$. If $\ell > z_i$, then we also add vectors corresponding to the polynomials $\{y^a \cdot (y-r_i)^{z_i}\}_{a=1}^{\ell-z_i}$. Let $M^{(i)}$ be the $(\ell+1)$ by $(\ell+1)$ matrix whose rows are the vectors from this basis. It is straightforward to check that the integer linear combinations of these vectors correspond exactly to the set of polynomials of degree at most ℓ in the ideal $J_i^{z_i}$.

Thus bases for each L_i can be computed efficiently. Using standard techniques (see the discussion immediately following this proof), given bases for the (full-dimensional) lattices L_i, a basis B for the intersection lattice $L = \cap_i L_i$ can be computed in polynomial time.

With this basis in hand, our goal is to find a short vector in L (intuitively, short vectors in the lattice L correspond to polynomials in $\prod_i J_i^{z_i}$ with small coefficients). We argued earlier that there exists a vector $\mathbf{c} = (c_0, c_1, \ldots, c_\ell) \in L$ with $|c_j| \le \frac{F}{(\ell+1)(K/2)^j}$, and we would like to find a vector in L with components not much bigger than this. To do so, it is convenient to work with a re-scaled version L' of the lattice L where $(v_0, v_1, \ldots, v_\ell) \in L$ iff $(v_0, v_1 \cdot (K/2), \ldots, v_\ell \cdot (K/2)^\ell) \in L'$. The vector corresponding to $\mathbf{c}$ in L' has L_2-norm less than $F/\sqrt{\ell+1}$. Applying the LLL algorithm to the $(\ell+1)$-dimensional lattice L', we can therefore find a non-zero vector $\mathbf{w} = (w_0, \ldots, w_\ell) \in L'$ with L_2-norm $\|\mathbf{w}\|_2 < 2^{\ell/2} F/\sqrt{\ell+1}$ in polynomial time. By Cauchy-Schwartz, we have that the L_1-norm of $\mathbf{w}$ satisfies $\|\mathbf{w}\|_1 \le \sqrt{\ell+1} \cdot \|\mathbf{w}\|_2 < 2^{\ell/2} F$. Clearly this implies that the polynomial $w(y) = w_0 + w_1 y + \ldots + w_\ell y^\ell$ satisfies $|w(m)| < 2^{\ell/2} F$ whenever $|m| \le K/2$.

Thus one can apply Lemma 7.7 with F replaced by $2^{\ell/2}F$. Hence the decoding Condition (7.21) must be modified by adding a $\log(2^{\ell/2}) = \ell/2$ term to the right hand side, and then we will have a polynomial time list decoding algorithm working under the modified condition. We therefore conclude that one can list decode in *polynomial* time and output every m with $|m| \le K/2$ that satisfies

$$\sum_{i=1}^{n} a_i z_i \log p_i > \log(\ell+1) + \frac{\ell}{2} \log K + \frac{1}{\ell+1} \sum_{i=1}^{n} \binom{z_i+1}{2} \log p_i \ ,$$

as claimed in the theorem.[5] For easy reference, the CRT list decoding algorithm is described in Figure 7.6.2. □

Discussion of the assumed lattice algorithm: In the above proof we assumed a subroutine to compute the basis for an intersection lattice given the basis of the individual lattices. We now discuss how this may be done — further details and a more formal treatment may be found in [Coh93, Mic99].

[5]The astute reader might have noticed and be slightly bothered by the fact that we have ignored the $\log(2/(2-\gamma))$ term from Equation (7.21). This would cause an $o(1)$ difference to the result stated. Nevertheless, the result of Theorem 7.10 is itself accurate in its stated form. This is because, instead of the LLL algorithm, one can use Schnorr's improvement to the LLL algorithm, which finds an $2^{\varepsilon\ell}$ approximation to the shortest lattice vector in polynomial time, for any desired constant $\varepsilon > 0$. In this way, we can in fact weaken the requirement of Condition (7.20) by subtracting $(1/2-\varepsilon)\ell$ from the right hand side.

```
List Decode(r⃗, ℓ, z_1, z_2, ..., z_n)
```

1. Let $I_1^{z_i}$ be the set of polynomials that are integer linear combinations of $\{p_i^a(x - r_i)^{(z_i-a)}\}_{a=0}^{z_i}$.
2. Compute a basis for the *lattice* L of all degree ℓ polynomials belonging to $\bigcap_{i=1}^{n} I_i^{z_i}$.
3. Scale this lattice by multiplying the i'th coordinate by $(K/2)^{i-1}$ to produce the lattice L'.
4. Run LLL to find a short vector v' in L'; let it correspond to a degree ℓ polynomial $c(x) \in \mathbb{Z}[x]$.
5. Find all integer roots m of $c(x)$ (for example, by factoring $c(x)$ over $\mathbb{Z}[x]$ using [LLL82]).
6. For each root m with $|m| \leq K/2$, define the vector $\vec{a} = (a_1, a_2, \ldots, a_n)$ by $a_i = 1$ if $m \equiv r_i (\mathrm{mod} p_i)$, and $a_i = 0$ otherwise. Output m if $\vec{a}$ satisfies Condition (7.20).

Figure 7-2: The list decoding algorithm for Chinese Remainder codes

Let L be any full-dimensional lattice of dimension d, with basis given by the rows of the matrix M. We define the *dual* L^* of the lattice L to be $\{u \in \mathbb{R}^d \ : \ u \cdot v \in \mathbb{Z} \text{ for all } v \in L\}$. Note that the rows of $\left(M^{-1}\right)^T$ give a basis for L^*.

Note also that given bases for two lattices L_1 and L_2, a basis for (the closure of) the union of the two lattices, denoted $L_1 \cup L_2$, can be found efficiently using algorithms for computing the Hermite Normal Form of a generating set of vectors. Now, to compute a basis for the intersection of two lattices L_1 and L_2, observe that $L_1 \cap L_2 = (L_1^* \cup L_2^*)^*$. Therefore, by combining the facts above, one obtains an efficient algorithm for computing a basis for the intersection of full-dimensional lattices given bases for the individual lattices.

7.6.3 Applications to "interesting" weight settings

The result of Theorem 7.10 gives a general list decoding algorithm that works as long as a certain "weighted" condition is satisfied. We now get specific results for the CRT code for interesting choices of weights on the coordinate positions, through an appropriate choice of parameters (like ℓ, z_i) in Theorem 7.10. We begin by proving a version of Theorem 7.10 with arbitrary (not necessarily integer) values of z_i. The proof is somewhat technical but the main idea is simple: approximate the z_i's by large integers z_i^*, and pick a large enough "list size" parameter ℓ.

Theorem 7.11 *For list decoding of CRT codes, for any tolerance parameter $\varepsilon > 0$, and non-negative reals z_i, when given as input a received word $\mathbf{r}$, we can in time polynomial in $n, \log N$ and $1/\varepsilon$, find a list of all codewords such that*

$$\sum_{i=1}^{n} a_i z_i \log p_i \geq \sqrt{\log K \Big(\sum_{i=1}^{n} z_i^2 \log p_i + \varepsilon z_{\max}^2 \Big)} \ , \tag{7.22}$$

where the a_i's are defined as earlier.

Proof: We may assume that $z_{\max} = 1$ (note that the condition of (7.22) is invariant under scaling of the z_i's, so this can be ensured by dividing out all weights by $z_{\max}$). We will prove the claimed result by appealing to Theorem 7.10 on a suitably chosen set of *integer* weights z_i^*.

Let A be a large integer to be specified later in the proof. Set $z_i^* = \lceil Az_i \rceil$. By Theorem 7.10, for any positive integer ℓ we can successfully list decode (in $\mathrm{poly}(n, \log N, A, \ell)$ time) provided

$$\sum_{i=1}^{n} a_i z_i^* \log p_i > \log(\ell+1) + \frac{\ell}{2}\log K + \frac{1}{\ell+1}\sum_{i=1}^{n}\binom{z_i^*+1}{2}\log p_i.$$

We would like to pick a good choice for ℓ. Since $Az_i \le z_i^* < Az_i + 1$, the above condition is met whenever

$$\sum_{i=1}^{n} a_i z_i \log p_i \ge \frac{\log(\ell+1)}{A} + \frac{\ell}{2A}\log K + \frac{A}{2(\ell+1)}\sum_{i=1}^{n}\Big(z_i^2 + \frac{3}{A}z_i + \frac{2}{A^2}\Big)\log p_i\ . \qquad (7.23)$$

Define $Z_i = z_i^2 + \frac{3}{A}z_i + \frac{2}{A^2}$ for $1 \le i \le n$. Let us pick

$$\ell = \left\lceil A\sqrt{\frac{\sum_{i=1}^{n} Z_i \log p_i}{\log K}}\right\rceil - 1. \qquad (7.24)$$

It is not difficult to see that for this choice of ℓ, Condition (7.23) is met whenever

$$\sum_{i=1}^{n} a_i z_i \log p_i \ge \frac{1}{A}\log\Big(A\sqrt{\frac{\sum_{i=1}^{n} Z_i \log p_i}{\log K}} + 1\Big) + \sqrt{\log K\Big(\sum_{i=1}^{n} Z_i \log p_i\Big)}\ . \qquad (7.25)$$

For $A \ge \frac{10\log N}{\varepsilon}$, the right side of Equation (7.25) above is at most

$$O(\frac{\log\log N}{\log N}) + \sqrt{\log K\Big(\sum_{i=1}^{n} z_i^2 \log p_i + \frac{\varepsilon}{2}\Big)} \le \sqrt{\log K\Big(\sum_{i=1}^{n} z_i^2 \log p_i + \varepsilon\Big)}$$

for large N. Thus, Condition (7.25) is met provided

$$\sum_{i=1} a_i z_i \log p_i \ge \sqrt{\log K\Big(\sum_{i=1}^{n} z_i^2 \log p_i + \varepsilon\Big)},$$

and the proof is complete by noting that $A = O(\frac{\log N}{\varepsilon})$ and $\ell = O(\varepsilon^{-1}\log^{3/2} N)$, and so the overall runtime is polynomial in $n, \log N$ and ε^{-1}. □

Corollary 7.12 *For list decoding of CRT codes, for any tolerance parameter $\varepsilon > 0$, and non-negative real weights β_i, when given as input a received word $\mathbf{r}$, we can, in time polynomial in $n, \log N$ and $1/\varepsilon$, find a list of all codewords whose $\vec{\beta}$-weighted agreement with $\mathbf{r}$ satisfies:*

$$\sum_{i=1}^{n} a_i\beta_i \ge \sqrt{\log K\left(\sum_{i=1}^{n}\frac{\beta_i^2}{\log p_i} + \varepsilon\max_j \frac{\beta_j^2}{\log p_j}\right)}\ .$$

Proof: Follows by setting $z_i = \beta_i / \log p_i$ in the result of the above theorem. $\square$

Note that the above corollary implies that we can essentially "match" the combinatorial bound of Condition (7.19). Let us now collect further results for the "usual" uniform weighting of the codeword positions, namely $\beta_i = 1$ for each i.

Theorem 7.13 *For list decoding of CRT codes with parameters $(p_1, p_2, \ldots, p_n; K)$, for any $\varepsilon > 0$, we can in time polynomial in $n, \sum_i \log p_i$ and $1/\varepsilon$, find a list of all codewords which agree with a received word in t places provided $t \geq \sqrt{k(n+\varepsilon)}$.*

Proof: Let us apply Theorem 7.11 with $z_i = 1/\log p_{k+1}$ for $1 \leq i \leq k$, $z_i = 1/\log p_i$ for $k < i \leq n$, and $\varepsilon' = \varepsilon \log p_{k+1}$. This gives that we can decode whenever the number of agreements t is at least

$$k - \frac{\log K}{\log p_{k+1}} + \sqrt{\frac{\log K}{\log p_{k+1}} \left(\frac{\log K}{\log p_{k+1}} + \sum_{i=k+1}^{n} \frac{\log p_{k+1}}{\log p_i} + \varepsilon' \right)} \ .$$

Define $\Delta \stackrel{\text{def}}{=} k - \frac{\log K}{\log p_{k+1}}$; clearly $\Delta \geq 0$. Since $\log p_{k+1} \leq \log p_i$ for $i = k+1, \cdots, n$, the above condition is met whenever $t \geq \Delta + \sqrt{(k-\Delta)(n-\Delta+\varepsilon)}$. Now, a simple application of the Cauchy-Schwartz inequality shows that $\Delta + \sqrt{(k-\Delta)(n-\Delta+\varepsilon)} \leq \sqrt{k(n+\varepsilon)}$, and thus our decoding algorithm works whenever $t \geq \sqrt{k(n+\varepsilon)}$. $\square$

Theorem 7.14 *For list decoding of CRT codes with parameters $(p_1, p_2, \ldots, p_n; K)$, for any $\varepsilon > 0$, we can in time polynomial in $n, \sum_i \log p_i$ and $1/\varepsilon$, find a list of all codewords which agree with a received word in t places provided*

$$t \geq \sqrt{\log K \left(\sum_{i=1}^{n} \frac{1}{\log p_i} + \varepsilon \right)} \ .$$

Proof: This follows from Corollary 7.12 with $\beta_i = 1$ for $1 \leq i \leq n$. $\square$

Note that the result of Theorem 7.13 matches the combinatorial bound of Condition (7.18). The bounds in Theorem 7.13 and Theorem 7.14 are incomparable in general.

7.7 GMD decoding for CRT codes

For integers k, n, relatively prime integers $p_1 < p_2 < \cdots < p_n$, $K = \prod_{i=1}^{k} p_i$, and any integer j, $1 \leq j \leq n$, Goldreich, Ron, and Sudan [GRS00] gave a near-linear time algorithm to compute the unique integer m in the range $-K/2 < m \leq K/2$, if any, that satisfies

$$\sum_{i=1}^{j} a_i \log p_i > \frac{1}{2} \left(\sum_{i=1}^{j} \log p_i + \sum_{i=1}^{k} \log p_i \right) \tag{7.26}$$

where a_i is defined in the usual way: $a_i = 1$ if $m = r_i (\text{mod } p_i)$ and $a_i = 0$ otherwise. Note that the above algorithm decodes up to half the minimum $\vec{w}$-weighted distance ($\frac{1}{2} \cdot \log(N/K)$) for the "natural" weighting $w_i = \log p_i$ of the CRT code. Using this algorithm as the basic subroutine and running a GMD style algorithm similar to Forney [For66b], we are able to perform such a decoding for *any* "user-specified" choice of weights $\vec{\beta} =$

$\langle \beta_1, \beta_2, \ldots, \beta_n \rangle$. In other words, we give a soft decoding algorithm for CRT codes for the case of unambiguous decoding (the result of Theorem 7.10 being for the case of soft decoding with lists of size ℓ). While the list decoding algorithm decodes under a more general condition than the soft decoding algorithm to be discussed here, the advantage of this GMD based decoding algorithm is its simplicity and faster runtime (we will get a near-quadratic time algorithm).

To obtain the claimed decoding algorithm, we prove a more general result that applies to any code, and then apply it to the CRT code. Suppose we have an arbitrary code $\mathbf{C}$ of blocklength n. We show how to use a decoding algorithm designed for *any* weighting $\vec{\alpha}$ to produce one that works for the *desired* weighting $\vec{\beta}$. Define $A_\alpha = \sum_{i=1}^n \alpha_i - D_\alpha$ where D_α is $\vec{\alpha}$-weighted distance of the code, so that A_α is the maximum $\vec{\alpha}$-weighted agreement between two distinct codewords of $\mathbf{C}$. A_β for the weight vector $\vec{\beta}$ is defined similarly. We are now ready to state and prove the main result of this section:

Proposition 7.15 *Let $\mathbf{C}$ be an arbitrary code of blocklength n. Let $\vec{\alpha}, \vec{\beta} \in \mathbb{R}_+^n$ be positive real vectors such that $\frac{\beta_1}{\alpha_1} \geq \frac{\beta_2}{\alpha_2} \geq \cdots \geq \frac{\beta_n}{\alpha_n}$, and let A_α, A_β for the code $\mathbf{C}$ defined as described above. Suppose we have a polynomial time algorithm Alg_α that when given as input a received word $\mathbf{r} = \langle r_1, \ldots, r_n \rangle$ and an index j ($1 \leq j \leq n$), can find the unique codeword $\mathbf{c} \in \mathbf{C}$, if any, whose $\vec{\alpha}$-weighted agreement with $\mathbf{r}$ in the first j codeword positions is more than $\frac{1}{2}(\sum_{i=1}^{j} \alpha_i + A_\alpha)$. Then, for any vector of positive reals $\vec{\beta} = \langle \beta_1, \ldots, \beta_n \rangle$, there is a polynomial time algorithm Alg_β that when given as input a received word $\mathbf{r}$, outputs the unique codeword, if any, whose $\vec{\beta}$-weighted agreement with $\mathbf{r}$ is at least*

$$\frac{1}{2}\Big(\sum_{i=1}^{n} \beta_i + A_\beta + \beta_{\max}\Big) .$$

Moreover, the run-time of Alg_β is at most $O(n)$ times that of Alg_α.

Proof: Recall that the codeword positions i are ordered so that $\frac{\beta_1}{\alpha_1} \geq \frac{\beta_2}{\alpha_2} \geq \cdots \geq \frac{\beta_n}{\alpha_n}$. Define

$$\tilde{A}_\beta \stackrel{\text{def}}{=} \max_{\substack{\mathbf{x} \in [0,1]^n \\ \sum \alpha_i x_i \leq A_\alpha}} \left\{ \sum_{i=1}^{n} \beta_i x_i \right\} . \tag{7.27}$$

Note that under the condition $\mathbf{x} \in \{0,1\}^n$, the above would just define A_β; we relax the condition to $\mathbf{x} \in [0,1]^n$ in the above to define $\tilde{A}_\beta$. Clearly $\tilde{A}_\beta \geq A_\beta$. It is also easy to verify that $\tilde{A}_\beta < A_\beta + \beta_{\max}$. We will present an algorithm to find the unique codeword $\mathbf{c} = \langle c_1, c_2, \ldots, c_n \rangle \in \mathbf{C}$, if any, that satisfies

$$\sum_{i=1}^{n} a_i \beta_i > \frac{1}{2}(\sum_{i=1}^{n} \beta_i + \tilde{A}_\beta) \tag{7.28}$$

(where $a_i = 1$ if $c_i = r_i$ and 0 otherwise), and this will imply the claimed result (since $\tilde{A}_\beta < A_\beta + \beta_{\max}$). We now assume such a codeword $\mathbf{c}$ exists, as otherwise there is nothing to prove.

The algorithm Alg_β will simply run Alg_α for all values of j, $1 \leq j \leq n$, and pick the closest codeword among the (at most n) codewords which the runs of Alg_α returns. If this algorithm fails to find the codeword $\mathbf{c}$ that satisfies Condition (7.28), then, by the hypothesis of the Theorem, the following condition must hold for every j, $1 \le j \le n$:

$$2\sum_{i=1}^{j} a_i\alpha_i \le \sum_{i=1}^{j} \alpha_i + A_\alpha \ . \tag{7.29}$$

Let $\tilde{\mathbf{x}} = \langle 1\ 1\ \cdots\ 1\ \varepsilon\ 0\ \cdots\ 0\rangle$ be a vector such that $\sum_{i=1}^{n} \alpha_i \tilde{x}_i = A_\alpha$ (here $0 \le \varepsilon < 1$). Denote by ℓ the last position where $\tilde{x}_i = 1$ (so that $\tilde{x}_\ell = 1$ and $\tilde{x}_{\ell+1} = \varepsilon$). By our definition (7.27) $\tilde{A}_\beta \ge \sum_i \beta_i \tilde{x}_i$ (in fact by the ordering of the codeword positions it is also true that $\tilde{A}_\beta = \sum \beta_i \tilde{x}_i$, though we will not need this). Now for $j \ge \ell + 1$, $A_\alpha = \sum_{i=1}^{n} \alpha_i \tilde{x}_i = \sum_{i=1}^{j} \alpha_i \tilde{x}_i$. Also, for $1 \le j \le \ell$, we have the obvious inequality $\sum_{i=1}^{j} a_i\alpha_i \le \sum_{i=1}^{j} \alpha_i = \sum_{i=1}^{j} \alpha_i \tilde{x}_i$, which implies

$$2\sum_{i=1}^{j} a_i\alpha_i \le \sum_{i=1}^{j} \alpha_i + \sum_{i=1}^{j} \alpha_i \tilde{x}_i \ .$$

Combining the above with Equation (7.29) we obtain that the following uniform condition that holds for all j, $1 \le j \le n$:

$$2\sum_{i=1}^{j} a_i\alpha_i \le \sum_{i=1}^{j} \alpha_i + \sum_{i=1}^{j} \alpha_i \tilde{x}_i \ . \tag{7.30}$$

Multiplying the j^{th} inequality above by the non-negative quantity $(\frac{\beta_j}{\alpha_j} - \frac{\beta_{j+1}}{\alpha_{j+1}})$ for $1 \le j \le n$ (define $\beta_{n+1} = 0$ and $\alpha_{n+1} = 1$), and adding the resulting inequalities, we get

$$2\sum_{i=1}^{n} a_i\beta_i \le \sum_{i=1}^{n} \beta_i + \sum_{i=1}^{n} \beta_i \tilde{x}_i \le \sum_{i=1}^{n} \beta_i + \tilde{A}_\beta \ ,$$

which contradicts Condition (7.28). Thus the codeword $\mathbf{c}$ that satisfies (7.28), if any, will indeed be output by the algorithm Alg_β. □

Theorem 7.16 *For the CRT code with parameters $(n, k; p_1, p_2, \ldots, p_n)$, for any received word $\mathbf{r} = \langle r_1, r_2, \ldots, r_n\rangle$, there is a polynomial time (in fact near-quadratic time) algorithm to find the unique codeword $m = (m_1, m_2, \ \ldots, m_n)$, if any, that agrees with $\mathbf{r}$ in at least $\frac{n+k}{2}$ positions.*

Proof: By the result of [GRS00], we have a near-linear time decoding algorithm for the weighting $\alpha_i = \log p_i$ and $A_\alpha = \log K$ (where $K = p_1 p_2 \cdots p_k$). For $\vec{\beta}$ equal to the all-ones vector, we have $A_\beta = k - 1$. Therefore, by Proposition 7.15, we can find the unique codeword m that agrees with $\mathbf{r}$ in at least $(n+k)/2$ places, as claimed. □

7.8 Bibliographic Notes

The redundancy property of the Chinese Remainder representation has been exploited often in theoretical computer science. For example, the Karp-Rabin pattern matching algorithm is based on this redundancy [KR81]. The CRT representation of an integer allows one to reduce computation over large integers to that over small integers. This is also useful in certain complexity-theoretic settings, a notable example being its use in showing the hardness of computing the permanent of 0/1 matrices [Val79].

The natural error-correcting code (the CRT code) that results from the Chinese Remainder representation has also been studied often in the literature (see [SJJT86, KKLS94] and the references there in). The CRT code was proposed as an alternate method for implementing secret sharing [Den83, AB83]. Mandelbaum [Man76, Man78] was the first to consider the basic algorithmic question of decoding the CRT code up to half the minimum distance. He succeeded in giving such a decoding algorithm; however, the runtime of his algorithm was polynomial only when the p_i's are very close to one another, and could be exponential in n otherwise. Goldreich, Ron and Sudan [GRS00] present and analyze a variant of Mandelbaum's algorithm, which can be implemented in near-linear time, and can unique decode the CRT code up to $\frac{(n-k)\log p_1}{\log p_1+\log p_n}$ errors. This is a close approximation to half the distance when the primes are reasonably close to one another.

Inspired by the success of list decoding algorithms for Reed-Solomon and AG-codes, Goldreich *et al* [GRS00] considered the list decoding problem for CRT codes. They presented a polynomial time algorithm to list decode CRT codes up to (about) $(n-\sqrt{2kn\frac{\log p_n}{\log p_1}})$ errors. For primes which are close to one another and for small values of k/n, this decodes well beyond half the distance of the code. However, this is not the case when the primes vary widely in size and/or the "rate" k/n is large. One of the motivations of the list decoding algorithm in [GRS00] was an application to the average-case hardness of the permanent on certain random matrices — a discussion of this connection appears in the conference version [GRS99] of the same paper. Håstad and Näslund [HN98] used the algorithm of [GRS00] to construct new hardcore predicates based on one-way functions.

Subsequent to this, Boneh [Bon00] improved the list decoding algorithm of [GRS00]. His algorithm could correct up to about $(n-\sqrt{kn\frac{\log p_n}{\log p_1}})$ errors. One weakness common to all the above results on CRT decoding is their poor(er) performance if the primes vary significantly in size. This can cause the algorithm of Mandelbaum [Man76] to take exponential time, while it degrades the number of errors that the algorithms of Goldreich *et al.* [GRS00], or Boneh [Bon00] can correct. This weakness is due to an eccentricity of the CRT code: its alphabet size is not uniform, and so the "contribution" of an error is not independent of its location (knowing a residue modulo a larger p_i correctly gives more information than knowing a residue modulo a smaller p_i). Hence one needs to suitably "reweight" the coordinate positions in order to compensate for this inherent disparity between the various positions. This is exactly what the weighted decoding algorithm we discussed in this chapter allows us to do. It thereby permits efficient decoding up to about $(n-\sqrt{kn})$ errors, and thus completely removes the dependence of the number of correctable errors on the size of the p_i's. It was the development of this soft decoding algorithm for CRT codes that caused us to examine in greater detail the algebra underlying the various list decoding algorithms and unveil the unified ideal-theoretic view of decoding presented in this chapter.

The CRT decoding algorithms discussed in this chapter appear in [GSS00]. The general "ideal-theoretic" approach to list decoding algebraic codes was sketched in [GSS00] as an appendix, and it has been further developed and expanded for presentation in this chapter.

Chapter 8

List Decoding of Concatenated Codes

8.1 Introduction

The decoding algorithms for Reed-Solomon and AG-codes provide the first results which algorithmically exploit the potential of list decoding well beyond half the minimum distance. In addition, these codes are widely studied and used, and thus these algorithms are not only theoretically interesting, but could also have a lot of practical impact. In this chapter, we are interested in polynomial time constructible linear codes over $\mathbb{F}_q$ for a *small*, fixed q, which can be efficiently list decoded from a large, and essentially "maximum" possible, fraction of errors, and which have good rate. Codes over small alphabets are desirable for several applications. Of particular interest to us will be *binary* codes. Such small alphabet codes with high list decodability cannot be directly obtained from Reed-Solomon or algebraic-geometric codes. Recall that Reed-Solomon codes require the alphabet size to be at least as large as the blocklength of the code. While AG-codes can be defined over an alphabet of fixed size q, their performance is limited by certain algebraic barriers. In particular these rule out the existence of good binary AG-codes, and even for larger q limit their list decodability to much less than what is in general possible for q-ary codes.

The reader will recall that in Chapter 5 we had investigated trade-offs between list decodability and rate for q-ary codes. The results of this chapter can be viewed as an attempt to constructivize, to whatever extent possible, the existential bounds established in Chapter 5.

While Reed-Solomon and AG-codes do not yield good list decodable q-ary codes for small q, we show in this chapter that concatenated codes that use them as outer codes along with appropriate inner codes do achieve small alphabet size together with excellent algorithmic list decodability properties. The concatenated codes are decoded in two steps: in the first step, a decoding of the portions of the received word corresponding to the various inner encodings is performed. The inner code, owing to its small dimension, can be decoded by brute-force in the allowed runtime (which is polynomial in the entire blocklength). The inner decoding passes to the outer decoder, information concerning the possible symbols at each position, together with appropriate weights or confidence information. The decoding is then completed by running the soft (list) decoding algorithms for the outer Reed-Solomon or AG-code from Chapter 6. This represents a novel use of the soft decoding algorithm, and is one of the few such uses where a simple worst-case analytic bound on the number of errors corrected by the algorithm can be proved. (In contrast, a large body of literature on soft decoding applies it to probabilistic channels and obtains either analytic or experimental estimates of the decoding error probability under the specific error model (cf. [For66a, For66b, KV00]).)

8.2 Context and motivation of results

We are interested in families of q-ary codes that can be efficiently list decoded from a large fraction of errors. Decoding from a fraction of errors beyond $(1 - 1/q)$ is information-theoretically impossible for q-ary codes (of positive rate). This is because a random received word will differ from any particular codeword in a expected fraction $(1 - 1/q)$ of positions. Therefore, list decoding beyond a fraction $(1 - 1/q)$ of errors will require a list size proportional to the total number of codewords, and hence an exponential list size for code families with positive rate.

Therefore, we are interested in families of q-ary codes that can be list decoded from a fraction p of errors, for $0 < p < (1 - 1/q)$. Having fixed the desired level of error-resilience, the quantity we would like to optimize is the rate of the code family. This is exactly in the spirit of the results from Chapter 5, except that we are now interested in both explicit specifications or polynomial time constructions of the code, *and* efficient list decoding algorithms (and not just a good combinatorial list decodability property).

The main tools used for the above pursuit are concatenated codes with outer Reed-Solomon or AG-codes and inner codes with good combinatorial list decodability and/or distance properties. This gives a polynomial time construction (and sometimes even explicit specification) of, say, a binary code with good combinatorial list decodability. We enhance the nice combinatorial properties of concatenated codes with algorithmic ones, by presenting fairly general schemes to efficiently decode these codes to close to their Johnson radius (which is the *a priori* "list decoding capacity" of any code).

In order to present the above constructive results, we focus on the "high-noise regime", i.e., list decoding up to a fraction $(1 - 1/q - \varepsilon)$ of errors for q-ary codes. For such codes, the results of Chapter 5 imply that the best rate achievable is $\Theta(\varepsilon^2)$. Our goal will be to approach this performance with explicit codes and efficient decoding algorithms. We loosely refer to codes that can correct such a large fraction (approaching $(1 - 1/q)$) of errors as *highly list decodable.*

We focus on the high-noise regime since it brings out the asymptotics very well. Even optimizing the exponent of ε in the rate is a non-trivial problem to begin with in this context. Hence, working in the high-noise regime implies that (at least for current results) we need not be very careful with the constant factors in the rate that are independent of ε (since the $\varepsilon^{O(1)}$ term is the dominant one in the rate). Moreover, there is a natural and well-posed goal of approaching the "optimal" rate, i.e., obtaining the best possible exponent of ε in the rate. We note here that this is a very asymptotic and computer science style perspective, and indeed the motivation comes partly from applications of list decoding to complexity theory, to be discussed in Chapter 12, where the high-noise regime is the most interesting and useful one to focus on. Coding theorists are sometimes disturbed by the low rate in the way we state some of our results. But we would like to stress that the low rate is unavoidable since we are targeting decoding from a very large fraction of errors. Moreover, we believe that optimizing our techniques for the high-noise (and *consequently* low-rate) regime is a good first step, and that the techniques can eventually be applied to a more careful, thorough investigation of the situation where we do not wish to correct such a large fraction of errors. The results of the next two chapters will also be motivated by and stated for the high-noise regime – these chapters will deal with codes over large (but constant-sized) alphabets and erasure codes, respectively.

8.3 Overview of Results

We present list decoding algorithms for several families of concatenated codes. Recall that the distance of a concatenated code whose outer code has distance D and inner code has distance d is at least Dd (and this quantity is referred to as the *designed distance* of the concatenated code). Unique decoding algorithms to decode up to the *product bound*, namely to correct fewer than $Dd/2$ errors, are known based on Generalized Minimum Distance decoding of the outer code [For66b, Jus72] (this is also discussed in detail in Appendix A of this thesis). The focus of this chapter is on list decoding algorithms that permit recovery well beyond the product bound for certain families of concatenated codes. A discussion of the specific results follows.

In Section 8.4, we give list decoding algorithms for codes where the outer code is a Reed-Solomon or Algebraic-geometric code and the inner code is a Hadamard code. Our algorithms decode these codes up to the Johnson bound on list decoding radius. These algorithms also serve as a beautiful illustration of the power of our soft decoding algorithms for list decoding Reed-Solomon and AG-codes from Chapter 6. The construction with an appropriate algebraic-geometric outer code, upon picking parameters suitably, gives us a construction of q-ary codes of rate $\Omega(\varepsilon^6)$ list decodable up to a fraction $(1 - 1/q - \varepsilon)$ of errors.

In Section 8.5, we present a decoding algorithm for concatenated codes with outer Reed-Solomon or AG-code and an arbitrary inner code. The algorithm falls short of decoding up to the Johnson radius, but decodes well beyond half the distance when the rate of the outer code is small. In particular, it gives an alternative construction of q-ary codes of rate $\Omega(\varepsilon^6)$ decodable up to a fraction $(1-1/q-\varepsilon)$ of errors. The advantage of this construction is that one can use Reed-Solomon codes as opposed to the more complicated AG-codes necessary for the earlier result using Hadamard codes. The construction and decoding algorithms are consequently also easier and faster.

Finally, in Section 8.6, we use special purpose codes as inner codes in a concatenated construction to obtain binary linear codes of rate $\Omega(\varepsilon^4)$ efficiently list decodable from a fraction $(1/2-\varepsilon)$ of errors. The inner codes are a more general variant of the ones guaranteed by Theorem 5.5 of Chapter 5. We should remark that we are able to obtain this result only for binary codes.[1]

We stress here that our construction that has rate $\Omega(\varepsilon^4)$ is **not** obtained by constructing a large distance binary code and then appealing to the Johnson bound to argue that the list decoding radius is at least $(1/2 - \varepsilon)$. Indeed, this will require the relative distance to be at least $(1/2 - O(\varepsilon^2))$ and the best known polynomial time constructions of such codes yield a rate of only about $\Omega(\varepsilon^6)$. In fact, a polynomial time construction of binary code

[1] We know how to achieve a similar performance for general alphabets if we relax the requirement of linearity. The next chapter will discuss several non-linear code constructions with good list decodability. The codes, though not linear, will be based on "pseudolinear" codes, and will possess succinct representation and be efficiently encodable/decodable. Using random q-ary pseudolinear codes as inner codes will permit us to obtain codes of rate $\Omega(\varepsilon^4)$ list decodable up to a fraction $(1-1/q-\varepsilon)$ of errors, for every prime power q. We, however, do not elaborate on this point further.

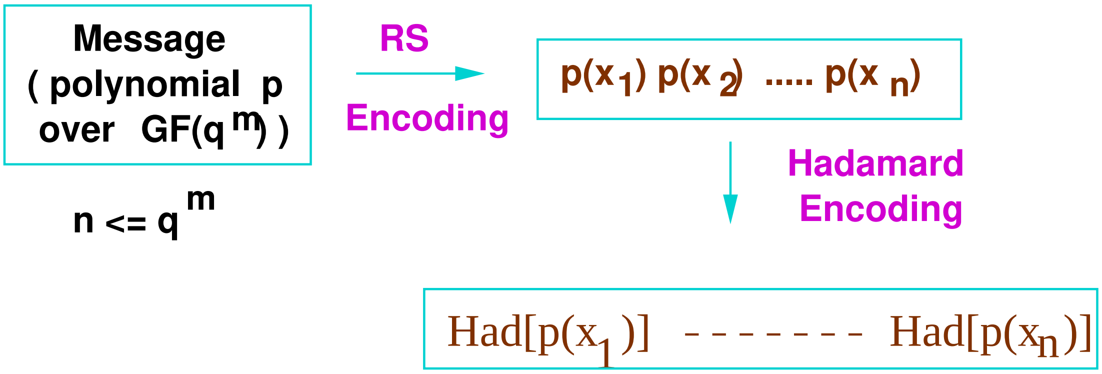

Figure 8-1: Reed-Solomon concatenated with a Hadamard code

families of relative distance $(1/2 - O(\varepsilon^2))$ and rate $\Omega(\varepsilon^4)$ will asymptotically match the Gilbert-Varshamov bound at low rates, and will be a *major* breakthrough in coding theory.

The results of this chapter focus exclusively on linear codes. Some results in the next two chapters will resort to a certain "limited" amount of non-linearity.

8.4 Decoding concatenated codes with inner Hadamard code

We present list decoding algorithms for concatenated codes with an outer algebraic code and inner Hadamard code. The motivation of considering the Hadamard code is its nice properties which we exploit to decode the concatenated codes up to their Johnson radius. Concatenated codes with algebraic-geometric outer code and inner Hadamard code are among the best explicitly known codes in terms of the rate vs. distance trade-offs. By decoding such codes up to the Johnson radius, we will get codes of good rate and very high list decodability. This is our primary motivation for considering decoding algorithms for such concatenated codes.

Recall the definition of Hadamard codes from Chapter 2. The q-ary Hadamard code of dimension m encodes an $\mathbf{x} \in \mathbb{F}_q^m$ by $\langle x \cdot z \rangle_{z \in \mathbb{F}_q^m}$ (i.e. by its dot product over $\mathbb{F}_q$ with every vector $\mathbf{z} \in \mathbb{F}_q^m$). It has blocklength q^m and minimum distance $(1 - 1/q) \cdot q^m$; in fact all non-zero codewords in the code have Hamming weight $(1 - 1/q) \cdot q^m$.

The codes considered in this section will be the concatenation of a Reed-Solomon or AG-code over $\mathrm{GF}(q^m)$ with the q-ary Hadamard code of dimension m. Note the number of outer codeword symbols (i.e., q^m) exactly equals the number of Hadamard codewords, so concatenation of these codes is well-defined. Figure 8-1 depicts the structure of a Reed-Solomon concatenated with a Hadamard code. The encoding of a message (a polynomial) p will be $\mathrm{Had}(p(x_1))\mathrm{Had}(p(x_2)) \cdots \mathrm{Had}(p(x_n))$, where $x_1, \ldots, x_n$ are distinct elements in $\mathrm{GF}(q^m)$ that are used in defining the Reed-Solomon code. (To encode an element $\alpha \in \mathrm{GF}(q^m)$ using the Hadamard code, one views α as a string of length m over $\mathrm{GF}(q)$ using some fixed representation of $\mathrm{GF}(q^m)$ as vectors of length m over $\mathrm{GF}(q)$.)

Jumping ahead to how our decoding will proceed, the inner decoder will "decode" the Hadamard code and pass information concerning the possible symbols at each position, together with appropriate weights. Suppose the i'th block (corresponding to the inner

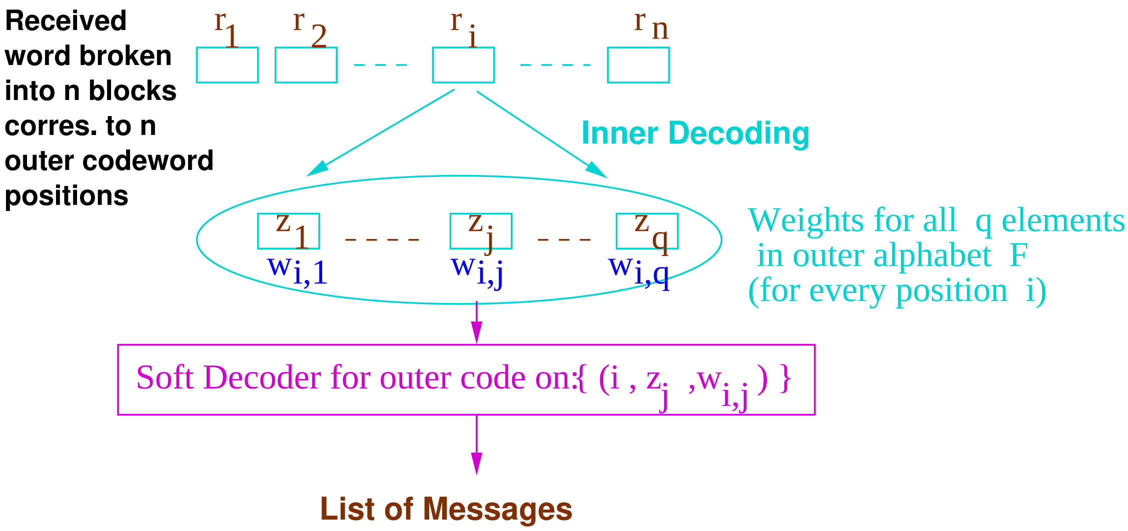

Figure 8-2: The basic idea behind our decoding algorithms for concatenated codes. For each position of the outer code, the inner decoding passes a weight or confidence rating for every element of the field $F = \mathrm{GF}(q)$. These are then used by a soft list decoding algorithm for the outer code to finish the decoding.

encoding of the i'th outer codeword symbol) of the received word is r_i. It is natural that the weight that the inner decoder gives to a symbol $\alpha \in \mathrm{GF}(q^m)$ for position i should be a decreasing function of $\Delta(\mathrm{Had}(\alpha), r_i)$ (where $\Delta(x, y)$ measures the Hamming distance between x and y). This is because, intuitively, the larger this distance, the smaller is the likelihood that the i'th symbol of the outer codeword was α. In fact, the inner decoder will set weights to be a decreasing linear function of this distance (the linearity makes possible a precise analysis of the number of errors corrected). Specifically, the weight for symbol α for the i'th block r_i of received word will be set to

$$\left(1 - \frac{q}{q-1} \frac{\Delta(r_i, \mathrm{Had}(\alpha))}{q^m}\right) .$$

The decoding is then completed by running the soft (list) decoding algorithms for the outer Reed-Solomon or AG-code from Chapter 6 with these choice of weights. This is in fact the procedure used for decoding all of the concatenated codes in this chapter. Figure 8-2 illustrates the basic structure of our decoding schemes for concatenated codes.

Recalling the statements of Theorems 6.21 and 6.35, the sum of the squares of the weights is an important quantity that governs the performance of the decoding algorithm. Good upper bounds on this sum will permit a good analysis of the error-correction performance of the algorithm. Below, we provide such an upper bound for the choice of weights made by the inner decoder in decoding the Hadamard code.

Proposition 8.1 *Let q be a prime power and let m be a positive integer. Let* $\mathrm{Had} : \mathbb{F}_{q^m} \to \mathbb{F}_q^{q^m}$ *be the q-ary Hadamard code of dimension m and blocklength q^m. Let $f \in \mathbb{F}_q^{q^m}$ be an* *arbitrary vector of length q^m over $\mathbb{F}_q$. Then*

$$\sum_{\alpha \in \mathrm{GF}(q^m)} \Big(1 - \frac{q}{q-1} \cdot \frac{\Delta(f, \mathrm{Had}(\alpha))}{q^m}\Big)^2 \le 1 \ . \tag{8.1}$$

Remark: For the case $q = 2$, $(1 - 2\Delta(f, \mathrm{Had}(\alpha))/2^m)$ equals the *Fourier coefficient* $\hat{f}_\alpha$ of f with respect to $\mathrm{Had}(\alpha)$, viewed as a linear function mapping $\mathbb{F}_q^m$ to $\mathbb{F}_q$. In this case, the statement of the Proposition in fact holds with equality, and is simply the standard Plancherel's identity $\sum_\alpha \hat{f}_\alpha^2 = 1$. The result for the non-binary case appears in [Kiw97], and the proof there is based on the MacWilliams-Sloane identities for the weight distribution of dual codes; we give a more elementary proof below.

Proof: The proof works by embedding any string $f \in \mathbb{F}_q^{q^m}$ as a q^{m+1}-dimensional *real unit vector*. The embedding will be such that for every $\alpha \neq \beta \in \mathrm{GF}(q^m)$, the vectors associated with the Hadamard codewords $\mathrm{Had}(\alpha)$ and $\mathrm{Had}(\beta)$ will be orthogonal (in the usual real dot product over $\mathbb{R}^{q^m \cdot q}$). Furthermore, the embedding will be such that the quantity

$$\Big(1 - \frac{q}{q-1} \cdot \frac{\Delta(f, g)}{q^m}\Big)$$

for every two functions $f, g \in \mathbb{F}_q^{q^m}$ will simply be the dot product of the vectors associated with f, g. The result will then follow since the sum of the squares of the projections of a unit vector along pairwise orthogonal vectors can be at most 1.

Suppose the q elements of $\mathbb{F}_q$ are $\gamma_1, \gamma_2, \ldots, \gamma_q$. Associate a q-dimensional vector e_i with γ_i as follows (e_{il} denotes the l'th component of e_i): $e_{ii} = \sqrt{(q-1)/q}$ and $e_{il} = -1/\sqrt{q(q-1)}$ for $l \neq i$. Note that this definition satisfies $\langle e_i, e_i \rangle = 1$ and $\langle e_i, e_j \rangle = -1/(q-1)$ for $i \neq j$. For a string $f \in \mathbb{F}_q^{q^m}$, we view f as the q^{m+1}-dimensional vector obtained in the obvious way by juxtaposing the q-dimensional vectors for each of the q^m values which f takes on its domain, and then normalizing it to a unit vector (by dividing every component by $\sqrt{q^m}$). By abuse of notation, we will denote the real vector associated with f also by f.

Note that when we take the inner product $\langle f, g \rangle$, we get a contribution of $1/q^m$ corresponding to the positions where f, g agree, and a contribution of $\frac{-1}{(q-1)} \cdot q^{-m}$ corresponding to places where f, g differ. Hence we have

$$\langle f, g \rangle = (q^m - \Delta(f, g)) \cdot q^{-m} + \Delta(f, g) \cdot \Big(\frac{-1}{q-1}\Big) \cdot q^{-m} = 1 - \frac{q}{q-1} \cdot \frac{\Delta(f, g)}{q^m} \ .$$

Now, for $\alpha \neq \beta \in \mathrm{GF}(q^m)$, $\Delta(\mathrm{Had}(\alpha), \mathrm{Had}(\beta)) = (1 - 1/q) \cdot q^m$ (recall that two distinct codewords in the Hadamard code corresponding to $\mathbb{F}_q^m$ agree on exactly q^{m-1} places and differ at $q^{m-1}(q-1)$ places). Thus, for $\alpha \neq \beta$, we have $\langle \mathrm{Had}(\alpha), \mathrm{Had}(\beta) \rangle = 0$. Also by our choice of vectors, $\langle \mathrm{Had}(\alpha), \mathrm{Had}(\alpha) \rangle = 1$. Hence the q^m vectors associated with the Hadamard codewords are pairwise orthogonal unit vectors. Using this fact the result follows since

$$\sum_{\alpha \in \mathbb{F}_q^m} \Big(1 - \frac{q}{q-1} \frac{\Delta(f, \mathrm{Had}(\alpha))}{q^m}\Big)^2 = \sum_\alpha \langle f, \mathrm{Had}(\alpha) \rangle^2 \le \langle f, f \rangle = 1 \ . \qquad \Box$$

8.4.1 Reed-Solomon concatenated with Hadamard code

We now use the above result to analyze the error-correction capability of Reed-Solomon codes concatenated with Hadamard code, when using the soft decoding algorithm for Reed-Solomon together with the weights passed by the Hadamard decoding.

Theorem 8.2 *Let C be q-ary code of blocklength n and relative distance δ, that is obtained by concatenation of a Reed-Solomon code over* $\mathrm{GF}(q^m)$ *with the Hadamard code of dimension m, for some m. Then, there is a polynomial time list decoding algorithm for C that decodes up to E errors where*

$$E = n\Big(1 - \frac{1}{q}\Big)\Big(1 - \sqrt{1 - \frac{q\delta}{q-1}}\Big) - O(1) \ .$$

(In other words, one can decode such a code up to, essentially, the q-ary Johnson bound on list decoding radius.)

Proof: The relative distance of a q-ary Hadamard code is $(1-1/q)$, and in fact all non-zero codewords have the same Hamming weight. Hence, it follows that in order for the relative distance of the concatenated code C to be δ, the relative distance of the outer Reed-Solomon code, call it C_{RS}, must be $q\delta/(q-1)$. Let the blocklength of C_{RS} be $n_0 \le q^m$, and its dimension be (k_0+1), where

$$k_0 = n_0\Big(1 - \frac{q\delta}{q-1}\Big) \ . \tag{8.2}$$

Let $x_1, x_2, \ldots, x_{n_0}$ be distinct elements of $\mathrm{GF}(q^m)$ that are used to define C_{RS}. Thus, the messages of C_{RS} (and hence C, too) are degree k_0 polynomials over $\mathrm{GF}(q^m)$, and a polynomial p is encoded under C_{RS} as $\langle p(x_1), p(x_2), \ldots, p(x_{n_0})\rangle$. The blocklength n of the overall concatenated code C satisfies $n = n_0 q^m$, and its dimension equals $(k_0+1)m$.

Let $y \in \mathbb{F}_q^n$ be a "received word"; the task of list decoding that we wish to solve is to obtain a list of all codewords of C within a Hamming distance of E from y. For $1 \le i \le n_0$, denote by y_i the portion of y in block i of the codeword (i.e., the portion corresponding to the Hadamard encoding of the i^{th} symbol of the outer code).

We now perform the "decoding" of each of the n_0 blocks y_i as follows. For $1 \le i \le n_0$ and $\alpha \in \mathrm{GF}(q^m)$, compute the Hamming distance $e_{i,\alpha}$ between y_i and $\mathrm{Had}(\alpha)$, and then compute the *weight* $w_{i,\alpha}$ as:

$$w_{i,\alpha} \stackrel{\text{def}}{=} \max\Big\{ \Big(1 - \frac{q}{q-1} \cdot \frac{e_{i,\alpha}}{q^m}\Big),\ 0 \Big\} \ . \tag{8.3}$$

Note the computation of all these weights can be done by a straightforward brute-force computation in $O(n_0(q^m)^2) = O(n^2/n_0)$ time. Thus all the inner decodings can be performed efficiently in at most quadratic time.

The key combinatorial property of these weights, that follows from Proposition 8.1 above, is that

$$\sum_{\alpha} w_{i,\alpha}^2 \le 1 \ , \tag{8.4}$$

for every i, $1 \le i \le n_0$. These weights will now be "passed" to the outer Reed-Solomon decoder as the confidence information about the various symbols of the Reed-Solomon codeword. For the outer decoder, we will use the soft decoding algorithm from Chapter 6.

Specifically, we will use the result of Theorem 6.21. Applied to this context, the result implies that, for any desired tolerance parameter $\varepsilon > 0$, we can find in time polynomial in n_0 and $1/\epsilon$, a list of all polynomials p over $\mathrm{GF}(q^m)$ of degree at most k_0 that satisfy

$$\sum_{i=1}^{n_0} w_{i,p(x_i)} \geq \Big(k_0 \cdot \sum_{\substack{1 \leq i \leq n_0 \\ \alpha \in \mathrm{GF}(q^m)}} w_{i,\alpha}^2\Big)^{1/2} + \epsilon \max_{i,\alpha} w_{i,\alpha} \,. \tag{8.5}$$

Applied to the choice of weights (8.3) and using Equation (8.4), the decoding algorithm can thus retrieve all codewords corresponding to degree k_0 polynomials p for which

$$\sum_{i=1}^{n_0} \Big(1 - \frac{q}{q-1} \cdot \frac{e_{i,p(x_i)}}{q^m}\Big) \geq \sqrt{k_0 n_0} + \epsilon \,. \tag{8.6}$$

Note that $w_{i,p(x_i)} \geq (1 - \frac{q}{q-1} \cdot \frac{e_{i,p(x_i)}}{q^m})$, and hence if the above condition is satisfied then so is Condition (8.5).

Now, recall that $e_{i,p(x_i)} = \Delta(y_i, \mathrm{Had}(p(x_i)))$. Hence, (8.6) above implies that we can find all codewords at a distance E from the received word y provided

$$\begin{aligned} n_0 - \frac{qE}{(q-1)\cdot q^m} &\geq \sqrt{k_0 n_0} + \epsilon \text{ or} \\ \frac{qE}{q-1} &\leq n\Big(1 - \sqrt{\frac{k_0}{n_0}} - \frac{\epsilon}{\sqrt{n}}\Big) \quad (\text{since } n = n_0 q^m) \\ \Longleftarrow \qquad E &\leq n\Big(\frac{q-1}{q}\Big)\Big(1 - \sqrt{1 - \frac{q\delta}{q-1}}\Big) - \epsilon q^m \,, \end{aligned}$$

where in the last step we use the value of k_0 from Equation (8.2). If we pick $\epsilon \leq 1/n$, this implies we can list decode up to

$$E = n\Big(\frac{q-1}{q}\Big)\Big(1 - \sqrt{1 - \frac{q\delta}{q-1}}\Big) - O(1)$$

errors, as desired. □

8.4.2 AG-code concatenated with Hadamard code

The result of Theorem 8.2 decodes the concatenated code up to the Johnson radius, and thus has very good error-correction performance for the concerned code. However, while interesting for a variety of reasons, from a coding standpoint, the Reed-Solomon concatenated with Hadamard codes are not very attractive. This is because they have very low rate, since the inner Hadamard code maps m symbols into q^m symbols, and thus has very poor, vanishing, rate for large m. In particular, the family of codes is not asymptotically good, and has rate rapidly tending to 0 in the limit of large blocklengths. It is thus way off our pursuit of codes list decodable to a fraction $(1 - 1/q - \varepsilon)$ of errors with rate somewhat close to $\Theta(\varepsilon^2)$.

In this section, we will adapt the result of Theorem 8.2 to concatenated codes with outer AG-code (instead of Reed-Solomon code). The inner code will be the Hadamard code as before. The rate of the overall code will once again not be great, since it will inherit the poor rate of the Hadamard code. But since AG-codes with good parameters exist over a fixed alphabet of size independent of the blocklength, the inner Hadamard code will now be a constant-sized code, and thus will have some fixed, albeit small, rate. Thus, we will be able to achieve positive rate (i.e. rate which is at least r for some fixed constant $r > 0$ that is independent of the blocklength) for the overall code. As a corollary, in the next section, we will plug in the best-known AG-codes (those discussed in Section 6.3.9) to obtain constructions of codes which are list decodable up to a fraction $(1 - 1/q - \varepsilon)$ of errors and have rate $\Omega(\varepsilon^6)$.

The formal result concerning list decoding AG-codes concatenated with Hadamard codes is stated below. The hypothesis about a suitable representation of the code is necessary in the statement of the theorem, since the decoding algorithms of Chapter 6 also made this assumption.

Theorem 8.3 *Let $C_{\mathrm{AG-Had}}$ be q-ary code of blocklength n and relative distance at least δ, that is obtained by concatenation of an algebraic-geometric code over $\mathrm{GF}(q^m)$ of relative designed distance $q\delta/(q-1)$ with the q-ary Hadamard code of dimension m, for some m. Then, there exists a representation of the code of size polynomial in n under which a polynomial time list decoding algorithm exists to list decode $C_{\mathrm{AG-Had}}$ up to E errors, where*

$$E = n\Big(1 - \frac{1}{q}\Big)\Big(1 - \sqrt{1 - \frac{q\delta}{q-1}}\Big) - O(1) \ .$$

(In other words, one can decode such a code up to, essentially, the q-ary Johnson bound on list decoding radius.)

Proof: The proof parallels that of the earlier result (Theorem 8.2) where the outer code was a Reed-Solomon code. The inner decodings of the various Hadamard codes proceeds exactly as before, passing weights to the outer decoder. Now, for the outer decoder we can make use of the soft list decoding algorithm for AG-codes developed in Theorem 6.35, instead of the Reed-Solomon soft decoder. This is really the only change necessary to the proof of Theorem 8.2, and the claimed bound on the number of errors corrected follows as before. We omit the details. The soft decoding algorithm for AG-codes from Theorem 6.35 works in polynomial time only assuming a specific (non-standard) representation of the AG-code, which necessitates the hypothesis about the representation of the code in the statement of the theorem. □

8.4.3 Consequence for highly list decodable codes

We now apply Theorem 6.35 with AG-codes that achieve the best known trade-off between rate and distance (from Section 6.3.9 of Chapter 6). This gives us codes list decodable up to a fraction $(1 - 1/q - \varepsilon)$ of errors and which have reasonably good rate.

Corollary 8.4 *For every fixed prime power q, the following holds: For every $\varepsilon > 0$, there exists a family of linear codes over $\mathbb{F}_q$ with the following properties:*

(i) The family is polynomial time constructible in that the generator matrix of a code of blocklength n in the family can be computed in time a fixed polynomial in n.

(ii) Its rate is $\Omega(\varepsilon^6 \cdot \log(1/\varepsilon))$.

(iii) For each code in the family, there exists a polynomial amount of advice information given which there is a polynomial time list decoding algorithm that decodes the code up to a fraction $(1 - 1/q - \varepsilon)$ of errors.

Proof: We will employ the concatenated code construction of Theorem 8.3 applied with the outer code being AG-codes that meet the Drinfeld-Vlăduţ bound (as guaranteed by Fact 6.1). By picking m even, we know there exist AG-codes over $\mathrm{GF}(q^m)$ of relative designed distance δ' and rate $R \geq 1 - 1/(q^{m/2} - 1) - \delta'$. The fraction of errors corrected by the algorithm of Theorem 8.3 is $(1 - 1/q)(1 - \sqrt{1 - \delta'})$. Picking $\delta' = 1 - O(\varepsilon^2)$, we can get a list decoding radius of $(1 - 1/q - \varepsilon)$. For such a value of δ', the rate R of the AG-code can be $\Omega(\varepsilon^2)$, provided $q^{m/2} = \Omega(\varepsilon^{-2})$. This can be achieved with $m = \Theta(\log(1/\varepsilon))$ (since q is fixed, we absorb constant terms that depend on q into the Θ-notation). The rate of the concatenated code is the rate of the AG-code multiplied by the rate of the Hadamard code, and is thus $R \cdot (m/q^m)$. Since $R = \Omega(\varepsilon^2)$, $m = \Theta(\log(1/\varepsilon))$ and $q^m = O(\varepsilon^{-4})$, the rate is $\Omega(\varepsilon^6 \log(1/\varepsilon))$. □

8.5 Decoding a general concatenated code with outer Reed-Solomon or AG-code

The concatenated codes in the previous section used the Hadamard code as inner code. This permitted an elegant analysis of the decoding algorithms based on the combinatorial identity of Proposition 8.1 and the soft decoding algorithms from Chapter 6. However, the Hadamard code has very poor rate which makes these codes not so attractive from a coding theory viewpoint.

In this section, we present an algorithm to decode concatenated codes with outer Reed-Solomon or AG-codes when the inner code is an arbitrary q-ary code. The idea behind the decoding will remain the same (recall Figure 8-2) — in the first step, the inner decoding will pass weights which are linear functions of the distance between the received word and the concerned inner codeword. These weights will then be used in a soft decoding of the outer code. The key technical step in making this work when the inner code is not Hadamard but arbitrary is to prove an analog of the combinatorial bound of Proposition 8.1 for a general q-ary code. We do so next.

8.5.1 A relevant combinatorial result

To motivate the exact statement of the combinatorial result, we jump ahead to give a hint of how the decoding will exactly proceed. When presented a received word $\mathbf{r}$, the inner decoder will simply search for and output all codewords which lie in a Hamming ball of a certain radius R around $\mathbf{r}$. The weight associated with a codeword $\mathbf{c}$ at a distance $e_{\mathbf{c}} = \Delta(\mathbf{r}, \mathbf{c}) \leq R$ from $\mathbf{r}$ will be set to be $(R - e_{\mathbf{c}})$. These weights will be used in a soft decoding of the outer code as before. We now state and prove a combinatorial result that gives an upper bound on the sum of squares of the weights $(R - e_{\mathbf{c}})$. Some readers may prefer to take the result below on faith and jump right ahead to the decoding algorithm and its analysis in Section 8.5.2.

Proposition 8.5 *Let $C \subseteq [q]^n$ be a q-ary code (not necessarily linear), and let d be the minimum distance of C, and $\delta = d/n$ its relative distance. Let $\mathbf{r} \in [q]^n$ be arbitrary, and let*

$$R = n\Big(1 - \frac{1}{q}\Big)\Big(1 - \sqrt{1 - \frac{\delta}{(1 - 1/q)}}\Big) \tag{8.7}$$

be the q-ary Johnson radius of the code. Then we have

$$\sum_{\mathbf{c}\in C}\Big(\max\{(R-\Delta(\mathbf{r},\mathbf{c})),0\ \}\Big)^2 \le \delta n^2 \tag{8.8}$$

Proof: The proof follows essentially the same approach as in the proof of the Johnson bound (Theorems 3.1 and 3.2) from Chapter 3. Instead of bounding the number of codewords within a distance R from $\mathbf{r}$, we now require an upper bound on the sum of squares of linear functions of the distance over all such codewords. The proof will be identical to that of Theorem 3.1 for the most part, with a change towards the end. For purposes of readability, we give the full proof here. The reader familiar with the proof of Theorem 3.1 can jump to just past Equation (8.14) since the proof is identical till that stage.[2]

We identify elements of $[q]$ with vectors in $\mathbb{R}^q$ by replacing the symbol i $(1 \le i \le q)$ by the unit vector of length q with a 1 in position i. We then associate elements in $[q]^n$ with vectors in $\mathbb{R}^{nq}$ by writing down the vectors for each of the n symbols in sequence. This allows us to embed the codewords of C as well as the received word $\mathbf{r}$ into $\mathbb{R}^{nq}$. Let $\mathbf{c_1},\mathbf{c_2},\ldots,\mathbf{c_M}$ be all the codewords that satisfy $\Delta(\mathbf{r},\mathbf{c_i}) \le R$, where R is a parameter that will be set shortly (it will end up being set to the Johnson radius as in Equation (8.7)). By abuse of notation, let us denote by $\mathbf{c_i}$ also the nq-dimensional real vector associated with the codeword $\mathbf{c_i}$, for $1 \le i \le M$ (using the above mentioned identification), and by $\mathbf{r}$ the vector corresponding to $\mathbf{r} \in [q]^n$. Let $\mathbf{1} \in \mathbb{R}^{nq}$ be the all 1's vector. Now define $\mathbf{v} = \alpha\mathbf{r} + \frac{(1-\alpha)}{q}\mathbf{1}$ for a parameter $0 \le \alpha \le 1$ to be specified later in the proof.

The idea behind the rest of the proof is the following. We will pick α so that the nq-dimensional vectors $\mathbf{d_i} = (\mathbf{c_i} - \mathbf{v})$, for $1 \le i \le M$, have all pairwise dot products less than 0. Geometrically speaking, we shift the origin O to O' where $OO' = \mathbf{v}$, and require that relative to the new origin the vectors corresponding to the codewords have pairwise angles which are greater than 90 degrees. We will then exploit the geometric fact that for such vectors $\mathbf{d_i}$, for any vector $\mathbf{w}$, the sum of the squares of its projections along the $\mathbf{d_i}$'s is at most $\langle \mathbf{w},\mathbf{w}\rangle$ (this is proved in Lemma 8.6). This will then give us the required bound (8.8).

For $1 \le i \le M$, let $e_i = \Delta(\mathbf{r},\mathbf{c_i})$ be the Hamming distance between $\mathbf{c_i}$ and $\mathbf{r}$. Note by the way we associate vectors with elements of $[q]^n$, we have $\langle \mathbf{c_i},\mathbf{r}\rangle = n - e_i$. Now

$$\begin{aligned}
\langle \mathbf{c_i},\mathbf{v}\rangle &= \alpha\langle \mathbf{c_i},\mathbf{r}\rangle + \frac{(1-\alpha)}{q}\langle \mathbf{c_i},\mathbf{1}\rangle = \alpha(n-e_i) + (1-\alpha)\frac{n}{q} && (8.9)\\
\langle \mathbf{v},\mathbf{v}\rangle &= \alpha^2 n + 2(1-\alpha)\alpha\frac{n}{q} + (1-\alpha)^2\frac{n}{q} = \frac{n}{q} + \alpha^2\Big(1-\frac{1}{q}\Big)n && (8.10)\\
\langle \mathbf{c_i},\mathbf{c_j}\rangle &= n - \Delta(\mathbf{c_i},\mathbf{c_j}) \le n-d\ . && (8.11)
\end{aligned}$$

Using (8.9), (8.10) and (8.11), we get for $i \ne j$

$$\begin{aligned}
\langle \mathbf{d_i},\mathbf{d_j}\rangle = \langle \mathbf{c_i}-\mathbf{v},\mathbf{c_j}-\mathbf{v}\rangle &\le \alpha e_i + \alpha e_j - d + \Big(1-\frac{1}{q}\Big)(1-\alpha)^2 n\\
&\le 2\alpha R - d + \Big(1-\frac{1}{q}\Big)(1-\alpha)^2 n && (8.12)
\end{aligned}$$

Hence we have $\langle \mathbf{d_i},\mathbf{d_j}\rangle \le 0$ as long as

$$R \le (1-1/q)n - \Big((1-1/q)\frac{\alpha n}{2} + \frac{(1-1/q)n-d}{2\alpha}\Big)\ .$$

[2]We prove this result here and not in Chapter 3 due to the local nature of its context and use.

Picking $\alpha = \sqrt{1 - \frac{d/n}{(1-1/q)}} = \sqrt{1 - \frac{\delta}{(1-1/q)}}$ maximizes the "radius" R for which our bound will apply. Hence we pick

$$\alpha = \Big(1 - \frac{\delta}{(1-1/q)}\Big)^{1/2} . \tag{8.13}$$

and

$$R = n\Big(1 - \frac{1}{q}\Big)\Big(1 - \sqrt{1 - \frac{\delta}{(1-1/q)}}\Big) = n(1 - \frac{1}{q})(1-\alpha) . \tag{8.14}$$

For this choice of α, R, we have $\langle \mathbf{d_i}, \mathbf{d_j} \rangle \le 0$ for every $1 \le i < j \le M$. Now a simple geometric fact, proved in Lemma 8.6 at the end of this proof, implies that for any vector $\mathbf{x} \in \mathbb{R}^{nq}$ that satisfies $\langle \mathbf{x}, \mathbf{d_i} \rangle \ge 0$ for $i = 1, 2, \ldots, M$, we have

$$\sum_{i=1}^{M} \frac{\langle \mathbf{x}, \mathbf{d_i} \rangle^2}{\langle \mathbf{d_i}, \mathbf{d_i} \rangle} \le \langle \mathbf{x}, \mathbf{x} \rangle . \tag{8.15}$$

We will apply this to the choice $\mathbf{x} = \mathbf{r}$. Straightforward computations show that

$$\begin{aligned} \langle \mathbf{r}, \mathbf{r} \rangle &= n && (8.16)\\ \langle \mathbf{d_i}, \mathbf{d_i} \rangle &= \langle \mathbf{c_i} - \mathbf{v}, \mathbf{c_i} - \mathbf{v} \rangle = 2\alpha e_i + (1-\alpha)^2 (1 - \frac{1}{q}) n && (8.17)\\ \langle \mathbf{r}, \mathbf{d_i} \rangle &= (1-\alpha)\Big(1 - \frac{1}{q}\Big) n - e_i = R - e_i . && (8.18) \end{aligned}$$

Since each $e_i \le R$, we have $\langle \mathbf{r}, \mathbf{d_i} \rangle \ge 0$ for each i, $1 \le i \le M$, and therefore we can apply Equation (8.15) above. For $1 \le i \le M$, define

$$W_i = \frac{\langle \mathbf{r}, \mathbf{d_i} \rangle}{\sqrt{\langle \mathbf{d_i}, \mathbf{d_i} \rangle}} = \frac{R - e_i}{\sqrt{2\alpha e_i + (1-\alpha) R}} \tag{8.19}$$

(the second step follows using (8.14), (8.17) and (8.18)). Since each $e_i \le R$, we have

$$W_i = \frac{R - e_i}{\sqrt{2\alpha e_i + (1-\alpha) R}} \ge \frac{R - e_i}{\sqrt{(1+\alpha) R}} = \frac{R - e_i}{\sqrt{\delta n}} , \tag{8.20}$$

where the last equality follows by substituting the values of α and R from (8.13) and (8.14). Now combining (8.16), (8.17) and (8.18), and applying Equation (8.15) to the choice $\mathbf{x} = \mathbf{r}$, we get

$$\sum_{i=1}^{M} W_i^2 \le n . \tag{8.21}$$

Now from (8.20) and (8.21) it follows that

$$\sum_{i=1}^{M} (R - \Delta(\mathbf{r}, \mathbf{c_i}))^2 \le \delta n^2 . \tag{8.22}$$

This clearly implies the bound (8.8) claimed in the statement of the proposition, since the codewords $\mathbf{c_i}$, $1 \le i \le M$, include *all* codewords $\mathbf{c}$ that satisfy $\Delta(\mathbf{r}, \mathbf{c}) \le R$, and the remaining codewords contribute zeroes to the left hand side of Equation (8.8). □

We now prove the geometric fact that was used in the above proof. Once again the reader should feel to skip its proof and move on to the decoding algorithm in the next section, since there is no harm taking its statement on faith.

Lemma 8.6 *Let* $\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_M}$ *be* distinct *unit vectors in* $\mathbb{R}^N$ *such that* $\langle \mathbf{v_i}, \mathbf{v_j} \rangle \leq 0$ *for* $1 \leq i < j \leq M$. *Further, suppose* $\mathbf{x} \in \mathbb{R}^N$ *is a vector such that* $\langle \mathbf{x}, \mathbf{v_i} \rangle \geq 0$ *for each* i, $1 \leq i \leq M$. *Then*

$$\sum_{i=1}^{m} \langle \mathbf{x}, \mathbf{v_i} \rangle^2 \leq \langle \mathbf{x}, \mathbf{x} \rangle \tag{8.23}$$

Proof: Note that if $\langle \mathbf{v_i}, \mathbf{v_j} \rangle = 0$ for every $i \neq j$, then the $\mathbf{v_i}$'s form a linearly independent set of pairwise orthogonal unit vectors. They may thus be extended to an orthonormal basis. The bound (8.23) then holds since the sum of squares of projection of a vector on vectors in an orthonormal basis *equals* the square of its norm, and hence the sum of squares when restricted to the $\mathbf{v_i}$'s cannot be larger than $\langle \mathbf{x}, \mathbf{x} \rangle$. We need to show this holds even if the $\mathbf{v_i}$'s are more than 90 degrees apart.

Firstly, we can assume $\langle \mathbf{x}, \mathbf{v_i} \rangle > 0$ for $i = 1, 2, \ldots, M$. This is because if $\langle \mathbf{x}, \mathbf{v_i} \rangle = 0$, then it does not contribute to the left hand side of Equation (8.23) and may therefore be discarded. In particular, this implies that we may assume $(\mathbf{v_i} \neq -\mathbf{v_j})$ for any $1 \leq i, j \leq M$. Since the $\mathbf{v_i}$'s are distinct unit vectors, this means that $|\langle \mathbf{v_i}, \mathbf{v_j} \rangle| < 1$ for all $i \neq j$.

We will prove the claimed bound (8.23) by induction on M. When $M = 1$ the result is obvious. For $M > 1$, we will project the vectors $\mathbf{v_1}, \ldots, \mathbf{v_{M-1}}$, and also $\mathbf{x}$, onto the space orthogonal to $\mathbf{v_M}$. We will then apply the induction hypothesis to the projected vectors and conclude our final bound using the analog of (8.23) for the set of projected vectors. The formal details follow.

For $1 \leq i \leq M-1$, define $\mathbf{v'_i} = \mathbf{v_i} - \langle \mathbf{v_i}, \mathbf{v_M} \rangle \mathbf{v_M}$. Since $\mathbf{v_i}$ is different from $\mathbf{v_M}$ and $-\mathbf{v_M}$, each $\mathbf{v'_i}$ is a non-zero vector. Let $\mathbf{u_i}$ be the unit vector associated with $\mathbf{v'_i}$. Let us also define $\mathbf{x'} = \mathbf{x} - \langle \mathbf{x}, \mathbf{v_M} \rangle \mathbf{v_M}$. We wish to apply the induction hypothesis to the vectors $\mathbf{u_1}, \ldots, \mathbf{u_{M-1}}$ and $\mathbf{x'}$.

Now, for $1 \leq i < j \leq M-1$, we have $\langle \mathbf{v'_i}, \mathbf{v'_j} \rangle = \langle \mathbf{v_i}, \mathbf{v_j} \rangle - \langle \mathbf{v_i}, \mathbf{v_M} \rangle \langle \mathbf{v_j}, \mathbf{v_M} \rangle \leq \langle \mathbf{v_i}, \mathbf{v_j} \rangle \leq 0$, since all pairwise dot products between the $\mathbf{v_i}$'s are non-positive. Hence the pairwise dot products $\langle \mathbf{u_i}, \mathbf{u_j} \rangle$, $1 \leq i < j \leq M-1$, are all non-positive. To apply the induction hypothesis we should also verify that $\langle \mathbf{x'}, \mathbf{u_i} \rangle > 0$ for $i = 1, 2, \ldots, (M-1)$. It will be enough to verify that $\langle \mathbf{x'}, \mathbf{v'_i} \rangle > 0$ for each i. But this is easy to check since

$$\begin{aligned} \langle \mathbf{x'}, \mathbf{v'_i} \rangle &= \langle \mathbf{x}, \mathbf{v_i} \rangle - \langle \mathbf{x}, \mathbf{v_M} \rangle \cdot \langle \mathbf{v_i}, \mathbf{v_M} \rangle \\ &\geq \langle \mathbf{x}, \mathbf{v_i} \rangle \qquad (8.24) \\ &> 0 \end{aligned}$$

where (8.24) follows since $\langle \mathbf{x}, \mathbf{v_M} \rangle > 0$ and $\langle \mathbf{v_i}, \mathbf{v_M} \rangle \leq 0$.

We can therefore apply the induction hypothesis to the $(M-1)$ unit vectors $\mathbf{u_1}, \mathbf{u_2}, \ldots, \mathbf{u_{M-1}}$ and the vector $\mathbf{x'}$. This gives

$$\sum_{i=1}^{M-1} \langle \mathbf{x'}, \mathbf{u_i} \rangle^2 \leq \langle \mathbf{x'}, \mathbf{x'} \rangle \ . \tag{8.25}$$

Now, $\|\mathbf{v'_i}\|^2 = \langle \mathbf{v'_i}, \mathbf{v'_i} \rangle = \langle \mathbf{v_i}, \mathbf{v_i} \rangle - \langle \mathbf{v_i}, \mathbf{v_M} \rangle^2 \leq \|\mathbf{v_i}\|^2 = 1 = \|\mathbf{u_i}\|^2$. This implies that $\langle \mathbf{x'}, \mathbf{v'_i} \rangle \leq \langle \mathbf{x'}, \mathbf{u_i} \rangle$, for $1 \leq i \leq M-1$. Also, by (8.24) $\langle \mathbf{x'}, \mathbf{v'_i} \rangle \geq \langle \mathbf{x}, \mathbf{v_i} \rangle$, and therefore

$$\langle \mathbf{x}, \mathbf{v_i} \rangle \leq \langle \mathbf{x'}, \mathbf{u_i} \rangle \ , \tag{8.26}$$

for $i = 1, 2, \ldots, (M-1)$. Also, we have

$$\langle \mathbf{x'}, \mathbf{x'} \rangle = \langle \mathbf{x}, \mathbf{x} \rangle - \langle \mathbf{x}, \mathbf{v_M} \rangle^2 \ . \tag{8.27}$$

The claimed result now follows by using (8.26) and (8.27) together with the inequality (8.25). □

8.5.2 The formal decoding algorithm and its analysis

We are now ready to state and prove our result about decoding concatenated codes with a general inner code.

Theorem 8.7 *Consider a family of linear q-ary concatenated codes where the outer codes belong to a family of Reed-Solomon codes of relative distance Δ over a field of size at most polynomial in the blocklength, and the inner codes belong to any family of q-ary linear codes of relative distance δ. There is a polynomial time decoding procedure to list decode codes from such a family up to a fractional radius of*

$$\Big(1-\frac{1}{q}\Big)\Big(1-\sqrt{1-\frac{q\delta}{q-1}}\Big)-\sqrt{\delta(1-\Delta)}\ . \tag{8.28}$$

Proof: (Sketch) Consider a concatenated code C with outer code a Reed-Solomon code over $\mathrm{GF}(q^m)$ of blocklength n_0, relative distance Δ and dimension $(1-\Delta)n_0+1$. We assume $q^m \le n_0^{O(1)}$, so that the field over which the Reed-Solomon code is defined is of size polynomial in the blocklength. Let the inner code $C_{\rm in}$ be any q-ary linear code of dimension m, blocklength n_1 and relative distance δ. Messages of C correspond to polynomials of degree at most $k_0=(1-\Delta)n_0$ over $\mathrm{GF}(q^m)$, and a polynomial p is encoded as $\langle C_{\rm in}(p(x_1)),\dots,C_{\rm in}(p(x_{n_0}))\rangle$ where $x_1,x_2,\dots,x_{n_0}$ are distinct elements of $\mathrm{GF}(q^m)$ that are used to define the Reed-Solomon encoding.

The proof parallels that of the earlier result (Theorem 8.2) where the inner code was the Hadamard code. Let $y\in\mathbb{F}_q^n$ be a received word. For $1\le i\le n_0$, denote by y_i the portion of y in block i of the codeword (namely, the portion corresponding to the encoding by $C_{\rm in}$ of the $i^{\rm th}$ symbol of the outer Reed-Solomon code).

We now perform the "decoding" of each of the n_0 blocks y_i as follows. Let

$$R=n_1(1-1/q)\Big(1-\sqrt{1-\frac{q\delta}{q-1}}\Big) \tag{8.29}$$

be the Johnson radius of the inner code $C_{\rm in}$. For $1\le i\le n_0$ and $\alpha\in\mathrm{GF}(q^m)$, compute the Hamming distance $e_{i,\alpha}$ between y_i and the codeword $C_{\rm in}(\alpha)$, and then compute the *weight* $w_{i,\alpha}$ as:

$$w_{i,\alpha}\stackrel{\rm def}{=}\max\{(R-e_{i,\alpha}),0\}\ . \tag{8.30}$$

Note the computation of all these weights can be done by a straightforward brute-force computation in $O(n_0n_1q^m)=O(n_1n_0^{O(1)})=\mathrm{poly}(n)$ time. Thus all the inner decodings can be performed efficiently in polynomial time.

By Proposition 8.5 applied to the y_i's, for $1\le i\le n_0$, we know that the above weights have the crucial combinatorial property

$$\sum_\alpha w_{i,\alpha}^2\le\delta n_1^2\ , \tag{8.31}$$

for $i=1,2,\dots,n_0$. We will then run the soft decoding algorithm for Reed-Solomon codes from Theorem 6.21 for this choice of weights. Now, arguing exactly as in the proof of

Theorem 8.2 that and using (8.31) above, we conclude that we can find in time polynomial in n and $1/\epsilon$, a list of all polynomials p over $\mathrm{GF}(q^m)$ of degree at most k_0 for which the condition

$$\sum_{i=1}^{n_0}(R - e_{i,p(x_i)}) \geq \sqrt{k_0 n_0 \delta n_1^2} + \epsilon n_1 \tag{8.32}$$

holds. Recalling the definition of R (Equation (8.29)) and using $k_0 = (1-\Delta)n_0$, we conclude that we can find a list of all codewords that are at a Hamming distance of at most

$$n(1-\frac{1}{q})\Big(1-\sqrt{1-\frac{q\delta}{q-1}}\Big) - n\sqrt{\delta(1-\Delta)} - \epsilon n_1 \ ,$$

from y. Picking $\epsilon < 1/n_1$, we get decoding up to the claimed fraction of errors. $\Box$

Comment on the error-correction performance of above: The bound of (8.28) is attractive only for very large values of Δ, or in other words when the rate of the outer Reed-Solomon code is rather small. For example, for the binary case $q = 2$, even for $\Delta = 3/4$, the bound does not even achieve the product bound (namely, $\Delta\delta/2$), for *any* value of δ in the range $0 < \delta < 1/2$ (in fact, the bound as stated in (8.28) is negative unless Δ is quite large). However, the merit of the bound is that as Δ gets very close to 1, the bound (8.28) approaches the quantity $(1-1/q)(1-\sqrt{1-\frac{q\delta}{q-1}})$, and since the relative designed distance of the concatenated code is $\Delta\cdot\delta \to \delta$, it approaches the Johnson bound on list decoding radius. Therefore for $\Delta \to 1$, the result of Theorem 8.7 performs very well and decodes almost up to the Johnson bound, and hence beyond the product bound, for almost the entire range of the inner code distances $0 < \delta < 1/2$. In particular, for $\Delta \to 1$ and $\delta \to (1-1/q)$, the bound tends to $(1-1/q)$, permitting us to list decode up to close to the maximum possible fraction $(1-1/q)$ of errors.

Alternative decoding bound

By slightly modifying the analysis used in proving the combinatorial bound of Proposition 8.5, one can prove the following alternative bound instead of (8.8).

$$\sum_{\mathbf{c}\in C}\Big(\max\Big\{\Big(1-\frac{\Delta(\mathbf{r},\mathbf{c})}{\tilde{R}}\Big),0\ \Big\}\Big)^2 \leq \frac{q}{q-1} \ , \tag{8.33}$$

where we use the same notation as in the statement of Proposition 8.5 and $\tilde{R}$ is defined as

$$\tilde{R} \stackrel{\text{def}}{=} \left(1-\sqrt{1-\frac{q\delta}{q-1}}\right)^2 \Big(1-\frac{1}{q}\Big)n \ .$$

(The only change required in the proof is to replace the lower bound on W_i from Equation (8.20) with the alternative lower bound $W_i \geq (1-\frac{e_i}{\tilde{R}})\sqrt{n(q-1)/q}$, which follows easily from the definition of W_i in Equation (8.19).)

Now, replacing the choice of weights in Equation (8.30) in the proof of Theorem 8.7 by

$$w_{i,\alpha} \stackrel{\text{def}}{=} \max\Big\{\Big(1-\frac{e_{i,\alpha}}{\tilde{R}}\Big),0\Big\} \ ,$$

and then using (8.33), we obtain a decoding algorithm to decode up to a fraction

$$\left(1-\frac{1}{q}\right)\left(1-\sqrt{1-\frac{q\delta}{q-1}}\right)^2\left(1-\sqrt{\frac{1-\Delta}{(1-1/q)}}\right) \tag{8.34}$$

of errors. This bound is positive whenever $\Delta > 1/q$, and in general appears incomparable to that of (8.28). However, note that even for Δ very close to 1, the bound (8.34) does not approach the Johnson bound, except for δ very close to $(1-1/q)$. But as with the bound (8.28), for $\Delta \to 1$ and $\delta \to (1-1/q)$, the above tends to a fraction $(1-1/q)$ of errors. In particular, it can also be used, instead of (8.28), to obtain the results outlined in the next section for highly list decodable codes.

8.5.3 Consequence for highly list decodable codes

We now apply Theorem 8.7 with a suitable choice of parameters to obtain an alternative construction of codes list decodable to a fraction $(1-1/q-\varepsilon)$ of errors and which have rate $\Omega(\varepsilon^6)$. Compared to the construction of Corollary 8.4 that was based on a concatenation of AG-codes with Hadamard codes, the rate is slightly worse – namely by a factor of $O(\log(1/\varepsilon))$. But the following construction offers several advantages compared to that of Corollary 8.4. Firstly, it is based on outer Reed-Solomon codes, and hence does not suffer from the high construction and decoding complexity of AG-codes. In particular, the claim of polynomial time decoding is unconditional and does not depend on having access to precomputed advice information about the outer code. Secondly, the inner code can be *any* linear code of large minimum distance, and not necessarily the Hadamard code. In fact, picking a random code as inner code will give a highly efficient probabilistic construction of the code that has the desired list decodability properties with high probability.

In the next section (Section 8.6) we will present a construction of highly list decodable codes of rate $\Omega(\varepsilon^4)$. Even with this substantial improvement, the bound proved in this section is not strictly subsumed. This is for two reasons. Firstly, the results of Section 8.6 apply only to *binary* linear codes, where as the result below applies to linear codes over any finite field $\mathbb{F}_q$. Secondly, while the deterministic construction complexity of both the constructions in this section and the one with rate $\Omega(\varepsilon^4)$ are almost similar (both of them being fairly high), the codes of this section have very efficient probabilistic constructions, where as we do not know a faster probabilistic construction for the codes of Section 8.6. In conclusion, despite the improvement in rate that will be obtained in Section 8.6, the construction presented next remains interesting.

Theorem 8.8 *For every fixed prime power q, the following holds: For every $\varepsilon > 0$, there exists a family of linear codes over $\mathbb{F}_q$ with the following properties:*

(i) A description of a code of blocklength, say n, in the family can be constructed deterministically in $n^{O(1/\varepsilon^4)}$ time. For probabilistic constructions, a Las Vegas construction can be obtained in time which with high probability will be $O(n \log n/\varepsilon^4)$, or a Monte Carlo construction that has the claimed properties with high probability can be obtained in $O(\log n/\varepsilon^4)$ time.

(ii) Its rate is $\Omega(\varepsilon^6)$ and its relative minimum distance is $(1-1/q-O(\varepsilon^2))$.

(iii) There is a polynomial time list decoding algorithm for every code in the family to perform list decoding up to a fraction $(1-1/q-\varepsilon)$ of errors.

Proof: We will use Theorem 8.7 with the choice of parameters $\Delta = 1 - O(\varepsilon^2)$ and $\delta = 1 - 1/q - O(\varepsilon^2)$. Substituting in the bound (8.28), the fraction of errors corrected by the decoding algorithm from Section 8.5.2 will be $(1 - 1/q - \varepsilon)$, which handles Property (iii) claimed above. Also, the relative distance of the code is at least $\Delta \cdot \delta$, and is thus $(1 - 1/q - O(\varepsilon^2))$, verifying the distance claim in (ii) above. The outer Reed-Solomon code has rate $1 - \Delta = \Omega(\varepsilon^2)$. For the inner code, if we pick a random linear code, then it will meet the Gilbert-Varshamov bound ($R = 1 - H_q(\delta)$) with high probability (cf. [vL99, Chapter 5]). Therefore, a random inner code of rate $\Omega(\varepsilon^4)$ will have relative distance $\delta = 1 - 1/q - O(\varepsilon^2)$, exactly as we desire. The overall rate of the concatenated code is just the product of the rates of the Reed-Solomon code and the inner code, and is thus $\Omega(\varepsilon^2 \cdot \varepsilon^4) = \Omega(\varepsilon^6)$, proving Property (ii).

We now turn to Property (i) about the complexity of constructing the code. We may pick the outer Reed-Solomon code over a field of size at most $O(n)$. Hence, the inner code has at most $O(n)$ codewords and thus dimension at most $O(\log_q n)$. The inner code can be specified by its $O(\log_q n) \times O(\log_q n/\varepsilon^4)$ generator matrix G. To construct an inner code that has relative distance $(1 - 1/q - O(\varepsilon^2))$, we can pick such a generator matrix G at *random*, and then check, by a brute-force search over the at most $O(n)$ codewords, that the code has the desired distance. Since the distance property holds with high probability, we conclude that the generator matrix an inner code with the required rate and distance property can be found in $O(n \log^2 n/\varepsilon^4)$ time with high probability. Allowing for a small probability for error, a Monte Carlo construction can be obtained in $O(\log^2 n/\varepsilon^4)$ probabilistic time by picking a random linear code as inner code (the claimed distance and list decodability properties (ii), (iii) will then hold with high probability). As the outer Reed-Solomon code is explicitly specified, this implies that the description of the concatenated code can be found within the same time bound.

A naive derandomization of the above procedure will require time which is quasi-polynomial in n. But the construction time can be made polynomial by reducing the size of the sample space from which the inner codes is picked. For this, we note that, for every prime power q, there is a small sample space of q-ary linear codes of any desired rate, called a "Wozencraft ensemble" in the literature, with the properties that: (a) a random code can be drawn from this family using a linear (in the blocklength) number of random elements from $\mathbb{F}_q$, and (b) such a code will meet the Gilbert-Varshamov bound with high probability. We record this fact together with a proof as Proposition 8.10 at the end of this section. Applying Proposition 8.10 for the choice of parameters $b = O(\varepsilon^{-4})$, $k = O(\log_q n)$, and using the fact that for small γ, $H_q^{-1}(1 - O(\gamma^2))$ is approximately $(1 - 1/q - O(\gamma))$, we obtain a sample space of linear codes of size $q^{O(\log_q n/\varepsilon^4)} = n^{O(1/\varepsilon^4)}$ which includes a code of rate $\Omega(\varepsilon^4)$ and relative distance $(1 - 1/q - O(\varepsilon^2))$. One can simply perform a brute-force search for the desired code in such a sample space. Thus one can find an inner code of rate $\Omega(\varepsilon^4)$ and relative distance $(1 - 1/q - O(\varepsilon^2))$ deterministically in $n^{O(1/\varepsilon^4)}$ time. Moreover, picking a random code from this sample space, which works just as well as picking a general random linear code, takes only $O(\log n/\varepsilon^4)$ time. This reduces the probabilistic construction times claimed earlier by a factor of $\log n$. Hence a description of the overall concatenated code can be obtained within the claimed time bounds. This completes the verification of Property (i) as well. □

Obtaining an explicit construction: The high deterministic construction complexity or the probabilistic nature of construction in Theorem 8.8 can be removed at the expense of a slight worsening of the rate of the code. One can pick for inner code an *explicitly specified*

q-ary code of relative distance $(1 - 1/q - O(\varepsilon^2))$ and rate $\Omega(\varepsilon^6)$. A fairly simple explicit construction of such codes is known [ABN+92] (see also [She93]). This will give an *explicit construction* of the overall concatenated code with rate $\Omega(\varepsilon^8)$. We record this below.

Theorem 8.9 *For every fixed prime power q, the following holds: For every $\varepsilon > 0$, there exists a family of* explicitly specified *linear codes over $\mathbb{F}_q$ with the following properties:*

(i) Its rate is $\Omega(\varepsilon^8)$ and its relative minimum distance is $(1 - 1/q - O(\varepsilon^2))$.

(ii) There is a polynomial time list decoding algorithm for every code in the family to perform list decoding up to a fraction $(1 - 1/q - \varepsilon)$ of errors.

A small space of linear codes meeting the Gilbert-Varshamov bound

We now turn to the result about a small space of linear codes meeting the Gilbert-Varshamov bound. Such an ensemble of codes is referred to as a "Wozencraft ensemble" in the literature. Recall that we made use of such a result in the proof of Theorem 8.8.

Proposition 8.10 (cf. [Wel73]) *For every prime power q, and every integer $b \geq 1$, the following holds. For all large enough k, there exists a sample space, denoted $S_q(b, n)$ where $n \stackrel{\text{def}}{=} (b+1)k$, consisting of $[n, k]_q$ linear codes of rate $1/(b+1)$ such that:*

(i) There are at most $q^{bn/(b+1)}$ codes in $S_q(b, n)$. In particular, one can pick a code at random from $S_q(b, n)$ using at most $O(n \log q)$ random bits.

(ii) A random code drawn from $S_q(b, n)$ meets the Gilbert-Varshamov bound, i.e. has minimum distance $n \cdot H_q^{-1}(\frac{b}{b+1} - o(1))$, with overwhelming (i.e. $1 - o(1)$) probability.

Proof: The fact that a code that meets the Gilbert-Varshamov bound can be picked by investing a linear amount of randomness is by now a folklore result. The proof we present here follows the construction due to Weldon [Wel73], which in turn was a generalization of a construction for the rate 1/2 case that Justesen used in the first explicit construction of a family of asymptotically good binary codes [Jus72].

Let α be a primitive element of the finite field $\mathrm{GF}(q^k)$, so that $\{\alpha^i : 0 \leq i < q^k - 1\}$ are all the non-zero elements of $\mathrm{GF}(q^k)$. A code in $S_q(b, n)$ will be specified by a b-tuple $\mathcal{I}_b = (i_0, i_1, \ldots, i_{b-1})$ where each i_s, $0 \leq s \leq b-1$, is an integer that satisfies $0 \leq i_s \leq q^k - 1$. Note that there are $q^{kb} = q^{bn/(b+1)}$ codes in the sample space $S_q(b, n)$, since there are exactly so many b-tuples. A random code in $S_q(b, n)$ can be picked by choosing a random b-tuple $\mathcal{I}_b$. Hence the sample space $S_q(b, n)$ meets the requirement (i).

A message $\mathbf{a} \in \mathbb{F}_q^k$, will be encoded by a code indexed by a b-tuple $(i_0, i_1, \ldots, i_{b-1})$ as follows: view $\mathbf{a}$ as a field element $\gamma \in \mathrm{GF}(q^k)$ (using some fixed representation of $\mathrm{GF}(q^k)$ over $\mathrm{GF}(q)$), then encode it as $\langle \gamma, \gamma\alpha^{i_0}, \gamma\alpha^{i_1}, \ldots, \gamma\alpha^{i_{b-1}} \rangle$. This gives a $(b+1)$-tuple over $\mathrm{GF}(q^k)$ or equivalently a word of length $(b+1)k = n$ over $\mathrm{GF}(q)$, as desired.

The crucial observation used to prove (ii) is the following. Any non-zero vector $\mathbf{v} \in \mathbb{F}_q^n$ can belong to at most one of the codes in $S_q(b, n)$. Indeed, it is easily checked that the b-tuple associated with a code containing $\mathbf{v}$ can be uniquely reconstructed from $\mathbf{v}$. Property (ii) is now a simple consequence of this fact. Indeed, the number of vectors $\mathbf{v} \in \mathbb{F}_q^n$ of Hamming weight at most w is at most $q^{H_q(w/n)n}$ (see for example [vL99, Chapter 1]). Applying this to $w = d \stackrel{\text{def}}{=} n \cdot H_q^{-1}(\frac{b}{b+1} - \zeta)$, the number of vectors of Hamming weight less than or equal to

d is at most $q^{(\frac{b}{b+1}-\zeta)n}$. Since a non-zero vector belongs to at most one code among those in $S_q(b,n)$, this implies that the fraction of codes in $S_q(b,n)$ that have some non-zero codeword of weight less than or equal to d is at most $q^{-\zeta n}$. Picking $\zeta = o(1)$, say $1/\sqrt{n}$, we conclude that a random code from $S_q(b,n)$ has minimum distance greater than $n \cdot H_q^{-1}(\frac{b}{b+1} - o(1))$ with very high (i.e., $(1 - o(1))$) probability. □

8.6 Improved rate using tailor-made concatenated code

We now proceed to a construction of highly list decodable codes that improves over the rate of ε^6 that was achieved by Theorem 8.8 (and also by Corollary 8.4). The results of this section apply *only* to binary linear codes. Recall that binary codes that can be list decoded from $(1/2 - \varepsilon)$ errors using polynomial sized lists can have rate at best $\Omega(\varepsilon^2)$. We will be able to attain a rate of $\Omega(\varepsilon^4)$. The formal result is stated below.

Theorem 8.11 *There exist absolute constants $b, d > 0$ such that for each fixed $\varepsilon > 0$, there exists a polynomial time constructible* binary linear *code family $\mathcal{C}$ with the following properties:*

1. *A code of blocklength N from the family $\mathcal{C}$ can be constructed in $N^{O(1/\varepsilon^2)}$ time deterministically.*
2. *The rate $R(\mathcal{C})$ of $\mathcal{C}$ is at least $\frac{\varepsilon^4}{b}$, and its relative distance $\delta(\mathcal{C})$ is at least $(1/2 - \varepsilon)$.*
3. *There is a* polynomial time list decoding algorithm *that can list decode codes in $\mathcal{C}$ from up to a fraction $(1/2 - \varepsilon)$ of errors, using lists of size at most d/ε^2.* □

The above theorem will follow from Theorem 8.14, which is stated and proved in Section 8.6.2. The basic idea is to use a concatenated code with the outer code being a Reed-Solomon code and the inner code being a "tailor-made" one. The inner code will be chosen so that it possesses a rather peculiar looking combinatorial property, which is formalized in Lemma 8.12. This property will be very useful when it is used in conjunction with the soft decoding algorithm for Reed-Solomon codes (Theorem 6.21). We first turn to the existence and construction of the necessary inner code.

8.6.1 The inner code construction

Existence of a "good" code

We now prove the existence of codes that will serve as excellent inner codes in our later concatenated code construction. The proof is an adaptation of that of Theorem 5.5. We will then show how such a code can be constructed in $2^{O(n)}$ time (where n is the blocklength) using an iterative greedy procedure.

Lemma 8.12 *There exist absolute constants $\sigma, A > 0$ such that for any $\varepsilon > 0$ there exists a binary linear code family $\mathcal{C}$ with the following properties:*

1. *The rate of the family satisfies $R(\mathcal{C}) = \sigma\varepsilon^2$*
2. *For every code $C \in \mathcal{C}$ and every $x \in \{0,1\}^n$ where n is the blocklength of C, we have*

$$\sum_{\substack{\mathbf{c} \in C \\ \Delta(x,\mathbf{c}) \le (1/2-\varepsilon)n}} \left(1 - \frac{2\Delta(x,\mathbf{c})}{n}\right)^2 \le A \ . \tag{8.35}$$

Proof: For every large enough n, we will prove the existence of a binary linear code C_k of blocklength n and dimension $k \geq \sigma\varepsilon^2 n$ which satisfies Condition (8.35) for every $x \in \{0,1\}^n$.

The proof will follow very closely the proof of Theorem 5.5 and in particular we will again build the code C_k iteratively in k steps by randomly picking the k linearly independent basis vectors $b_1, b_2, \ldots, b_k$ in turn. Define $C_i = \text{span}(b_1, \ldots, b_i)$ for $1 \leq i \leq k$ (and define $C_0 = \{\mathbf{0}\}$). The key to our proof is the following potential function W_C defined for a code C of blocklength n (compare with the potential function (5.12) from the proof of Theorem 5.5):

$$W_C = \frac{1}{2^n} \sum_{x \in \{0,1\}^n} \exp_2 \left(\frac{n}{A} \cdot \sum_{\mathbf{c} \in C : \Delta(x,\mathbf{c}) \leq (1/2-\varepsilon)n} \left(1 - \frac{2\Delta(x,\mathbf{c})}{n}\right)^2 \right) , \tag{8.36}$$

where, for readability, we used $\exp_2(z)$ to denote 2^z. (The constant A will be fixed later in the proof, and we assume that $A > \ln 4$.) Denote the random variable W_{C_i} by the shorthand W_i. For $x \in \{0,1\}^n$, define

$$R_x^i = \sum_{\substack{\mathbf{c} \in C_i \\ \Delta(x,\mathbf{c}) \leq (1/2-\varepsilon)n}} \left(1 - \frac{2\Delta(x,\mathbf{c})}{n}\right)^2 , \tag{8.37}$$

so that

$$W_i = 2^{-n} \sum_x \exp_2 \left(\frac{n}{A} \cdot R_x^i \right) .$$

Now, exactly as in the proof of Theorem 5.5, we have $R_x^{i+1} = R_x^i + R_{x+b_{i+1}}^i$ when b_{i+1} is picked outside the span of $\{b_1, b_2, \ldots, b_i\}$. Now, arguing as in the proof of Theorem 5.5, one can deduce that

$$\mathbf{E}[W_{i+1} | W_i = \hat{W}_i] \leq \frac{\hat{W}_i^2}{1 - 2^{i-n}} . \tag{8.38}$$

when the expectation is taken over a random choice of b_{i+1} outside $\text{span}(b_1, \ldots, b_i)$. Applying (8.38) repeatedly for $i = 0, 1, \ldots, k-1$, we conclude that there exists an $[n,k]_2$ binary linear code $C = C_k$ with

$$W_C = W_k \leq \frac{W_0^{2^k}}{1 - k2^{k-n}} . \tag{8.39}$$

If we could prove, for example, that $W_C = O(1)$, then this would imply, using (8.36), that $R_x^k \leq A$ for every $x \in \{0,1\}^n$ and thus C would satisfy Condition (8.35), as desired. To show this, we need an estimate of (upper bound on) W_0, to which we turn next.

Define $a = (1/2 - \varepsilon)n$. Since C_0 consists of only the all-zeroes codeword, we have $R_x^0 = (1 - 2\text{wt}(x)/n)^2$ if $\text{wt}(x) \leq a$ and $R_x^0 = 0$ otherwise (here we use $\text{wt}(x) = \Delta(x, \mathbf{0})$ to denote the Hamming weight of x). We now have

$$\begin{aligned}
W_0 &= 2^{-n} \sum_{x \in \{0,1\}^n} \exp_2 \left(\frac{n}{A} R_x^0 \right) \\
&\leq 1 + 2^{-n} \sum_{i=0}^{a} \binom{n}{i} \exp_2 \left(\frac{n}{A} \left(1 - \frac{2i}{n}\right)^2 \right) \\
&\leq 1 + n 2^{-n} \exp_2 \left(\max_{0 \leq i \leq a} \left\{ H\left(\frac{i}{n}\right) n + \frac{4n}{A} \left(\frac{1}{2} - \frac{i}{n}\right)^2 \right\} \right) \\
&\leq 1 + n 2^{un}
\end{aligned} \tag{8.40}$$

where $u \stackrel{\text{def}}{=} \max_{0 \le y \le (1/2-\varepsilon)} \{H(y) - 1 + \frac{4}{A}(\frac{1}{2} - y)^2\}$. We now claim that for every y, $0 \le y \le 1/2$, we have $H(y) \le 1 - \frac{2}{\ln 2}(\frac{1}{2} - y)^2$. One way to prove this is to consider the Taylor expansion around $1/2$ of $H(y)$, which is valid for the range $0 \le y \le 1/2$. We have $H'(1/2) = 0$ and $H''(1/2) = -4/\ln 2$. Also it is easy to check that all odd derivatives of $H(y)$ at $y = 1/2$ are non-negative while the even derivatives are non-positive. Thus $H(y) \le H(1/2) - H''(1/2)\frac{(1/2-y)^2}{2} = 1 - \frac{2}{\ln 2}(\frac{1}{2} - y)^2$. Therefore

$$u \le \max_{0 \le y \le (1/2-\varepsilon)} \Big(\frac{4}{A} - \frac{2}{\ln 2}\Big)\Big(\frac{1}{2} - y\Big)^2 = -4\Big(\frac{1}{\ln 4} - \frac{1}{A}\Big)\varepsilon^2 \ , \tag{8.41}$$

since $A > \ln 4$. Combining (8.39), (8.40) and (8.41), it is now easy to argue that we will have $W_C = W_k = O(1)$ as long as $k < -un$, which will be satisfied if $k < 4(\frac{1}{\ln 4} - \frac{1}{A})\varepsilon^2 n$. Thus the statement of the lemma holds, for example, with $A = 2$ and $\sigma = 0.85$. $\Box$ *(Lemma 8.12)*

Remark: Arguing exactly as in the remark following the proof of Theorem 5.5, one can also add the condition $\delta(\mathcal{C}) \ge (1/2-\varepsilon)$ to the claim of Lemma 8.12. The proof will then pick b_{i+1} randomly from among all choices such that $\mathrm{span}(b_1, b_2, \ldots, b_{i+1}) \cap B(\mathbf{0}, (\frac{1}{2} - \varepsilon)n) = \emptyset$.

A greedy construction of the "inner" code

We now discuss how a code guaranteed by Lemma 8.12 can be constructed in a greedy fashion. We will refer to some notation that was used in the proof of Lemma 8.12. The algorithm works as follows:

Algorithm GREEDY-INNER:

Parameters: Dimension k; $\varepsilon, A > 0$ (where A is the absolute constant from Lemma 8.12)

Output: A binary linear code $C = \text{GREEDY}(k, \varepsilon)$ with dimension k, blocklength $n = O(k/\varepsilon^2)$ and minimum distance at least $(1/2 - \varepsilon)n$ such that for every $x \in \{0,1\}^n$, Condition (8.35) holds.

1. Start with $b_0 = \mathbf{0}$.
2. For $i = 1, 2, \ldots, k$:
   - Let $U_i = \{x \in \{0,1\}^n : \mathrm{span}(b_1, b_2, \ldots, b_{i-1}, x) \cap B(\mathbf{0}, (1/2 - \varepsilon)n) = \emptyset\ \}$.
   - Pick $b_i \in U_i$ that *minimizes* the potential function $W_i = 2^{-n}\sum_x 2^{\frac{n}{A}\cdot R_x^i}$, where R_x^i is as defined in Equation (8.37) (break ties arbitrarily)
3. Output $C = \mathrm{span}(b_1, b_2, \ldots, b_k)$.

The following result easily follows from the proof of Lemma 8.12 since each of the k iterations of the for loop above can be implemented to run in $2^{O(n)}$ time.

Lemma 8.13 *Algorithm* GREEDY-INNER *constructs a code* GREEDY(k, ε) *with the desired properties in* $k \cdot 2^{O(n)}$ *time.*

8.6.2 The concatenated code and the decoding algorithm

The statement of Theorem 8.11 that we set out to prove, follows immediately from the concatenated code construction guaranteed by the following theorem.

Theorem 8.14 *There exist absolute constants $b, d > 0$ such that for every integer K and every $\varepsilon > 0$, there exists a concatenated code $C_K \stackrel{\text{def}}{=} \mathrm{RS}\oplus\ \textsc{Greedy}(m, \varepsilon/2)$ (for a suitable parameter m) that has the following properties:*

1. *C_K is a linear code of dimension K, blocklength $N \le \frac{bK}{\varepsilon^4}$, and minimum distance at least $(\frac{1}{2} - \varepsilon)N$.*
2. *The generator matrix of C_K can be constructed in $N^{O(\varepsilon^{-2})}$ time.*
3. *C_K is $((\frac{1}{2} - \varepsilon)N, d/\varepsilon^2)$-list decodable; i.e. any Hamming ball of radius $(1/2 - \varepsilon)N$ has at most $O(\varepsilon^{-2})$ codewords of C_K.*
4. *There exists a polynomial time list decoding algorithm for C_K that can correct up to $(1/2 - \varepsilon)N$ errors.*

Proof: The code C_K is constructed by concatenating an outer Reed-Solomon code C_{RS} over $\mathrm{GF}(2^m)$ of blocklength $n_0 = 2^m$ and dimension $k_0 = K/m$ (for some integer m which will be specified later in the proof) with an inner code $C_{\text{inner}} = \textsc{Greedy}(m, \varepsilon/2)$ (as guaranteed by Lemma 8.13). Since the blocklength of C_{inner} is $n_1 = O(\frac{m}{\varepsilon^2})$, the concatenated code C_K has dimension K and blocklength

$$N = n_0 n_1 = O\Big(\frac{n_0 m}{\varepsilon^2}\Big) . \tag{8.42}$$

and minimum distance D at least

$$D \ge N\Big(1 - \frac{K}{m n_0}\Big)\Big(\frac{1}{2} - \frac{\varepsilon}{2}\Big) . \tag{8.43}$$

For ease of notation, we often hide constants using the big-Oh notation in what follows, but in all these cases the hidden constants will be absolute constants that do not depend upon ε. By Lemma 8.13, C_{inner} is constructible in $2^{O(n_1)} = 2^{O(m/\varepsilon^2)}$ time, and since $m = \lg n_0$, the generator matrix for C_K can be constructed in $N^{O(\varepsilon^{-2})}$ time. This proves Property 2 claimed in the theorem.

We will now present a polynomial time list decoding algorithm for C_K to correct a fraction $(1/2 - \varepsilon)$ of errors using lists of size $O(1/\varepsilon^2)$. This will clearly establish both Properties 3 and 4 claimed in the theorem.

The decoding algorithm will follow the same approach as that of Theorems 8.2 and 8.7. Let $y \in \{0,1\}^N$ be any received word. We wish to find a list of all codewords $\mathbf{c} \in C_K$ such that $\Delta(y, \mathbf{c}) \le (1/2 - \varepsilon)N$. For $1 \le i \le n_0$, denote by y_i the portion of y in block i of the codeword (i.e. the portion corresponding to the encoding by C_{inner} of the i^{th} Reed-Solomon symbol).

Now, consider the following decoding algorithm for C_K. First, the inner codes are decoded by a brute force procedure that goes over all codewords. Specifically, for each position i, $1 \le i \le n_0$, of the outer Reed-Solomon code, and for each $\alpha \in \mathrm{GF}(2^m)$, the inner decoder computes a set of weights $w_{i,\alpha}$ defined by:

$$w_{i,\alpha} = \max\Big\{\Big(\frac{1}{2} - \frac{\varepsilon}{2} - \Delta(y_i, C_{\text{inner}}(\alpha))\Big), 0\Big\} \tag{8.44}$$

Once again all the n_0 inner decodings can be performed in $O(n_0 \cdot 2^m \cdot m/\varepsilon^2) = O(n_0^2 m/\varepsilon^2)$ time, and thus certainly in $O(N^2)$ time.

These weights are then passed to the soft decoding algorithm for Reed-Solomon codes from Theorem 6.21. To analyze the performance of the soft decoding algorithm, we will make use of the crucial combinatorial property of C_{inner} which is guaranteed by Lemmas 8.12 and 8.13. Using this property of C_{inner}, we have, for each i, $1 \le i \le n_0$,

$$\sum_{\alpha \in \mathrm{GF}(2^m)} w_{i,\alpha}^2 \le B' \ , \tag{8.45}$$

for some *absolute* constant B'.

Using the soft decoding algorithm to complete the decoding implies that one can find, in time polynomial in n_0 and $1/\gamma$, a list of *all* codewords $\mathbf{c} \in C_K$ that satisfy

$$\sum_{i=1}^{n_0} w_{i,\mathbf{c}_i} \ge \sqrt{\Big(n_0 - \frac{n_0 - K/m + 1}{1+\gamma}\Big) \cdot \sum_{i,\alpha} w_{i,\alpha}^2} \ . \tag{8.46}$$

In the above, $\gamma > 0$ is a parameter to be set later, and we have abused notation to denote $w_{i,\mathbf{c}_i} = w_{i,\alpha_i}$ where $\alpha_i \in \mathrm{GF}(2^m)$ is such that $C_{\text{inner}}(\alpha_i) = \mathbf{c}_i$.

The soft decoding algorithm, used as stated in Theorem 6.21, can decode even with the choice $\gamma = 0$ in the above Condition (8.46). However, with a positive value of γ, we can appeal to the weighted Johnson bounds from Chapter 3, specifically the result stated in Part (ii) of Corollary 3.7, to conclude that there will be at most $(1 + 1/\gamma)$ codewords $\mathbf{c}$ that satisfy Condition (8.46) for *any* choice of weights $w_{i,\alpha}$. Hence, our decoding algorithm, too, will output only a list of at most $O(1/\gamma)$ codewords.

We now analyze the number of errors corrected by the algorithm. Using (8.44) and (8.45), we notice that Condition (8.46) will be satisfied if

$$\begin{aligned}
\sum_{i=1}^{n_0}\Big(\frac{1}{2} - \frac{\varepsilon}{2} - \frac{\Delta(y_i, \mathbf{c}_i)}{n_1}\Big) &\ge \sqrt{\Big(\gamma n_0 + \frac{K}{m}\Big) \cdot n_0 B'} \\
\Longleftarrow \quad \Delta(y, \mathbf{c}) &\le N\Big(\frac{1}{2} - \frac{\varepsilon}{2} - \sqrt{B'\Big(\gamma + \frac{K}{m n_0}\Big)}\Big) \\
\Longleftarrow \quad \Delta(y, \mathbf{c}) &\le \Big(\frac{1}{2} - \varepsilon\Big) N \ ,
\end{aligned}$$

where the last step holds as long as we pick $\gamma \le \frac{\varepsilon^2}{8B'}$ and m such that

$$\frac{K}{m n_0} = \frac{K}{m 2^m} \le \frac{\varepsilon^2}{8B'} \ . \tag{8.47}$$

Thus we have a decoding algorithm that outputs a list of all $O(1/\gamma) = O(\varepsilon^{-2})$ codewords that differ from y in at most $(1/2 - \varepsilon)N$ positions. This establishes Properties 3 and 4 claimed in the theorem.

Also, by (8.47), we have $m n_0 = O(K/\varepsilon^2)$. Plugging this into (8.42) and (8.43), we have that the blocklength N of C_K satisfies $N = O(K/\varepsilon^4)$ and the distance D satisfies $D \ge (1/2 - \varepsilon)N$. This establishes Property 1 as well, and completes the proof of the theorem. □

Discussion: The time required to construct a code with the properties claimed in Theorem 8.14, though polynomial for every fixed ε, grows as $N^{O(\varepsilon^{-2})}$. It is desirable to obtain a construction time of the form $O(f(\varepsilon)n^c)$ where c is a fixed constant independent of ε, for some arbitrary function f. A family whose codes can be constructed within such time bounds is often referred to as being *uniformly constructive* (see [ABN+92] for a formal definition).

If one uses the best known algebraic-geometric codes (namely those discussed in Section 6.3.9) as the outer code instead of Reed-Solomon codes, one can carry out the code construction of Theorem 8.14 in $2^{O(\varepsilon^{-2}\log(1/\varepsilon))}N^c$ time for a fixed constant c (the constant c will depend upon the time required to construct the outer algebraic-geometric code). This is not entirely satisfying since the construction complexity of the necessary algebraic-geometric codes is still quite high. A further drawback is that the promise of a polynomial time decoding algorithm will hinge on assumptions about specific representations of the AG-code.

The construction of Theorem 8.8 had a similar drawback in terms of high deterministic construction time. Nevertheless, it had a highly efficient probabilistic construction that had the claimed properties with high probability. A similar probabilistic construction for the codes of Theorem 8.11 is not known. The reason for this is that the existence result of Lemma 8.12 is not known to hold with high probability for a *random* code (unlike the situation in Theorem 8.8 where it is known that a the rate vs. distance trade-off of a random linear code meets the Gilbert-Varshamov bound with high probability). Thus the following is an interesting open question:

Open Question 8.1 *a) Is there a randomized (Monte Carlo) construction of a family of binary linear codes of rate $\Omega(\varepsilon^4)$ list decodable up to a fraction $(1/2-\varepsilon)$ of errors, that runs in, say, quadratic time in the blocklength?*

b) Is there a uniformly constructive family of binary linear codes which can be list decoded efficiently from a fraction $(1/2-\varepsilon)$ errors and which have rate $\Omega(\varepsilon^4)$ or better?

8.7 Open Questions

In addition to the above, there are two central open questions regarding the contents of this chapter. These are listed below.

Open Question 8.2 *Let C be a q-ary concatenated code of designed distance $\Delta\cdot\delta$ with the outer code being a Reed-Solomon code of relative distance Δ, and the inner code being an arbitrary q-ary code of relative distance δ. Is there a polynomial time list decoding algorithm for C to decode up to its Johnson radius? In other words, is there a polynomial time algorithm to list decode up to a fraction $(1-1/q)(1-\sqrt{1-\frac{\Delta\cdot\delta}{(1-1/q)}})$ of errors?*

In fact the following "easier" question is also open. As mentioned earlier, the GMD algorithm can be used to unique decode such codes up to the product bound (i.e. a fraction $\Delta\delta/2$ of errors) in polynomial time [For66a, Jus72]. The question below simply asks if one can always, for every concatenated code with an outer Reed-Solomon code, perform efficient list decoding beyond the product bound.

Open Question 8.3 *Let C be a q-ary concatenated code of designed distance $\Delta \cdot \delta$ with the outer code being a Reed-Solomon code of relative distance Δ, and the inner code being an arbitrary q-ary code of relative distance δ. Is there a polynomial time list decoding algorithm for C to decode up to a fraction $f(\Delta, \delta)$ of errors, where f is a real-valued function that takes values in $[0, 1-1/q)$ and which satisfies $f(\Delta, \delta) > \frac{\Delta\delta}{2}$ in the entire range $0 < \Delta < 1$ and $0 < \delta < 1 - 1/q$? In other words, is there a polynomial time algorithm to always list decode such concatenated codes beyond the product bound ?*

Finally, we state the open question concerning the best rate of a constructive family of binary codes with very high list decodability.

Open Question 8.4 *Is there a polynomial time constructible family of binary codes which have rate $\Omega(\varepsilon^a)$ for some $a < 4$ and which have a polynomial time list decoding algorithm to decode up to a fraction $(1/2 - \varepsilon)$ of errors ?*

We know that existentially $a = 2$ is achievable and that this is the best possible.

We note that even if Question 8.2 is answered in the affirmative, the rate achievable for a list decoding radius of $(1 - 1/q - \varepsilon)$ is only $O(\varepsilon^6 \log(1/\varepsilon))$. This is because we need to have $\Delta = 1 - O(\varepsilon^2)$ and $\delta = (1 - 1/q - O(\varepsilon^2))$ in order for the Johnson radius to be $(1 - 1/q - \varepsilon)$. The former implies that the rate of the Reed-Solomon code is $O(\varepsilon^2)$ and the latter, by appealing to the linear programming bounds [MRRW77], implies that the rate of the inner code is $O(\varepsilon^4 \log(1/\varepsilon))$. The overall rate is thus at most $O(\varepsilon^6 \log(1/\varepsilon))$. An answer in the affirmative to Question 8.4, therefore, has to either not be based on concatenation at all, or must use a special purpose construction, akin to the one in Section 8.6, which can be list decoded beyond its Johnson radius. In the next chapter, we will present a probabilistic construction with $\Omega(\varepsilon^3)$ rate, but the decoding time will be sub-exponential as opposed to polynomial.

8.8 Bibliographic Notes

Concatenated codes were defined and studied extensively in the seminal Ph.D. work of Forney [For66a], and by now have deservedly become standard textbook material. Forney [For66b] developed a Generalized Minimum Distance (GMD) decoding algorithm for Reed-Solomon codes, and used it as a soft decoding algorithm to decode concatenated schemes with outer Reed-Solomon code. He presented a detailed estimation of the probability of decoding error for such a scheme. Justesen [Jus72] used a concatenated scheme to give the first explicit construction of an asymptotically good binary code family, thereby refuting the popular myth existing at that time that explicitly specified codes would probably never be asymptotically good. Justesen also gave an algorithm using GMD decoding to decode his concatenated codes up to the product bound (i.e. half the designed distance). In fact, his result implicitly shows that any concatenated code whose outer code has an efficient error-and-erasure decoding algorithm (which in turn implies a GMD algorithm by results of Forney [For66b]) can be uniquely decoded up to the product bound. The GMD based algorithm for unique decoding concatenated codes up to the product bound is also described in detail in Appendix A of this thesis.

The inner decoding stage in all these algorithms passed to the outer Reed-Solomon decoder at most one field element together with an associated weight (confidence information) for each outer codeword position. This was also the case in a recent work of Nielsen [Nie00] who investigated in detail decoding algorithms for concatenated codes where the inner code is decoded uniquely but instead of the GMD algorithm, the weighted list decoding algorithm (from Chapter 6) is used for decoding the outer Reed-Solomon code. In contrast, in the algorithms discussed in this chapter, the inner decoders pass to the outer Reed-Solomon decoder not one, but several field elements, each with an associated weight, as candidate symbols for each position. We should mention that Nielsen [Nie00] also considers a decoding algorithm where the inner codes are list decoded beyond half the minimum distance, but does not present a quantitative analysis of such an algorithm. Indeed to perform such an analysis one needs at least a partial knowledge of the weight distribution of cosets of the inner code, which is a highly non-trivial task in itself. The result of Proposition 8.5 from this chapter provides a non-trivial, and apparently new, bound on the weight distribution of cosets given the knowledge of *only* the minimum distance of the code. We believe, though, that to really reap the benefits of the soft Reed-Solomon decoder in concatenated code constructions, one must use special purpose inner codes for which we have good bounds on the weight distributions of cosets. In fact, our results in Section 8.6 follow this approach, but we believe there is still lots of improvements to be made.

The decoding algorithms from Section 8.4 when the inner code is the Hadamard code appear in [GS00]. The results of Section 8.5 have not yet been published in a conference/journal, and are stated for the first time in this thesis. The code construction and decoding algorithm of Section 8.6 appear in [GHSZ00].

Chapter 9

New, Expander-based List Decodable Codes

9.1 Introduction

In the previous chapters, we have already seen constructions of asymptotically good codes of good rate over both large alphabets (the AG-codes from Chapter 6) and the binary alphabet (the concatenated codes from Chapter 8), that are efficiently list decodable up to a "maximum" possible radius. By "maximum" possible radius we mean list decoding up to a fraction $(1 - 1/q - \varepsilon)$ of errors for q-ary codes. This translates into a fraction $(1 - \varepsilon)$ of errors for codes over large enough alphabets, and a fraction $(1/2 - \varepsilon)$ of errors for binary codes. For codes with such large list decodability, which we called "highly list decodable codes", the goal is to find efficient constructions that achieve good rate (typically of the form $\Omega(\varepsilon^a)$ for some reasonably small a), together with efficient decoding algorithms.

The earlier results achieve fairly non-trivial trade-offs in this regard. The list decoding algorithm for AG-codes from Chapter 6 implies highly list decodable codes over an alphabet of size $O(1/\varepsilon^4)$ that have rate $\Omega(\varepsilon^2)$. The results of the previous chapter on concatenated codes give constructions of highly list decodable binary codes of rate $\Omega(\varepsilon^4)$.

One shortcoming of the former result is that the necessary AG-codes are very complicated to construct and the known decoding algorithms need a non-standard representation of the code for the claim of polynomial runtime to hold. Families of Reed-Solomon codes also offer similar list decodability with a rate of $\Omega(\varepsilon^2)$, but their alphabet size is at least as large as the blocklength and hence they do not achieve a alphabet size that is a constant dependent only on ε. In fact, other than AG-codes, there were no other known families of codes that are list decodable to a fraction $(1 - \varepsilon)$ of errors, have reasonably large rate, and are defined over a constant-sized alphabet.

The other shortcomings of the above mentioned results are that there is potential for improvement in the rate. The existential results (Chapter 5) show that a rate $\Omega(\varepsilon)$ is possible for highly list decodable codes over large alphabets, and a rate $\Omega(\varepsilon^2)$ is possible for binary codes. Thus the constructive results are not optimal with respect to the rate (though they are not off by very much).

In this chapter, we present novel constructions of list decodable codes that address the above shortcomings. Our codes are simple to construct and decode, and share the common thread of using expander (or expander-like) graphs as a component. The expanders are used as a "dispersing" component which serve the purpose of redistributing symbols in order to ensure a near-uniform distribution of errors between the various blocks of the codeword. They thereby enable the design of efficient decoding algorithms that correct a large number of errors through various forms of "voting" procedures.

The basic idea behind the expander-based constructions of this chapter will also find use later in Chapter 11, where we will present codes of good rate that are uniquely decodable up to a "maximum" possible radius in *linear* time. This indicates the quite general applicability and power of the techniques used in this chapter.

An important combinatorial tool used in our constructions are "pseudolinear codes". We view the construction and use of pseudolinear codes as being of independent interest, and hope that it will find several applications in the future.[1] Pseudolinear codes possess the useful properties of efficient encoding and succinct representation which all linear codes automatically have, but they have the additional nice property that random pseudolinear codes (with suitable parameters) inherit the same list-of-L decoding properties as completely general random codes.

We next present a detailed statement of the results of this chapter, followed by an overview of the main techniques used.

9.2 Overview of Results and Techniques

9.2.1 Main results

Our constructions of highly list decodable codes give the following:

(1) Codes of rate $\Omega(\varepsilon^2)$ over an alphabet of size $2^{O(\varepsilon^{-1}\log(1/\varepsilon))}$, list decodable up to a fraction $(1-\varepsilon)$ of errors in near-quadratic time.

(2a) Codes of rate $\Omega(\varepsilon)$ over an alphabet of size $2^{O(\varepsilon^{-1}\log(1/\varepsilon))}$, list decodable up to a fraction $(1-\varepsilon)$ of errors in sub-exponential time.

(2b) Binary codes of rate $\Omega(\varepsilon^3)$ list decodable up to a fraction $(1/2-\varepsilon)$ of errors in sub-exponential time.

(3) Codes of rate $\Omega(t^{-3}\varepsilon^{2+2/t})$ over an alphabet of size $O(1/\varepsilon^b)$, list decodable up to a fraction $(1-\varepsilon)$ of errors. Here $t \geq 1$ is an arbitrary integer and $b > t$ an arbitrary real.

The first three constructions $(1, 2a, 2b)$ use the expander-based approach mentioned in the introduction. The last construction does not use expanders and is based on multiple concatenated codes combined together by juxtaposing symbols together — we call such codes *juxtaposed codes* for purposes of easy reference.[2] We discuss these codes also in this chapter since their construction has much the same motivation as that of (1). Moreover, they also use some of same machinery that construction (1) uses; specifically they too use pseudolinear codes as inner codes in a concatenated scheme. The main advantage of the juxtaposed code construction is that they can achieve better alphabet size than the construction (1), at the expense of a slight worsening of the rate.

[1] Pseudolinear codes are also used in the next chapter on list decoding from erasures.

[2] Juxtaposed codes will be used again in the next chapter to obtain constructions of good codes with very high list decodability from erasures. Codes similar to our juxtaposition based constructions are also called multilevel concatenated codes in the literature [Dum98], but we believe the term juxtaposed codes is more natural and we use this terminology.

The detailed specification of all parameters of our expander-based constructions are listed in Figure 9-1. We next present a discussion of the individual results and compare them with previously known constructions.

No	Alphabet	Decoding radius	Rate	Encoding time	Decoding time	Const. time (probabilistic)*
1	$2^{\varepsilon^{-1}\log(1/\varepsilon)}$	$1-\varepsilon$	ε^2	$n \log n$	$n^2 \log n$	$\log^2 n/\varepsilon$
2a	$2^{\varepsilon^{-1}\log(1/\varepsilon)}$	$1-\varepsilon$	ε	$n^{2(1-\gamma)}\log^2 n$	$2^{n^\gamma \log(1/\varepsilon)}$	$n^{2(1-\gamma)}/\varepsilon$
2b	2	$1/2-\varepsilon$	ε^3	$n^{2(1-\gamma)}\log^2 n$	$2^{n^\gamma \log(1/\varepsilon)}$	$n^{2(1-\gamma)}/\varepsilon$
3	$2^{\log^2(1/\varepsilon)}$	$1-\varepsilon$	$\varepsilon^2\log^{-3}(1/\varepsilon)$	$n\log^2 n$	$n^{1/\varepsilon}$	$\log^2 n/\varepsilon^2$

Figure 9-1: The parameters of our codes. n stands for the length of the code. For readability, the $O(\cdot)$ and $\Omega(\cdot)$ notation, and certain $\log^{O(1)}(1/\varepsilon)$ factors have been omitted. The value of γ is in the interval $(0, 1]$; its value influences the rate by a constant factor. The decoding radius shows the fraction of errors which the decoding algorithms can correct. .
$*$A detailed discussion on the construction times is presented later in this Section.

Our first code (1) enables efficient list decodability from up to a fraction $(1-\varepsilon)$ of errors, for an arbitrary constant $\varepsilon > 0$. Its distinguishing feature is the near-quadratic decoding time and fairly high (namely $\Omega(\varepsilon^2)$) rate, while maintaining a constant alphabet size. The only other known constructible codes with comparable parameters are certain families of algebraic-geometric codes [TVZ82, GS95b]. As discussed in Chapter 6 (specifically in Theorem 6.38), such AG-codes can achieve $\Omega(\varepsilon^2)$ rate and $O(1/\varepsilon^4)$ alphabet size. While they yield a much better alphabet size, AG-codes suffer from the drawback of complicated construction and decoding algorithms. It is only known how to list decode them in polynomial time using certain auxiliary advice (of polynomial size), and it not known how to compute this information in sub-exponential (randomized or deterministic) time (the reader might recall the discussion about this in Chapter 6, Section 6.3.9). Even regarding construction complexity, only very recently [SAK$^+$01] showed how to construct the generator matrix of the necessary AG-codes in near-cubic time. In comparison, our construction time, although probabilistic, is essentially negligible.

The second code (2a) also enables list decodability up to a fraction $(1-\varepsilon)$ of errors. Its distinguishing feature is the *optimal* $\Omega(\varepsilon)$ rate. The only previously known codes with such rate were purely random codes (even Reed-Solomon codes that have super-constant alphabet size only guarantee $\Omega(\varepsilon^2)$ rate). However, the best known decoding time for random codes is $2^{O(n)}$, and it is likely that no significantly better algorithm exists. Our codes also have significant random components; however, they can be decoded substantially faster in sub-exponential time. The binary version (2b) of the aforementioned codes, which correct up to a fraction $(1/2-\varepsilon)$ of errors, also beat the $\Omega(\varepsilon^4)$ rate of best constructive codes from the previous chapter (specifically, the result of Theorem 8.11). They are only off by a factor of $O(\varepsilon)$ from the optimal $\Omega(\varepsilon^2)$ rate implied by the existential results of Chapter 5.

For the codes (3), the rate is not as good as the construction (1), but one can get substantial improvements in alphabet size for a relatively small worsening of the rate. For example, as listed in Figure 9-1, it can achieve a rate of $\Omega(\varepsilon^2\log^{-3}(1/\varepsilon))$ for an alphabet size of $2^{O(\log^2(1/\varepsilon))}$. By worsening the rate further, it is even possible to achieve an alphabet size better than $O(1/\varepsilon^4)$, which is the alphabet size of the best known AG-codes that are list decodable up to a fraction $(1-\varepsilon)$ of errors (see Theorem 9.19).

Construction times. All of our constructions use the probabilistic method to obtain certain "gadgets" which are then used together with explicitly specified objects. The probabilistic method generates such building blocks with high probability. Therefore, our probabilistic construction algorithms are *randomized Monte Carlo*, and the claimed list decodability property holds with high probability over the choice of the random components. We note, however, that our probabilistic algorithms using R random bits can be derandomized and converted into deterministic algorithms using $O(R)$ space and running in $2^{O(R)}$ time in a straightforward manner. (The resulting code will be *guaranteed* to have the claimed list decodability property, i.e., the derandomization includes "verification" as well.) For the codes (1), a naive derandomization would only give a quasi-polynomial time construction. Nevertheless, by using the method of conditional expectations for derandomization, we will show the code can be constructed deterministically in time $n^{O(\varepsilon^{-1}\log(1/\varepsilon))}$. Similarly, for our constructions (2a,2b), a conditional expectations based derandomization enables a deterministic construction in (roughly) $2^{O(n^{1-\gamma})}$ time. Note that the both the probabilistic and deterministic construction times of (2a,2b) get worse as the decoding time gets better and better.

We stress that modulo the gadget construction, generating each symbol of a codeword can be done in *polylogarithmic time.*

9.2.2 Our techniques

Expander-based constructions

At a high level, the codes (1), (2a,2b) are all constructed using a similar scheme. The basic components of the constructions are: a "left" code (say, C) and a "dispersing" graph G, and in the case of binary codes, a "right" binary code C'. The left code C is typically a concatenation of efficient list decodable codes, namely Reed-Solomon codes and certain good list decodable "pseudolinear" codes whose existence we prove in Section 9.3. Such pseudolinear codes can either be found by brute-force or, one can pick a code at random and thus get a much faster probabilistic construction that works with high probability. The graph G is a suitable expander with good vertex-expansion, i.e., a bipartite graph with the property that the neighborhood of every reasonable sized subset of the left side (say, consisting of a fraction 1/2 of the left nodes) misses at most a fraction ε of the nodes on the right.

Given the above components, the codes are constructed as follows. For each codeword x of C, we construct a new codeword y by distributing the symbols of x from left to right according to the edges in G. The juxtaposition of symbols "sent" to each right node of G forms a symbol of the codeword y of the final code C_1. The code C_1 will thus be defined over a large alphabet. See Figure 9-2 for a sketch of the basic construction scheme. For construction (2b), in order to get a binary code, we add a final level of concatenation with an appropriate binary code C'. This is similar to the construction due to Alon *et al* in [ABN+92]. Our contribution is in the design of efficient decoding algorithms to correct a large fraction of errors for such code constructions.

The role of the dispersing graph G is, roughly speaking, to convert an arbitrary distribution of errors that could exist between the various blocks of the (concatenated) left

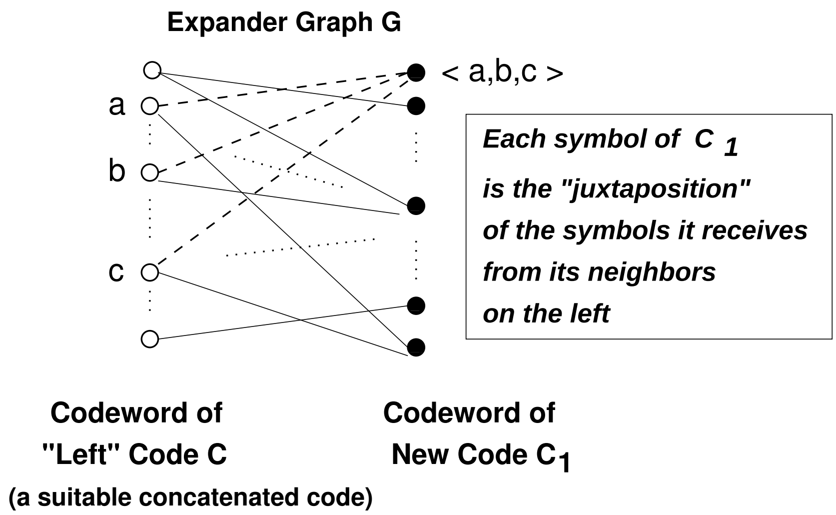

Figure 9-2: Basic structure of our code constructions. To get binary code constructions, each symbol of C_1 is further concatenated with a good, constant-sized binary code.

code C into a near-uniform distribution. This permits recovery of a (somewhat corrupted) received word x for C from a heavily corrupted received word y for the code C_1, using a certain "voting" scheme. The voting scheme we use is very simple: each position of y votes for all positions of x which are connected to it by an edge of G. This allows us to collect a list of potential symbols for each position of x. These lists are then used by a suitable decoding algorithm for C to finish the decoding.

The specifics of the implementation of the above ideas depend on the actual code construction. For the code (1), we take the left code C to be a concatenation of a Reed-Solomon code and a suitable pseudolinear code. Such a code can be list decoded in near-quadratic time using the Reed-Solomon decoding algorithms discussed in Chapter 6. The codes (2a,2b) are constructed by picking C to be a concatenation of a constant number of levels of "pseudo-linear" codes with an outermost Reed-Solomon code (we call such codes *multi-concatenated codes*). The pseudolinear codes can perform list decoding when given as input a vector of lists, one per codeword position, such that at least half of the lists contain the correct symbol. The important fact is that such pseudolinear codes exist with a fixed constant rate that is *independent* of the length of the lists that are involved. This allows the decoding algorithm to propagate the candidate symbols through the concatenation levels while decreasing the rate only by a small factor at each level. The parameters are so picked that the decoding of each of these pseudolinear codes as well as the overall code can be done in sub-exponential time.

Juxtaposed code constructions

The second approach behind our code constructions, which is used in Section 9.6, is aimed at obtaining similar (or slightly worse) rates using smaller alphabet size, and is the basis of the constructions described in Section 9.6. In this approach, multiple Reed-Solomon codes (of varying rates) are concatenated with *several* different inner codes (of varying rate and list decodability). Corresponding to each Reed-Solomon and inner code pair, we get one concatenated codeword, and the final encoding of a message is obtained by "juxtaposing together" the symbols from the various individual concatenated codewords.

The purpose of using multiple concatenated codes is that depending on the distribution of errors in the received word, the portions of it corresponding to a significant fraction of a *certain* inner encoding (that depends on the level of non-uniformity in the distribution of errors) will have relatively few errors. These can then be decoded to provide useful information about a large fraction of symbols to the decoder of the *corresponding* outer Reed-Solomon code. Essentially, depending on how (non)-uniformly the errors are distributed, a certain concatenated code "kicks in" and enables recovery of the message. The use of multiple concatenated codes reduces the rate compared to the expander based constructions, but we gain in the alphabet size. For example, for a near-quadratic (namely, $\Omega(\varepsilon^2 \log^{-O(1)}(1/\varepsilon))$) rate, the alphabet size can be quasi-polynomial as opposed to exponential in $1/\varepsilon$.

9.2.3 A useful definition

For our results, the following (more general) notion of good list decodability proves extremely useful — for purposes of disambiguation from (e, ℓ)-list decodability, we call this notion "list recoverability".

Definition 9.1 *For α, $0 < \alpha < 1$, and integers $L \geq \ell \geq 2$, a q-ary code C of blocklength n is said to be (α, ℓ, L)-*list recoverable *if given arbitrary "lists" $L_i \subseteq \mathbb{F}_q$ of size at most ℓ for each i, $1 \leq i \leq n$, the number of codewords $\mathbf{c} = \langle c_1, \ldots, c_n \rangle \in C$ such that $c_i \in L_i$ for at least αn values of i, is at most L.*

We will loosely refer to the task of decoding a code under the above model as "list recovering" the code.

Remark: A code of blocklength n is $(\alpha, 1, L)$-list recoverable if and only if it is $((1-\alpha)n, L)$-list decodable.

9.3 Pseudolinear Codes: Existence results and Properties

In this section, we prove existence results using the probabilistic method for codes which serve as inner codes in our concatenated code constructions. The inner codes will be "pseudolinear codes" with appropriate parameters. We now formally define the notion of "pseudolinear" code families and prove some of the basic list decodability properties offered by random pseudolinear codes. An informal description of pseudolinear codes was given in Chapter 2, where we had put off a more detailed treatment to later when the machinery is really used (which is in this chapter).

The notion of pseudolinear codes appears to be new, and it plays a critical role in translating list decodability results for general, non-linear codes into similar results for codes, which albeit not linear, still have a succinct description, and allow for efficient encoding. In our applications, these pseudolinear codes, which are typically used as inner codes in suitable concatenated schemes, are critical in getting efficient constructions for our codes.

9.3.1 Pseudolinear (Code) Families

Informally, an L-wise independent code family is a sample space of codes such that the encodings of any L non-zero messages are completely independent for a random code drawn from the family. The formal definition follows.

Definition 9.2 *An L-wise independent $(n,k)_q$-code family $\mathcal{F}$ is a sample space of codes that map k symbols over $\mathbb{F}_q$ to n symbols over $\mathbb{F}_q$ such that for every set of L non-zero messages $\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_L} \in \mathbb{F}_q^k$, the random variables $C(\mathbf{x_1}), C(\mathbf{x_2}), \ldots, C(\mathbf{x_L})$ are completely independent, for a code C picked uniformly at random from the family $\mathcal{F}$.*

A random code picked from an L-wise independent family often tends to have very good list decoding properties for decoding with list size L, owing to the mutual independence of any set of L (non-zero) codewords. An example of an L-wise independent code family is the space of all general, non-linear q-ary codes of blocklength n and dimension k, which is clearly L-wise independent, for all L, $1 \le L < q^k$. While a random, non-linear code has excellent randomness properties, it comes from a very large sample space and there is no succinct representation of a general code from the family.[3] We now define a family of codes which we call *pseudolinear* that has the desired L-wise independence property and in addition is succinct. Thus a random code drawn this family has the desired randomness properties, can be succinctly represented, and has an efficient encoding procedure.

Definition 9.3 (Pseudolinear Codes) *For a prime power q, integer $L \ge 1$, and positive integers k, n with $k \le n$, an (n,k,L,q)-pseudolinear family $\mathcal{F}(n,k,L,q)$ of codes is defined as follows. Let H be the parity check matrix of any q-ary linear code of blocklength $(q^k - 1)$, minimum distance at least $(L+1)$ and dimension $q^k - 1 - O(kL)$ (for example, one can use parity check matrices of q-ary BCH codes of designed distance $(L+1)$, cf. [AS92, Chap. 15]). A random code C_A in the pseudolinear family $\mathcal{F}(n,k,L,q)$ is specified by a random $n \times O(kL)$ matrix A over $\mathbb{F}_q$. Under the code C_A, a message $x \in \mathbb{F}_q^k \setminus \{0\}$ is mapped to $A \cdot H_x \in \mathbb{F}_q^n$ where $H_x \in \mathbb{F}_q^{O(kL)}$ is the column of H indexed by x (viewed as an integer in the range $[1, q^k)$). (We also define $H_0 = \mathbf{0}$ to be the all-zeroes vector.)*

Given $1 \le x < q^k$, a description of the column H_x can be obtained in time polynomial in k and $\log q$, since there are explicit descriptions of the parity check matrices of BCH codes of distance at least $(L+1)$ and blocklength (q^k-1), in terms of the powers of the generating element of $\mathrm{GF}(q^k)$ over $\mathrm{GF}(q)$ (see, for example, [MS81, Chap. 9]). Hence encoding as per these codes is an efficient operation. In addition to these complexity issues, the crucial combinatorial property about these pseudolinear codes that we exploit is that every set of L fixed non-zero codewords of the code C_A, for a random A, are completely independent. This is formalized in Lemma 9.1 below. Note also that, unlike general non-linear codes, codes from a pseudolinear family have a succinct representation, since they can be specified using the $n \times O(kL)$ "generator" matrix A and $\mathrm{poly}(k, \log q)$ sized information about the generating element of $\mathrm{GF}(q^k)$ over $\mathrm{GF}(q)$.

Lemma 9.1 *For every n, k, L, q, an (n,k,L,q)-pseudolinear family is an L-wise independent $(n,k)_q$ family of codes.*

Proof: Since H defines the parity check matrix of a code, say C, that has distance at least $(L+1)$, every set of L columns of H are linearly independent. Indeed, suppose this were not the case. Then there must exist a linear dependence $\alpha_1 H_{a_1} + \ldots + \alpha_L H_{a_L} = \mathbf{0}$ for integers $1 \le a_1 < a_2 < \cdots < a_L < q^k$ and $\alpha_i \in \mathbb{F}_q$ with not all $\alpha_i = 0$. This implies that

[3] The space of random $[n,k]_q$ linear codes has the desired succinctness properties, but however is in general not even 3-wise independent (it is 2-wise (or pairwise) independent, though). This is because for any linear map $E : [q]^k \to [q]^n$, we have $E(x+y) = E(x) + E(y)$ for every $x, y \in [q]^k$.

the non-zero vector $\mathbf{y}$ which has symbol α_i at location a_i for $i = 1, 2, \ldots, L$ and zeroes at all other locations satisfies $H \cdot \mathbf{y} = \mathbf{0}$ and hence belongs to the code C. But the Hamming weight of $\mathbf{y}$ is at most L, a contradiction to the fact that the distance of C is at least $(L+1)$.

Now consider any L non-zero codewords corresponding to messages $a_1, a_2, \ldots, a_L$ where each $a_i \in [1, q^k)$. They are encoded into the codewords $\mathbf{c_i} = A \cdot H_{a_i}$. Since the various H_{a_i} are linearly independent, for a random matrix A, the various $\mathbf{c_i}$'s are completely independent. This follows from the general fact that the images of a set $S = \{\mathbf{v_1}, \ldots, \mathbf{v_L}\}$ of linearly independent vectors in $\mathbb{F}_q^m$, under a linear transformation defined by a random $n \times m$ matrix A, are completely independent. This fact is easy to prove since the $\mathbf{v_i}$'s, being linearly independent, can be mapped by an invertible linear map into the standard basis vectors $\mathbf{e_1}, \mathbf{e_2}, \ldots, \mathbf{e_L}$, and the mutual independence of $A \cdot \mathbf{e_1}, A \cdot \mathbf{e_2}, \ldots, A \cdot \mathbf{e_L}$ for a completely random A is obvious. $\Box$

Remark: We note here that one of the standard constructions of d-wise independent binary random variables (say, $X_1, X_2, \ldots, X_n$) uses arguments similar to the above (cf. [AS92, Chap. 15], [Jof74]). It also proceeds by the construction of a set $S \subseteq \{0,1\}^a$, where $a = O(d \log n)$, consisting of n vectors with the property that any subset of d vectors in S are linearly independent. The set S is picked to be the columns of a parity check matrix of a binary code of blocklength n, dimension $(n - a)$, and minimum distance at least $(d+1)$. The random variable X_i is defined by picking a random vector in $\{0,1\}^a$ and taking its dot product with the i'th vector in S. The fact that any d of the vectors in S are linearly independent translates into the d-wise independence of the X_i's. Using parity check matrices of appropriate BCH codes, gives d-wise independent sample spaces of $O(n^{\lfloor d/2 \rfloor})$ size. This size is in fact optimal, up to a constant factor, cf. [AS92, Chap. 15].

We next define the notion of an infinite family of (L, q)-pseudolinear codes of increasing blocklength. Since we are interested in the asymptotic performance of codes, we will be interested in such code families of a certain rate.

Definition 9.4 *An infinite family of* (L, q)-pseudolinear codes $\mathcal{C}_{L,q}$ *is obtained by picking codes* $\{C_{A_i}\}_{i \geq 1}$ *of blocklengths* n_i *(with* $n_i \to \infty$ *as* $i \to \infty$*) where* C_{A_i} *belongs to the* (n_i, k_i, L, q)*-pseudolinear family.*

9.3.2 Probabilistic constructions of good, list decodable pseudolinear codes

We now analyze the list decodability properties of random pseudolinear codes and use it to prove the existence of pseudolinear codes with a certain trade-off between rate and list decodability. We stress that all existential results of this section are in fact "high probability results"; in other words, a random pseudolinear code with appropriate parameters achieves the claimed rate and list decodability properties with $(1 - o(1))$ probability. We will use this fact implicitly when we use the codes guaranteed by this section in later (probabilistic) code constructions.

Lemma 9.2 *For every prime power* $q \geq 2$*, every integer* ℓ*,* $1 \leq \ell \leq q$ *and* $L > \ell$*, and every* α*,* $0 < \alpha < 1$*, there exists an infinite family of* (L, q)*-pseudolinear codes of rate* r *given by*

$$r \geq \frac{1}{\lg q}\Big(\alpha \lg(q/\ell) - H(\alpha) - H(\ell/q) \cdot \frac{q}{L}\Big) - o(1) \ , \tag{9.1}$$

such that every code in the family is (α, ℓ, L)*-list recoverable. (Recall that for* $0 \leq x \leq 1$*,* $H(x) = -x \lg x - (1-x) \lg(1-x)$ *denotes the binary entropy function of* x*).*

Proof: The proof follows by employing the probabilistic method. Let n be large enough and r be as in the statement of the lemma. We will show that a code C picked at random from an (n, rn, L, q)-pseudolinear family is (α, ℓ, L)-list recoverable with high probability.

Let us estimate the probability that the code C is not (α, ℓ, L)-list recoverable. Fix a choice of $\mathcal{L}_i$, $1 \le i \le n$, where each $\mathcal{L}_i$ is a subset of $[q]$ of size ℓ. The probability that there exist some L non-zero codewords in C each of which has some element from $\mathcal{L}_i$ in its i'th coordinate for at least αn values of i, is clearly at most

$$\binom{q^{rn}-1}{L}\left(\binom{n}{\alpha n}\left(\frac{\ell}{q}\right)^{\alpha n}\right)^{L} . \tag{9.2}$$

Even pessimistically assuming that the zero codeword of C has an element from $\mathcal{L}_i$ for at least αn values of i, the above still upper bounds the probability that there exist $(L+1)$ codewords in C that violate (α, ℓ, L)-list recoverability for this choice of $\mathcal{L}_i$'s. By a union bound on the number of collections of the $\mathcal{L}_i$'s, the overall probability that C is *not* (α, ℓ, L)-list recoverable is at most $\binom{q}{\ell}^n$ times the above quantity, and hence is at most

$$\binom{q}{\ell}^{n} \cdot q^{rnL} \cdot 2^{H(\alpha)nL} \cdot 2^{-\lg(q/\ell)\alpha nL} \tag{9.3}$$

which is easily seen to be $o(1)$ for r as in Equation (9.1). Hence there is a $(1 - o(1))$ probability that the code C has the claimed list recoverability properties. $\square$

Corollary 9.3 *Let $\alpha > 1$ be an arbitrary constant. Then there exist positive constants a_α, b_α such that for every $\varepsilon > 0$, there exist $q = O(1/\varepsilon^2)$, $L = a_\alpha/\varepsilon$ and a family of (L, q)-pseudolinear codes of rate b_α which is $(\alpha, 1/\varepsilon, L)$-list recoverable.*

Proof: Follows by a straightforward substitution of $\ell = 1/\varepsilon$ and $q = O(1/\varepsilon^2)$ in the bound of Equation (9.1). $\square$

We now obtain the following results for the "usual" notion of list decodability.

Lemma 9.4 *For every prime power $q \ge 2$, every p, $0 < p < 1$, and every integer $L \ge 2$, there exists an infinite family $\mathcal{C}_{L,q}$ of (L, q)-pseudolinear codes of rate r given by*

$$r \ge 1 - H_q(p) - \frac{1}{L} - o(1) ,$$

such that $\mathsf{LDR}_L(\mathcal{C}_{L,q}) \ge p$.

Proof: The proof follows by an application of the probabilistic method similar to that of Lemma 9.2. Let us pick a code C at random from an (n, rn, L, q)-pseudolinear family where n is large enough and r is as in the statement of the lemma. Let us estimate the probability that C is *not* (pn, L)-list decodable. In this case there must be some L non-zero codewords of C all of which lie within a Hamming ball of radius pn. Since any L non-zero codewords of C are mutually independent, the probability of this happening for a fixed Hamming ball $B_q(\mathbf{x}, pn)$ is at most

$$\binom{q^{rn}-1}{L} \cdot \left(\frac{q^{H_q(p)n}}{q^n}\right)^{L}$$

since $|B_q(\mathbf{x}, pn)| \le q^{H_q(p)n}$. The probability that this happens for *some* $\mathbf{x} \in [q]^n$ is thus at most

$$q^n q^{rnL} q^{(H_q(p)-1)Ln}$$

which is $o(1)$ for r as in the statement of the lemma. Hence a random pseudolinear code of rate r is (pn, L)-list decodable with high probability. $\Box$

Corollary 9.5 *Let $a > 1$ be an arbitrary constant. Then there exist constants $b_a, c_a > 1$ such that for every $\varepsilon > 0$ the following holds: let $q = O(1/\varepsilon^a)$ and $L = b_a/\varepsilon$. Then there exists a rate ε/c_a family of (L, q)-pseudolinear codes PL_ε which satisfies $\mathsf{LDR}_L(\mathrm{PL}_\varepsilon) \geq 1-\varepsilon$.*

List recoverability of random linear codes

We now state the version of Lemma 9.2 that applies to random *linear* codes. This can be viewed as the generalization of Theorem 5.3 (from Chapter 5), which analyzed the list decodability of random linear codes, to the list recoverability situation. The result for linear codes will be used in the multi-concatenated code construction in Section 9.5 (specifically in the proof of Lemma 9.15).

Lemma 9.6 *For every prime power $q \geq 2$, every integer ℓ, $1 \leq \ell \leq q$ and $L > \ell$, and every α, $0 < \alpha < 1$, there exists an infinite family of linear codes of rate r given by*

$$r \geq \frac{1}{\lg q}\Big(\alpha \lg(q/\ell) - H(\alpha) - H(\ell/q) \cdot \frac{q}{\log_q(L+1)}\Big) - o(1) \ , \tag{9.4}$$

such that every code in the family is (α, ℓ, L)-list recoverable.

Proof: The proof follows along the lines of Lemma 9.2 by analyzing the performance of a linear code defined by a random $(n \times rn)$ generator matrix over $\mathbb{F}_q$. If some set of $(L+1)$ codewords violate the (α, ℓ, L)-list recoverability property, then there must be at least $\log_q(L+1)$ codewords among them that correspond to encodings of linearly independent messages in $\mathbb{F}_q^{rn}$. It therefore suffices to prove an upper bound on the probability that some set of $\log_q(L+1)$ *linearly independent* messages are mapped into codewords that violate the $(\alpha, \ell, \log_q(L+1))$-list recoverability property. Since, for a random linear code the codewords associated with a set of linearly independent messages are all *mutually independent* (cf. Lemma 9.1), the analysis of Lemma 9.2 goes through with $\log_q(L+1)$ taking the place of L. The claimed bound then follows. $\Box$

Corollary 9.7 *Let $\alpha > 1$ be an arbitrary constant. Then there exist positive constants a_α, c_α such that for every $\varepsilon > 0$, there exist $q = O(1/\varepsilon^2)$, $L = q^{c_\alpha/\varepsilon}$ and a family of q-ary* linear *codes of rate a_α which is $(\alpha, 1/\varepsilon, L)$-list recoverable.*

9.3.3 Derandomizing constructions of pseudolinear codes

One straightforward way to "constructivize" or "derandomize" the probabilistic result of Lemmas 9.2 and 9.4 is by a brute-force search over all codes in an (n, k, L, q)-pseudolinear family. Note that checking whether a fixed $(n, k)_q$ pseudolinear code has the necessary (α, ℓ, L)-list recoverability or (pn, L)-list decodability properties can be done by a search over all "received words" and over all codewords in $q^{O(\ell n + k)}$ and $q^{O(n)}$ time, respectively. However, going over all possible $(n, k)_q$ pseudolinear codes involves going over all $n \times O(kL)$ "generator" matrices and this requires $q^{O(knL)}$ time. Hence a naive derandomization of the probabilistic constructions of $(n, k)_q$ pseudolinear codes from the previous section will take $q^{O(knL)}$ time. This is prohibitive even for the blocklengths for inner codes. For example, if we wish to use a pseudolinear code as an inner code in a concatenation scheme with outer code being a Reed-Solomon code over a polynomially large field, then the dimension of the pseudolinear code will be logarithmic in the overall blocklength. The naive derandomization will take quasi-polynomial time in such a case, while we would clearly prefer a polynomial time deterministic construction. We next demonstrate how one can find codes in a (n, k, L, q)-pseudolinear family with the properties claimed in Lemmas 9.2 and 9.4 in $q^{O(kL+n\ell)}$ time. Applied to the above-mentioned concatenated code setting, this will enable polynomial time construction of the concatenated code, since ℓ, L, q will be constants and n, k will be logarithmic in the overall blocklength.

The basic idea is to derandomize the probabilistic constructions using the method of conditional expectations. Since the method is quite standard, we only discuss informally how to apply it to our context.[4] We focus on Lemma 9.2, and the result for Lemma 9.4 is similar. To derandomize the result of Lemma 9.2, we will successively find the n rows of a "good" $n \times O(kL)$ matrix A such that the associated code C_A in the pseudolinear family $\mathcal{F}(n, k, L, q)$ is (α, ℓ, L)-list recoverable. (Here and in what follows $k = rn$ is the dimension of the code.)

Assume that for some $1 \leq s \leq n$, the first $(s-1)$ rows of A have been picked to be $\mathbf{a_1}, \ldots, \mathbf{a_{s-1}}$ where each $\mathbf{a_i} \in \mathbb{F}_q^{O(kL)}$. We pick $\mathbf{a_s}$ so that it minimizes a certain conditional expectation by searching among all the $q^{O(kL)}$ possible choices for $\mathbf{a_s}$.

The relevant expectation that we bound is the following. For each (ordered) collection $\mathcal{D}$ of n "lists" $\mathcal{L}_i \subseteq \mathbb{F}_q$ with $|\mathcal{L}_i| = \ell$ for each i, $1 \leq i \leq n$, each set of L (non-zero) codewords (given by a subset $T = \{x_1, \ldots, x_L\} \subseteq \mathbb{F}_q^k$ of size L) of the pseudolinear code, and each (ordered) collection $\mathcal{S}$ of L subsets $S_1, \ldots, S_L \subseteq [n]$ with each $|S_j| = \alpha n$, define an indicator random variable $I(\mathcal{S}, \mathcal{D}, T)$ as follows. $I(\mathcal{S}, \mathcal{D}, T)$ equals 1 if, for each j, $1 \leq j \leq L$, the codeword corresponding to $x_j \in T$ agrees with an element of $\mathcal{L}_i$ for each of the αn values of $i \in S_j$. Otherwise, $I(\mathcal{S}, \mathcal{D}, T) = 0$. In words, $I(\mathcal{S}, \mathcal{D}, T) = 1$ iff the setting $\mathcal{S}$, $\mathcal{D}$, T shows a "counterexample" to the code that we construct being (α, ℓ, L)-list recoverable (and is thus a "bad" event that we wish to avoid).

The random variable we consider in order to apply the method of conditional expectations is

$$X(\alpha, \ell, L) = \sum_{\mathcal{S}, \mathcal{D}, T} I(\mathcal{S}, \mathcal{D}, T) \ . \tag{9.5}$$

We will exploit linearity of expectation to compute the conditional expectations of $X(\alpha, \ell, L)$. The initial expectation of each $I(\mathcal{S}, \mathcal{D}, T)$ (taken over the random choice of all rows $\mathbf{r_i}$ of A, where $1 \leq i \leq n$) clearly equals $(\frac{\ell}{q})^{\alpha n L}$ since the events for the various codewords in T are independent (by the L-wise independence property of the code). Multiplying this by the number of choices of $\mathcal{S}, \mathcal{D}, T$, we get (as in the proof of Lemma 9.2) that the initial expectation of $X(\alpha, \ell, L)$ is exponentially small (and in particular there exists a code with $X(\alpha, \ell, L) = 0$, or in other words which is (α, ℓ, L)-list recoverable).

Once we condition on the first s rows of A being fixed to, say, $\mathbf{a_1}, \ldots, \mathbf{a_s}$, the expected value of $I(\mathcal{S}, \mathcal{D}, T)$ taken over the random choices of the remaining $(n-s)$ rows $\mathbf{r_1}, \ldots, \mathbf{r_{n-s}}$ can still be exactly computed. Indeed, the first s coordinates of each of the codewords corresponding to each $x_j \in T$, for $1 \leq j \leq L$, are now fixed, and one can compute for each of them the number of coordinates in $S_j \cap \{1, 2, \ldots, s\}$ for which the codeword agrees with an element from the associate list $\mathcal{L}_i$. Thus, for each x_j, we can exactly compute the probability that the associated codeword will agree with an element from $\mathcal{L}_i$ for each $i \in S_j$

[4]The reader can find a discussion of the method of conditional expectations, for example, in [AS92, Chap. 15].

when the remaining rows are picked at random. By the L-wise independence property, we can then simply multiply the probabilities for the various x_j's to estimate the conditional expectation of $I(\mathcal{S}, \mathcal{D}, T)$. We can do so for each of $\binom{n}{\alpha n}^L \cdot \binom{q}{\ell}^n \cdot \binom{q^k - 1}{L}$ choices of $(\mathcal{S}, \mathcal{D}, T)$, and then add up all these expectations to exactly compute the conditional expectation of X. This is of course sufficient to make the best choice for $\mathbf{a_s}$, given that $\mathbf{a_1}, \ldots, \mathbf{a_{s-1}}$ have already been picked. Once we pick $\mathbf{a_i}$, for $1 \le i \le n$, we have the required pseudolinear code that satisfies the property of Lemma 9.2.

Applying the above to the case $q = O(1/\varepsilon^2)$, $\ell = 1/\varepsilon$ and $L = O(1/\varepsilon)$, one can thus prove the following lemma, which is one of the main results we need for our later constructions. This is the "constructive" version of Corollary 9.3. The alphabet size q can actually be made $O(1/\varepsilon^c)$ for any $c > 1$, but since this will not be important to us, we state the result for an alphabet size which is at least $\Omega(1/\varepsilon^2)$. The claims about the representation size and encoding follow since any member of an (n, k, L, q)-pseudolinear family can be represented by an $n \times O(kL)$ matrix over $\mathbb{F}_q$ and encoding involves multiplying a vector in $\mathbb{F}_q^{O(kL)}$ with this matrix. The lower bound claimed on the rate follows from Equation (9.1) after a simple calculation.

Lemma 9.8 *For every α, $0 < \alpha < 1$, and for all large enough constants $c > 1$, there exists a positive constant $a_\alpha \ge \frac{1}{3}(\alpha - 1/c)$ such that for all small enough $\varepsilon > 0$ the following holds. For all prime powers $q = \Omega(1/\varepsilon^2)$, there exist $L = c/\varepsilon$ and a family $\mathrm{PL}_\varepsilon^{(1)}$ of (L, q)-pseudolinear codes of rate a_α, such that a code of blocklength n in the family has the following properties:*

(a) it is $(\alpha, 1/\varepsilon, L)$-list recoverable,

(b) it is constructible in deterministic time $q^{O(n\varepsilon^{-1})} = 2^{O(n\varepsilon^{-1}\log q)}$ or with high probability in randomized $O(n^2\varepsilon^{-1}\log q)$ time (i.e., the constructed code will have the list recoverability property claimed in (a) with high probability), and

(c) it can can be represented in $O(n^2\varepsilon^{-1}\log q)$ space, and encoded using $O(n^2\varepsilon^{-2})$ operations over $\mathbb{F}_q$.

We also get the following constructive version of Corollary 9.5 by applying the same derandomization procedure.

Lemma 9.9 *Let $a > 1$ be an arbitrary constant. Then there exist constants $b_a, c_a > 1$ such that for every $\varepsilon > 0$ the following holds. For all prime powers $q = \Omega(1/\varepsilon^a)$, there exist $L = b_a/\varepsilon$ and a family $\mathrm{PL}_\varepsilon^{(2)}$ of (L, q)-pseudolinear codes of rate at least ε/c_a, such that a code of blocklength n in the family has the following properties:*

(a) it is $((1-\varepsilon)n, L)$-list decodable.

(b) it is constructible in deterministic time $q^{O(n)} = 2^{O(n\log q)}$, or with high probability in randomized $O(n^2\log q)$ time, and

(c) it can can be represented in $O(n^2\varepsilon^{-1}\log q)$ space, and encoded using $O(n^2\varepsilon^{-2})$ field operations over $\mathbb{F}_q$.

A similar result for linear codes. We now state a result analogous to Lemma 9.8 for the case of linear codes.

Lemma 9.10 *For every constant α, $0 < \alpha < 1$, and for all large enough constants $c > 1$, there exists a positive constant $a_\alpha \geq \frac{1}{3}(\alpha - 1/c)$ such that for all small enough $\varepsilon > 0$ the following holds. For all prime powers $q = \Omega(1/\varepsilon^2)$, there exists a family of q-ary linear codes of rate a_α, such that a code of blocklength n in the family has the following properties:*

(a) it is $(\alpha, 1/\varepsilon, q^{c/\varepsilon})$-list recoverable,

(b) it is constructible in deterministic time $q^{O(n\varepsilon^{-1})}$, or with high probability in randomized $O(n^2 \log q)$ time, and

(c) it can can be represented in $O(n^2 \log q)$ space, and encoded using in $O(n^2)$ operations over $\mathbb{F}_q$.

Proof: The claimed parameters follow by substituting $\ell = 1/\varepsilon$, $q = \Omega(1/\varepsilon^2)$, and $L = q^{c/\varepsilon}$ in Lemma 9.6. Note that since the code is linear, it can be represented using its generator matrix, which takes $O(n^2)$ entries in $\mathbb{F}_q$. The only non-trivial thing to check is the claimed deterministic construction time. A naive derandomization will involve trying out all $(n \times a_\alpha n)$ generator matrices, and this will take $q^{O(n^2)}$ time (the verification of the list recoverability property can be done in $q^{O(n\varepsilon^{-1})}$ time). However, as in the case of pseudolinear codes, one can use the method of conditional expectations to get a faster derandomization of the probabilistic construction of Lemma 9.6. This will involve picking the n rows of the generator matrix in sequence, each time searching for the best row from $\mathbb{F}_q^{a_\alpha n}$ that minimizes a certain conditional expectation. The relevant conditional expectations can be computed in $q^{O(n\varepsilon^{-1})}$ time. Hence, the total time required to find a generator matrix that defines a code with the required properties is $q^{O(n\varepsilon^{-1})}$. We omit the details which are very similar to the derandomization of the pseudolinear case. □

Remark concerning alphabet size. Even though the above results are stated for code families over a fixed constant-sized alphabet, a variant of it holds equally well also for alphabet size that grows with the length of the code (in some sense the large alphabet only "helps" these results; note also that the statements of Lemmas 9.8, 9.9, and 9.10 only pose lower bounds on q). This fact is later exploited in our multi-concatenated code constructions from Section 9.5, where we shall make use of such codes for q which is of the form 2^{n^p} for some integer p (n being the blocklength of the code). It is also used in the next section where we show how pseudolinear codes over such large alphabets can be decoded in time significantly better than a brute-force search over all codewords. It is in fact this construction that is used in Section 9.5.

Remark concerning "density" of the codes in the families. Since the existence results claimed in the previous several lemmas are proved by a straightforward application of the probabilistic method, it follows that there exist such codes with *any* (large enough) dimension one seeks (and the properties such as rate and list decodability stay as claimed in the lemmas). We do not explicitly state this fact in the results, but the result of the next section is conveniently stated by fixing the dimension, and hence we explicitly state that it achieves any desired dimension for the codes it constructs. In its proof, as well as in other proofs, we will implicitly use that this fact also holds for the codes from the several previous lemmas.

9.3.4 Faster decoding of pseudolinear codes over large alphabets

The naive algorithm to $(\alpha, 1/\varepsilon, O(1/\varepsilon))$-list recover the pseudolinear codes from Lemma 9.8 is to simply run over all the $q^{O(n)}$ codewords and output only those which satisfy the list recoverability requirement. This takes $q^{\Omega(n)}$ time. In Section 9.5, we will use pseudolinear codes over large alphabets (exponential in the blocklength) in a multi-concatenated scheme, in a hope of getting sub-exponential decoding algorithms for the final code that we construct. But the $q^{\Omega(n)}$ runtime for the decoding is prohibitive for such an application due to the huge value of q.

We now present a code construction that combines pseudolinear codes along with a "parallel" encoding by a linear code to improve the decoding time for codes over very large alphabets. For want of a better term, we refer to these codes as "large alphabet pseudolinear codes". Each symbol of the final encoding will be the "juxtaposition" of the symbols corresponding to the linear and pseudolinear encodings. The linear component of the encoding will be list recoverable in much faster time than the pseudolinear code. The exact details appear in Lemma 9.11 below. The codes constructed below will be the ones that are used in Section 9.5. The technique of symbol juxtaposition used here will be again used in Section 9.6 of this chapter, and in the next chapter on list decodable erasure codes. We believe that just like pseudolinear codes, it is also an important code design tool to take home from this chapter.

Lemma 9.11 *For every constant α, $0 < \alpha < 1$, and all sufficiently large constants $c > 1$, there exists a constant $b_\alpha \le 6(\alpha - 1/c)^{-1}$ such that $\forall\ \varepsilon > 0$ there exists $q = O(1/\varepsilon^2)$ for which the following holds. For all integers m, s, there exists a code of dimension m and blocklength at most $b_\alpha m$ over $\mathrm{GF}(q^{2s})$ with the following properties:*

(i) It is $(\alpha, 1/\varepsilon, c/\varepsilon)$-list recoverable in $O(s^3 (1/\varepsilon)^{O(m)})$ time.

(ii) It is constructible deterministically in $q^{O(sm\varepsilon^{-1})} = 2^{O(sm\varepsilon^{-1}\log q)}$ time. A probabilistic construction that has the claimed list recoverability property with high probability can be found in $O(m^2(s\varepsilon^{-1} + s^2)\log q)$ time. The code can be encoded in $O(m^2 s^2 \log^2 q)$ time.

Proof: Let $\varepsilon > 0$ be given, and let $q = O(1/\varepsilon^2)$ be a power of two. By Lemma 9.8, we know that for every α and all large enough c, there exists a pseudolinear code over $\mathrm{GF}(q^s)$, say C_1, of dimension $2m$, blocklength $b_\alpha m$ such that C_1 is $(\alpha, 1/\varepsilon, c/\varepsilon)$-list recoverable and is constructible in $2^{O(ms\varepsilon^{-1}\log q)}$ deterministic time or in $O(m^2 s\varepsilon^{-1}\log q)$ probabilistic time. Note that we may assume that $b_\alpha \le 6(\alpha - 1/c)^{-1}$ since the rate of the codes guaranteed by Lemma 9.8 is at least $\frac{1}{3}(\alpha - 1/c)$.

The only known list recovering algorithm for such a pseudolinear code is to perform a brute-force search over all $(q^s)^{2m}$ possible codewords, which takes $(1/\varepsilon)^{O(ms)}$ time. In order to speed up the algorithm, we perform an encoding with a suitable random linear code in *parallel* — each symbol of the final encoding will be the "juxtaposition" of the symbols corresponding to the linear and pseudolinear encodings. The linear code, say C_{lin}, will be a q-ary code of dimension $2ms$ and blocklength $b_\alpha ms$ which is $(\alpha, 1/\varepsilon, q^{c/\varepsilon})$-list recoverable. By the result of Lemma 9.10, such a linear code exists and can be constructed deterministically in $q^{O(ms\varepsilon^{-1})}$ time, or probabilistically in $O(m^2 s^2 \log q)$ time.

By "aggregating" each set of successive s symbols in both the message and its encoding by C_{lin}, we can view C_{lin} as a code over $\mathrm{GF}(q^s)$. Viewed this way, C_{lin} will map $2m$ symbols over $\mathrm{GF}(q^s)$ into $b_\alpha m$ symbols over $\mathrm{GF}(q^s)$. To avoid confusion, let us denote the code C_{lin} viewed as a code over $\mathrm{GF}(q^s)$ by $\tilde{C}_{\text{lin}}$.[5]

We now claim that the resulting code $\tilde{C}_{\text{lin}}$ is $(\alpha, 1/\varepsilon, q^{c/\varepsilon})$-list recoverable in

$$O((1/\varepsilon)^{O(m)} m^3 s^3 \log^{O(1)} q) = O(s^3 (1/\varepsilon)^{O(m)})$$

time. The combinatorial list recoverability property itself follows since C_{lin} is $(\alpha, 1/\varepsilon, q^{c/\varepsilon})$-list recoverable as a code over $\mathrm{GF}(q)$, and the property therefore definitely holds for the code $\tilde{C}_{\text{lin}}$ obtained by viewing C_{lin} as a code over $\mathrm{GF}(q^s)$. To prove the claim about the time complexity for list recovering $\tilde{C}_{\text{lin}}$, we present the following algorithm. The algorithm is simply to try out all possible subsets S of $\alpha b_\alpha m$ positions and for each such choice go over all possible sets T of $(1/\varepsilon)^{O(m)}$ symbols from the input lists. For each such choice of S and T, we find if any codeword of C_{lin} is consistent with these symbols (this is simply an erasure decoding of the linear code). This can be done by solving a linear system over $\mathrm{GF}(q)$ and takes at most $O((2ms)^3 \log^{O(1)} q)$ time since the blocklength of C_{lin} equals $2ms$. Since C_{lin} is $(\alpha, 1/\varepsilon, q^{c/\varepsilon})$-list recoverable, it is definitely true that the number of codewords of C_{lin} consistent with a certain choice of symbols in a fraction α of the positions is at most $q^{c/\varepsilon}$. Finally, we will have to check which of the codewords of C_{lin} actually yield codewords of $\tilde{C}_{\text{lin}}$ that meet the required list recoverability condition. Since each erasure decoding yields at most $q^{c/\varepsilon}$ solutions to check, the total runtime will be the number of choices of S, T multiplied by the time for each erasure decoding of C_{lin}, plus an additional time of roughly $O(q^{c/\varepsilon})$ to prune the list returned by the erasure decoding of C_{lin} . This gives the claimed $O((1/\varepsilon)^{O(m)} s^3)$ runtime.

We now define our final code C^* to be the *juxtaposition* of C_1 and $\tilde{C}_{\text{lin}}$; i.e. to encode a message according to C^*, we encode it using C_1 and $\tilde{C}_{\text{lin}}$ independently to get two strings, say, $\langle a_1, a_2, \ldots, a_t \rangle$ and $\langle b_1, b_2, \ldots, b_t \rangle$, where $t = b_\alpha m$ and each $a_i, b_i \in \mathrm{GF}(q^s)$. The encoding of that message as per C^* will then be $\langle c_1, \ldots, c_t \rangle$, where each $c_i = (a_i, b_i)$ is viewed as an element of $\mathrm{GF}(q^{2s})$. Note that C^* defined this way encodes $2m$ symbols over $\mathrm{GF}(q^s)$ into $b_\alpha m$ symbols over $\mathrm{GF}(q^{2s})$. We may equivalently view C^* as mapping m symbols over $\mathrm{GF}(q^{2s})$ into $b_\alpha m$ symbols over $\mathrm{GF}(q^{2s})$. In other words C^* has dimension m and blocklength $b_\alpha m$ as a q^{2s}-ary code.

Since C_1 is $(\alpha, 1/\varepsilon, c/\varepsilon)$-list recoverable, so is C^* (as would any juxtaposed code that involves C_1). This gives the combinatorial list recoverability property of C^*. To obtain the claim about the algorithmic list recoverability, we will use the "linear" component $\tilde{C}_{\text{lin}}$ of C^*. By the above argument, $\tilde{C}_{\text{lin}}$ can be $(\alpha, 1/\varepsilon, q^{c/\varepsilon})$-list recovered within the claimed runtime. One can then run through the at most $q^{O(1/\varepsilon)}$ messages output by this algorithm and "cross-check" if its encoding by the pseudolinear code C_1 agrees with the respective component of the symbols in the input lists on a fraction α of the positions. By the combinatorial list recoverability property of C_1, at most c/ε of the messages will pass this check. These will be the messages output by the algorithm. The running time of this procedure is dominated by that of the list recovering algorithm for $\tilde{C}_{\text{lin}}$, and is thus $O((1/\varepsilon)^{O(m)} s^3)$.

The encoding time for C^* is dominated by the time to encode the "linear" component. Since the code C_{lin} has both dimension and blocklength at most $O(ms)$, the encoding of C_{lin} takes at most $O((ms)^2 \log^2 q)$ time. This completes the proof of the lemma. □

[5]The code $\tilde{C}_{\text{lin}}$ will not in general be linear over $\mathrm{GF}(q^s)$, but we will only use linearity of C_{lin} over $\mathrm{GF}(q)$.

9.4 The basic expander-based construction of list decodable codes

For the construction in this section, we will use families of graphs with a small bounded degree which nevertheless have strong connectivity properties. Specifically, they will have the property that the neighborhood of any large enough subset of vertices misses a very small fraction of vertices. Such graphs are called *expanders*, due to their strong "vertex-expansion" properties. We mention here that code constructions in Chapter 11 are obtained using similar techniques and also use expander-like graphs, but there we will make use of stronger properties than just vertex-expansion — namely we will use certain isoperimetric properties offered by expanders. We next discuss the necessary facts about expanders.

9.4.1 Preliminaries on Expanders

We need the following basic definition about expanders.

Definition 9.5 *For integers $N, d \geq 1$ and $0 < \varepsilon, \alpha < 1$, an $(N, d, \alpha, \varepsilon)$-expander is a d-regular $N \times N$ bipartite graph $H = (A, B, E)$ (where A, B with $|A| = |B| = N$ are the two sets in the bipartition and E is the edge set), with the property that given any subset $X \subseteq B$ with $|X| \geq \varepsilon|B|$, the number of vertices in A with some neighbor in X is at least $\alpha|A|$.*

The following result on the existence of expander graphs is well known, see for instance [ABN$^+$92, Sec. II] (or [AS92, Chap. 9]) where an explicit construction using the Ramanujan graphs of [LPS88] is discussed.

Fact 9.1 *There is a constant c such that for every $\varepsilon > 0$ and for infinitely many n, there exists an explicitly constructible $(n, c/\varepsilon, 1/2, \varepsilon)$-expander.*

Of course the $1/2$ in the above claim can be changed to any fixed constant $\alpha < 1$. In such a case, the constant c in the degree will depend on α.

9.4.2 Reduction of list decoding to list recoverability using expanders

We now present an elegant and simple reduction of the problem of constructing codes which are efficiently $((1-\varepsilon)n, L)$-list decodable to the problem of constructing codes with efficient $(\alpha, O(1/\varepsilon), L)$-list recoverability, for some fixed constant α, say $\alpha = 1/2$. This idea is at the heart of all our expander-based code constructions that we present in this chapter. It is instructive to point out that the use of the expanders in our constructions is confined to this reduction, and the construction of good list recoverable codes itself is accomplished using other techniques.

The reduction is accomplished by redistributing the symbols of the codewords of a list recoverable code, say C_1, using a expander H, and thus define the codewords of a new code C_2 over a larger alphabet. The list recoverability property of C_1, together with the expansion property of H, will imply the good list decodability of C_2. Given a corrupted received word $\mathbf{r}$ of C_2, one can push the symbols of $\mathbf{r}$ along the edges the expander H to obtain a list of possible symbols for each position of C_1. The expansion property of H will imply that at least a fraction $1/2$ of these lists contain the correct symbol of the codeword of C_1. Now, the list recoverability property of C_1 can be used to complete the decoding. The formal statement of the reduction and the proof follow.

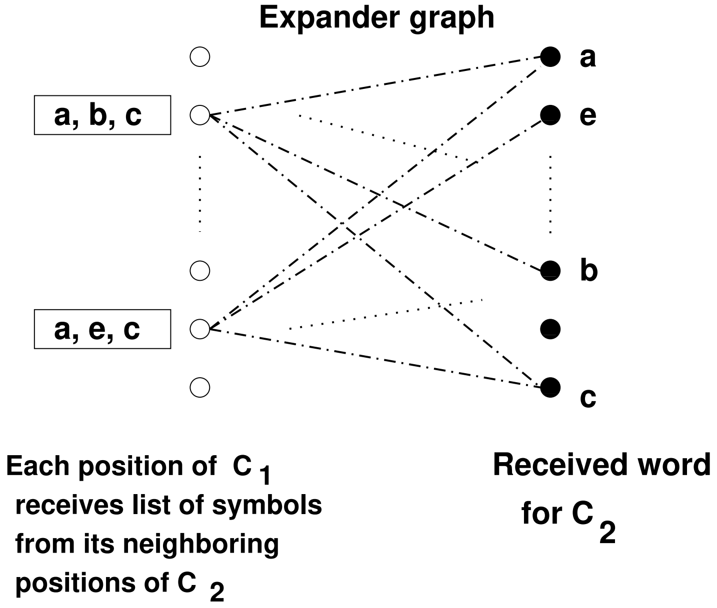

Figure 9-3: Illustration of the decoding algorithm. Each position on the left collects a list of symbols from all its neighbors on the right. These lists are then used as input to the list recovering algorithm for the left code C_1. The expansion property implies that even if the received word for C_2 had several errors, a good fraction of the lists obtained for C_1 contain the correct symbol.

Proposition 9.12 *There exists an absolute constant c such that for every $\varepsilon > 0$ the following holds. Suppose there exists a q-ary code C_1 of blocklength n and rate r that is $(1/2, c/\varepsilon, L)$-list recoverable by an algorithm running in time $O(T(n))$. Further assume that n is such that there exists an $(n, c/\varepsilon, 1/2, \varepsilon)$-expander. Then there exists a code C_2, which is explicitly specified given C_1, and which has the following properties:*

(i) It has blocklength n and rate $\varepsilon r/c$.

(ii) It is defined over an alphabet of size $q^{c/\varepsilon}$.

(iii) It is $((1-\varepsilon)n, L)$-list decodable, and moreover there is an algorithm to list decode C_2 up to a fraction $(1-\varepsilon)$ of errors in time $O(T(n) + n \log q/\varepsilon)$.

Proof: The code C_2 is obtained by distributing the symbols of codewords in C_1 using the edges of an $(n, \Delta, 1/2, \varepsilon)$-expander where $\Delta = c/\varepsilon$. This is in a manner similar to Alon *et al* [ABN$^+$92], who used such a symbol redistribution for the purpose of getting codes with a large (viz., $(1-\varepsilon)$) relative distance. Formally, let $H = ([n], [n], E)$ be an $(n, \Delta, 1/2, \varepsilon)$-expander. For $1 \le j \le \Delta$ and $1 \le i \le n$, denote by $\Gamma_j(i)$ the j'th neighbor (on the left side) of the i'th vertex on the right side of H (we assume some fixed ordering of the neighbors of each node). A codeword $(c_1, c_2, \ldots, c_n)$ of C_1 is mapped into a codeword $(\tilde{c}_1, \ldots, \tilde{c}_n)$ of C_2, where each $\tilde{c}_i \in [q]^\Delta$ is given by $\tilde{c}_i = \langle c_{\Gamma_1(i)}, \ldots, c_{\Gamma_\Delta(i)} \rangle$. The claim about the blocklength, rate and alphabet size of C_2 follow immediately.

The algorithm for list decoding C_2 up to a radius of $(1-\varepsilon)n$ proceeds in two steps. Assume $\mathbf{r}$ is a received word and the goal is to find all codewords of C_2 that are within a

Hamming distance of $(1-\varepsilon)n$ from $\mathbf{r}$. In other words, the goal is to find every message $\mathbf{x}$ that satisfies $\Delta(C_2(\mathbf{x}),\mathbf{r}) \le (1-\varepsilon)n$. In the first step of the decoding, each position of the received word $\mathbf{r}$ "votes" on those positions of the corresponding codeword in C_1 which are adjacent to it in the expander H. This gives for each i, $1 \le i \le n$, a list L_i of at most $\Delta = c/\varepsilon$ elements from $[q]$ for each position of the code C_1. See the illustration in Figure 9-3. In the second step, the $(1/2, c/\varepsilon, L)$-list recovering algorithm for C_1 is run with these lists L_i, $1 \le i \le n$, as input. Finally, for each message output by the list decoder for C_1, we check if its encoding under C_2 agrees with $\mathbf{r}$ in at least εn positions, and if so, we output it.

The time required for the above algorithm is the time for the first "voting" stage, which takes $O(n \log q/\varepsilon)$ time, followed by the time for list recovering C_1, which takes $O(T(n))$ time by hypothesis.

It remains to prove the correctness of the algorithm. Let $\mathbf{x}$ be any message such that $\Delta(C_2(\mathbf{x}),\mathbf{r}) \le (1-\varepsilon)n$. Let $X \subseteq [n]$ be the set of positions where $C_2(\mathbf{x})$ and $\mathbf{r}$ agree. By hypothesis $|X| \ge \varepsilon n$. Define $Y \subseteq [n]$ to be the set of vertices on the left side of H which have a neighbor in X on the right. By the expansion property of H, $|Y| \ge n/2$. Now, clearly for each $i \in Y$, the i'th symbol of $C_1(\mathbf{x})$ is included in the list L_i (since all votes coming from the positions in X are correct, and the symbols in Y are precisely those which receive at least one vote from the positions in X). Therefore, the message $\mathbf{x}$ will be included in the list output by the $(1/2, c/\varepsilon, L)$-list recovering algorithm for C_1, when it is run with the lists L_i as input. Hence, the above algorithm will successfully include $\mathbf{x}$ in the final list it outputs. $\square$

The following states a more general form of the above proposition which states a stronger list recoverability property for C_2 using that of C_1. The proof is identical to the above — at the voting stage of decoding, instead of each position of C_2 passing one vote to each of its neighbors, it passes ℓ votes where ℓ is the number of possible symbols listed for that position. Proposition 9.12 follows with the setting $\ell = 1$.

Lemma 9.13 *There exists an absolute constant c such that for every $\varepsilon > 0$ the following holds. Suppose there exists a q-ary code C_1 of blocklength n and rate r that is $(1/2, c\ell/\varepsilon, L)$-list recoverable by an algorithm running in time $O(T(n))$. Further assume that n is such that there exists an $(n, c/\varepsilon, 1/2, \varepsilon)$-expander. Then there is a code C_2, which is explicitly specified given C_1, with the following properties:*

(i) It has blocklength n and rate $\varepsilon r/c$.

(ii) It is defined over an alphabet of size $q^{c/\varepsilon}$.

(iii) It is (ε, ℓ, L)-list recoverable, and moreover there is an algorithm to (ε, ℓ, L)-list recover C_2 in time $O(T(n) + n\ell \log q/\varepsilon)$.

9.4.3 Codes of rate $\Omega(\varepsilon^2)$ list decodable to a fraction $(1-\varepsilon)$ of errors

We now present our code construction (number 1) which has rate ε^2 and is list decodable in near-quadratic time from up to a fraction $(1-\varepsilon)$ of errors. The formal result is stated in Theorem 9.14.

Before we state and prove this result, we would like to point out one technical point concerning the constructions. Recall the overall structure of all our constructions (Figure 9-2): they use a certain "left code" C and then redistribute symbols of a codeword of C using an expander. There is an implicit assumption here that each side of the bipartite expander has the same number of vertices, say n, as the blocklength of C. The known constructions of Ramanujan graphs (eg. [LPS88, Mar88]) work for infinitely many values of n, but not for *all* sufficiently large n (as would be ideal for our application). However, as discussed in [Spi95, Section 2.4.1], these constructions give a *dense* sequence of graphs, i.e., the sequence of number of vertices $\{n_l\}_{l\geq 1}$ for which the constructions work satisfies $n_{i+1} - n_i = o(n_i)$ for sufficiently large i. As a consequence, Spielman [Spi95] proves that it is possible to get expander graphs of every size with only a moderate loss in expansion, and uses this fact in his constructions of expander codes. The same argument will also work for us. Alternatively, since the sequence of graphs is dense, we can pad each codeword of the left code with a small number of additional 0's so that its blocklength exactly matches the number of vertices in an explicit Ramanujan graph construction, and then apply our construction. This "padding" will affect the relative distance, rate and list decoding radius of the left code only by a negligible amount, and will essentially have no impact on any of the bounds we claim for the overall code construction. Therefore, in order to keep things simple, in our constructions of this chapter, as well as those in Chapter 11, we will ignore the above issue and simply assume that the blocklength of our "left code" and the number of vertices in the "expander" graph match exactly.

Theorem 9.14 *For all $\varepsilon > 0$, there exists a code family with the following properties:*

- *(i) (Rate and alphabet size) It has rate $\Omega(\varepsilon^2)$ and is defined over an alphabet of size $2^{O(\varepsilon^{-1}\log(1/\varepsilon))}$.*
- *(ii) (Constructibility) A description of a code of blocklength N in the family can be constructed in deterministic $N^{O(\varepsilon^{-1})}$ time. A randomized Monte Carlo construction that has the list decodability claimed in (iii) with high probability can be obtained in probabilistic $O(\log^2 N\varepsilon^{-1}\log(1/\varepsilon))$ time.*
- *(iii) (List decodability) A code of blocklength N in the family can be list decoded from up to $(1-\varepsilon)N$ errors in $O(N^2\varepsilon^{-O(1)}\log N)$ time using lists of size $O(1/\varepsilon)$.*

Proof: The basic idea is to first construct a code C with good list recoverability properties by concatenating a Reed-Solomon code C_{RS} of rate $\Omega(\varepsilon)$ with a constant rate inner code C_{in} as guaranteed in Lemma 9.8. We will then apply the construction of Proposition 9.12 to obtain a code list decodable up to a fraction $(1-\varepsilon)$ of errors. Since the rate of the concatenated code is $\Theta(\varepsilon)$, and applying Proposition 9.12 incurs a further ε factor loss in the rate, we will get an overall rate of $\Omega(\varepsilon^2)$. The formal details follow. The basic structure of the construction is depicted in Figure 9-4.

Let m be any sufficiently large integer. Let $q_0 = O(1/\varepsilon^2)$ be a power of 2, and let F be a field of cardinality q_0^m. Let n_0 be in the range $q_0^{m-1} \leq n_0 \leq q_0^m$, $k_0 = \Theta(\varepsilon n_0)$, and C_{RS} be the Reed-Solomon code over F of blocklength n_0 and dimension k_0 (so C_{RS} has rate $\Theta(\varepsilon)$). Let C_{in} be a pseudolinear code over $\mathbb{F}_{q_0}$ that maps m symbols over $\mathbb{F}_{q_0}$ (or, alternatively, a symbol of F) into $n_1 = O(m)$ symbols over $\mathbb{F}_{q_0}$, and further is $(1/4, O(1/\varepsilon), O(1/\varepsilon))$-list recoverable. Such a code C_{in} exists by Lemma 9.8.

Define $C_{\mathrm{RS-in}}$ to be the code obtained by concatenating C_{RS} as outer code with C_{in} as inner code. $C_{\mathrm{RS-in}}$ is a code of blocklength $N = n_0 \cdot n_1 = O(mq_0^m)$ and rate $\Omega(\varepsilon)$ over $\mathbb{F}_{q_0}$. The codewords in $C_{\mathrm{RS-in}}$ can be divided into n_0 blocks of n_1 symbols each, corresponding to the encodings of the n_0 outer Reed-Solomon codeword symbols.

The final code C^* will be obtained from $C_{\mathrm{RS-in}}$ using the construction of Proposition 9.12 (i.e., by redistributing the symbols of a codeword of $C_{\mathrm{RS-in}}$ using an $(N, O(1/\varepsilon), 1/2, \varepsilon)$-expander). It is readily checked that C^*, thus defined, is a code of blocklength N and rate

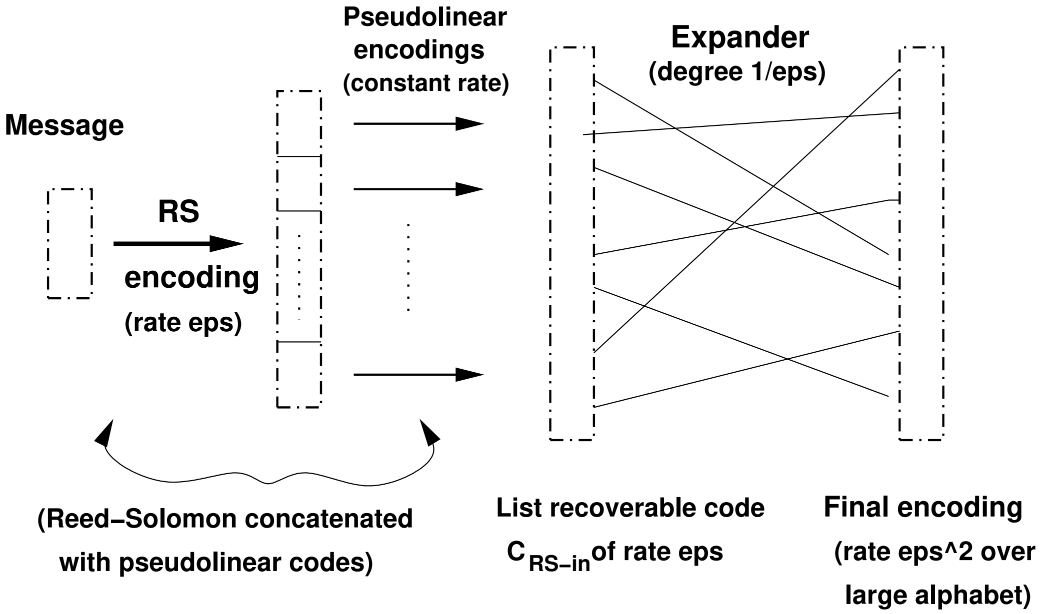

Figure 9-4: Basic structure of code construction that achieves rate $\Omega(\varepsilon^2)$ and list decoding radius $(1-\varepsilon)$. The list recoverability of the concatenated code $C_{\text{RS-in}}$, together with the expander, implies list decodability of the final code from a fraction $(1-\varepsilon)$ of errors.

$\Omega(\varepsilon^2)$ over an alphabet of size $q_0^{O(1/\varepsilon)} = 2^{O(\varepsilon^{-1}\log(1/\varepsilon))}$, proving Part (i) of the claim of the theorem.

The significant component in constructing C^* is finding the inner code C_{in} with the properties guaranteed in Lemma 9.8. Thus C^* can be constructed deterministically in $O(q_0^{O(m\varepsilon^{-1})}) = N^{O(\varepsilon^{-1})}$ time, or probabilistically in $O(m^2\varepsilon^{-1}\log q_0) = O(\log^2 N\varepsilon^{-1}\log(1/\varepsilon))$ time, as claimed in Part (ii) of the theorem statement.

It remains to prove the claim about the list decodability of C^*. For this, it suffices to prove that $C_{\text{RS-in}}$ is $(1/2, O(1/\varepsilon), O(1/\varepsilon))$-list recoverable in $O(N^2)$ time, since then the claim about list decoding C^* from a fraction $(1-\varepsilon)$ of errors will follow from the properties of C^* guaranteed by Proposition 9.12.

Suppose we are given lists L_i each consisting of at most $O(1/\varepsilon)$ elements of $\mathbb{F}_{q_0}$, for $1 \le i \le N$. We will present an $O(N^2\varepsilon^{-O(1)}\log N)$ time algorithm to find all codewords $\langle d_1, d_2, \ldots, d_N\rangle$ of $C_{\text{RS-in}}$ which satisfy $d_i \in L_i$ for at least $N/2$ values of i, $1 \le i \le N$. Recalling that a codeword in $C_{\text{RS-in}}$ comprises of n_0 blocks of n_1 symbols each, the lists L_i can be viewed as lists $L'_{j,s}$ for the possible symbols in position s of the codeword of C_{in} that encodes the j'th symbol of the Reed-Solomon codeword, for $1 \le s \le n_1$ and $1 \le j \le n_0$. Now consider the following list recovering procedure for $C_{\text{RS-in}}$. In the first step, the n_0 inner codes are decoded by brute-force by going over all codewords — namely, for each j, $1 \le j \le n_0$, one produces a list $\hat{L}_j$ of all elements of F whose encoding as per C_{in} contains an element from $L'_{j,s}$ for at least a fraction $1/4$ of the values of s. By the list recoverability property of C_{in} we have $|\hat{L}_j| = O(1/\varepsilon)$ for each j, $1 \le j \le n_0$. Note that all the inner decodings can be performed in $O(n_0^2/\varepsilon)$ time.

In the second step of the decoding, we run the list recovering algorithm for Reed-Solomon codes implied by the result of Theorem 6.17, to find a list $\mathcal{L}$ consisting of all messages $\mathbf{x}$ whose Reed-Solomon encoding contains an element of $\hat{L}_j$ for at least $n_0/4$ values of j, $1 \le j \le n_0$. Specifically, we apply the result of Theorem 6.17 with the choice $n = n_0$, $k = k_0 = O(\varepsilon n_0)$, $\ell \le \max_j |\hat{L}_j| = O(1/\varepsilon)$, and $\alpha = 1/4$ (one can check that the condition $\alpha > \sqrt{2k\ell/n}$ can be met for these values, with suitable constants in the big-Oh notation). The decoding returns lists of size $O(\sqrt{\frac{n_0}{\varepsilon k_0}}) = O(1/\varepsilon)$, and can certainly be performed in $O(n_0^2 \varepsilon^{-O(1)} \log^3(q_0^m)) = O(n_0^2 \varepsilon^{-O(1)} \log^3 n_0)$ time. Since $n_0 = O(N/\log_{q_0} N)$, the time for list recovering the Reed-Solomon code is $O(N^2 \varepsilon^{-O(1)} \log N)$.

The final step prunes the list output by the Reed-Solomon decoder to include only those messages whose encodings as per $C_{\mathrm{RS-in}}$ contain an element of L_i for at least $N/2$ values of i, and then outputs this pruned list. The overall decoding time is dominated by the Reed-Solomon decoding time and is $O(N^2 \varepsilon^{-O(1)} \log N)$.

We now argue the correctness of the list recovering procedure. Let $\mathbf{x}$ be a message whose encoding $C^*(\mathbf{x}) = \langle d_1, d_2, \ldots, d_N \rangle$, where $d_i \in L_i$ for at least $N/2$ values of i. The codeword $\langle d_1, d_2, \ldots, d_N \rangle$ can also be viewed as consisting of symbols $b_{j,s}$ for $1 \le j \le n_0$ and $1 \le s \le n_1$, where $\langle b_{j,1}, b_{j,2}, \ldots, b_{j,s} \rangle$ is the block of the codeword corresponding to the inner encoding of the j'th symbol of $C_{\mathrm{RS}}(\mathbf{x})$. Let $J \subseteq [n_0]$ be the set of all j, $1 \le j \le n_0$, for which $b_{j,s}$ belongs to the corresponding list $L'_{j,s}$ for at least a fraction $1/4$ of values of s in the range $1 \le s \le n_1$. If $d_i \in L_i$ for at least $n_0 n_1/2$ values of i, by a simple averaging argument we get that $|J| \ge n_0/4$. Now, by the $(1/4, O(1/\varepsilon), O(1/\varepsilon))$-list recoverability property of C_{in}, for each $j \in J$, the list $\hat{L}_j$ contains the correct symbol of the Reed-Solomon encoding of the concerned message $\mathbf{x}$. Since $|J| \ge n_0/4$, the condition under which the Reed-Solomon list decoder outputs a message is satisfied by $\mathbf{x}$, and therefore it will output $\mathbf{x}$. Hence the message $\mathbf{x}$ will be included in the list output by the algorithm, as we desired to show. $\square$

9.4.4 Better rate with sub-exponential decoding

In the proof of Theorem 9.14, we used an outer Reed-Solomon code over a field of size linear in the blocklength. This implied that the dimension of the inner pseudolinear code was at most $\log N$, enabling a deterministic polynomial time algorithm to find the necessary pseudolinear code. We now indicate how at the cost of sub-exponential (about $2^{O(\sqrt{N})}$) construction and decoding time, we can improve the rate of the construction of Theorem 9.14 from $\Omega(\varepsilon^2)$ to $\Omega(\varepsilon)$, which is optimal up to constant factors. We will keep the discussion informal since in the next section we will generalize this result and state formal theorems anyway.

The idea is to perform the same construction as in Theorem 9.14, except we use Reed-Solomon codes of *constant* (independent of ε) rate, blocklength $\sqrt{n}$, over an alphabet of size $q_0^{\sqrt{n}}$ where $q_0 = O(1/\varepsilon^2)$. For the inner code, we use a constant rate $(1/4, O(1/\varepsilon), O(1/\varepsilon))$-list recoverable pseudolinear code of dimension $\sqrt{n}$ (i.e., same as in Theorem 9.14, except with larger dimension). Note that the concatenated code also has constant rate, and the dominant component in its construction is once again the pseudolinear code construction, which takes $2^{O_\varepsilon(\sqrt{n})}$ time to perform deterministically, and $O_\varepsilon(n)$ time to perform probabilistically (here by the O_ε notation we are hiding also constant factors that depend on ε). We claim that the concatenated code can be $(1/2, O(1/\varepsilon), L)$-list recovered in $2^{O_\varepsilon(\sqrt{n})}$ time (for $L = 2^{O_\varepsilon(\sqrt{n})}$). At the first step, all the inner codes are $(1/4, O(1/\varepsilon), O(1/\varepsilon))$-list recovered by a brute-force search over all codewords in $q_0^{O(\sqrt{n})}$ time. This passes lists of size $O(1/\varepsilon)$ for the possible symbol at each position of the Reed-Solomon codeword. The decoding is completed by going over every set of a fraction $1/4$ of these lists and every choice of symbols from each of these lists, and for each of them checking if there is a Reed-Solomon codeword consistent with those symbols. This brute-force procedure takes about $(1/\varepsilon)^{O(\sqrt{n})}$ time and succeeds in $(1/4, O(1/\varepsilon), L)$-list recovering the Reed-Solomon code. The correctness of this procedure follows using arguments similar to those of Theorem 9.14.

We thus have a constant rate code which is $(1/2, O(1/\varepsilon), L)$-list recoverable in $2^{O_\varepsilon(\sqrt{n})}$ time. Using this in the construction of Proposition 9.12, we can get a rate $\Omega(\varepsilon)$ code C^* list decodable in $2^{O_\varepsilon(\sqrt{n})}$ time from up to a fraction $(1-\varepsilon)$ of errors, as we desired to show.

To get binary codes, we can concatenate C^* with a binary code of rate $\Omega(\varepsilon^2)$ which has list decoding radius $(1/2 - O(\varepsilon))$ for a list size of $O(1/\varepsilon^2)$. Such codes exist by the result of Theorem 5.5 (from Chapter 5). This gives binary codes of rate $\Omega(\varepsilon^3)$ that are list decodable in $2^{O_\varepsilon(\sqrt{n})}$ time from up to a fraction $(1/2 - \varepsilon)$ of errors. Note that the rate is better than the result of Theorem 8.11 that achieved a rate of $\Omega(\varepsilon^4)$. However, the construction and decoding time are no longer polynomial in the blocklength.

In the next section, we present a more complicated scheme to improve the decoding time to $2^{O(N^\gamma)}$ for any desired $\gamma > 0$. The spirit of the construction is the same as in this section; the details are however more complicated.

9.5 Constructions with better rate using multi-concatenated codes

We now introduce a code construction where an outer Reed-Solomon code is concatenated with multiple levels of inner codes (as guaranteed by Lemma 9.2, albeit over large, growing sized alphabets which decrease in size from the outermost to innermost levels). We call such codes *multi-concatenated codes*, which are discussed in Section 9.5.1. We will then, in Section 9.5.2, use these codes to prove Theorem 9.16 which allows us improve the rate (from Theorem 9.14) by an ε factor at the expense of the decoding time becoming sub-exponential in the blocklength. This gives our construction (2a), and yields codes of the optimal $\Omega(\varepsilon)$ rate that have list decoding algorithms of "reasonable" complexity for correcting a fraction $(1-\varepsilon)$ of errors. Following this, in Section 9.5.3, we will concatenate these codes with appropriate binary codes to get our construction (2b), i.e., binary codes of rate $\Omega(\varepsilon^3)$ list decodable in sub-exponential time from up to a fraction $(1/2 - \varepsilon)$ of errors.

9.5.1 The basic multi-concatenated code

We now describe the construction of multi-concatenated codes and their properties. This is stated formally in the lemma below. The result is similar to Lemma 9.8 in terms of the parameters of the codes it guarantees. In fact, for the case $p = 1$, the result is in fact just that of Lemma 9.8 (with the claimed decoding time being that of the naive decoding algorithm that does a brute-force search over all possible codewords).

For larger values of p, the construction is somewhat messy. The result for larger values of p is necessary only to improve the decoding time from the $2^{O(\sqrt{N})}$ bound that was presented in Section 9.4.4 to $2^{O(N^\gamma)}$. The reader might want to take the result of the lemma below as a black-box in the first reading and come back to its proof if interested after seeing its applications in Sections 9.5.2 and 9.5.3 (the case $p = 1$ for those applications gives precisely the constructions outlined in Section 9.4.4).

Lemma 9.15 *For every $p \geq 1$ and all sufficiently small $\varepsilon > 0$, there exist a code family with the following properties:*

(i) (Rate and alphabet size) The family has rate $2^{-O(p^2)}$ and is defined over an alphabet of size $O(1/\varepsilon^2)$.

(ii) (List decodability property) Each member of the code family is $(\frac{1}{2}, \frac{1}{\varepsilon}, \frac{2^{O(p^2)}}{\varepsilon})$-list recoverable. Furthermore such list decoding can be accomplished in $2^{O(N^{1/p}\log(1/\varepsilon))}$ time, where N is the blocklength of the concerned code.

(iii) (Constructibility) A code of blocklength N in the family can be constructed in probabilistic $O(N^2\log(1/\varepsilon))$ time (the code will have the list decodability claimed in (ii) with high probability). A deterministic construction can be obtained in $2^{O(N\varepsilon^{-1}\log(1/\varepsilon))}$ time. Also, encoding can be performed in $O(N^2\log^{O(1)}(1/\varepsilon))$ time.

Proof Idea. The basic idea is to use p levels of large alphabet pseudolinear codes as guaranteed by Lemma 9.11 in a suitable concatenation scheme. These codes will be defined over progressively decreasing alphabet size. The outermost code will be a constant-rate code of blocklength $O(N^{1/p})$ defined over an alphabet of size $q^{2\cdot N^{(p-1)/p}}$ (where $q = O(1/\varepsilon^2)$). Each symbol of this codeword will then be encoded by another code guaranteed by Lemma 9.11, this time over a smaller alphabet $\mathrm{GF}(q^{2\cdot N^{(p-2)/p}})$, but again of blocklength $O(N^{1/p})$ and constant rate. Each symbol of this encoding will be further encoded by a similar constant rate code of blocklength $O(N^{1/p})$, but over an even smaller alphabet $\mathrm{GF}(q^{2\cdot N^{(p-3)/p}})$, and so on. This will continue for several more levels till the alphabet size is down to $\mathrm{GF}(q^{2\cdot N^{1/p}})$. Finally each of the field symbols is encoded by one final constant rate pseudolinear code over $\mathrm{GF}(q)$ of dimension $2N^{1/p}$.

The big plus of using p levels is that the code at each level has dimension and blocklength $O(N^{1/p})$. Since the decoding time guaranteed by Lemma 9.11 was about $(1/\varepsilon)^{O(m)}$ where m was the dimension, we can exploit the "fast" decoding of the codes at each level to give a decoding algorithm for the overall multi-concatenated code with runtime exponential in $N^{1/p}$, or sub-exponential in N.

All in all, given a list L_i of $O(1/\varepsilon)$ symbols of $\mathrm{GF}(q)$ for each of the N positions of the final codeword, the successive decodings pass up a list of $O(1/\varepsilon)$ symbols (with larger and larger constants in the big-Oh notation) for each position of each pseudolinear codeword. Finally, after p levels of decoding, we will recover a list of at most $O(1/\varepsilon)$ codewords which includes all codewords that agree with an element of L_i for at least $N/2$ values of i.

The formal proof given below just follows the above idea, though it necessarily involves a somewhat careful choice of parameters to ensure that the decodings all work together to give the claimed list recoverability property. The reader satisfied with the above proof idea should feel free to skip it.

Proof: Let q be a power of 2 with $q = O(1/\varepsilon^2)$ – we will define a code over $\mathbb{F}_q$. We will describe the code family by describing a code C_p that encodes $\mathbf{x} \in \mathbb{F}_q^n$ into $C_p(\mathbf{x}) \in \mathbb{F}_q^{n\cdot 2^{O(p^2)}}$, for any large enough n which is of the form $n = 2 \cdot m^p$ for some integer m. The code C_p is described below inductively for increasing values of p.

Code Description. For $p = 1$, the code C_1 will be a q-ary $(\alpha_1, 1/\varepsilon, c/\varepsilon)$-list recoverable code that encodes a string of length $2m$ over $\mathbb{F}_q$ into a codeword of length $a_1 m$ (for suitable constants $a_1, c > 1$ and $\alpha_1 < 1$). Such a code is guaranteed to exist by Lemma 9.8.

For $p > 1$, the code C_p will be a q-ary code of dimension $2m^p$. We defined the code C_p inductively using C_{p-1} and a p'th level code G_p defined as follows. G_p will be a code defined over an alphabet Σ_p of size $q^{2m^{p-1}}$ as guaranteed by Lemma 9.11 (using the choice $s = m^{p-1}$ in that lemma). Specifically, G_p has dimension m and blocklength $a_p m$, where a_p is a constant that depends only on p but is independent of ε. Moreover, G_p is $(\alpha_p, c^{p-1}/\varepsilon, c^p/\varepsilon)$-list recoverable, for a suitable constant $\alpha_p > 1$ (the details on how to pick the constants will be clarified shortly).

We now give an inductive definition of C_p in terms of C_{p-1} and the above code G_p. To encode $\mathbf{x} \in \mathbb{F}_q^n$ using the code C_p, where $n = 2m^p$, we view $\mathbf{x}$ as a string of length m over $\mathrm{GF}(q^{2m^{p-1}})$, and first encode it using G_p. This gives us a string $\mathbf{x_1}$ of $a_p m$ symbols over $\mathrm{GF}(q^{2m^{p-1}})$. We now view each of these $a_p m$ symbols as a string of length $2m^{p-1}$ over $\mathbb{F}_q$ and independently encode them using C_{p-1}. This completes the inductive specification of the code C_p.

The list recoverability requirement on C_p will let us fix the constants α_j's above. This will in turn fix the rates of the codes (or in other words the constant a_j's). We sketch this next, followed by an analysis of the construction complexity (both probabilistic and deterministic) of the code C_p.

Rate of the Construction. For every $p \geq 1$, and each fixed $\alpha < 1$, for a large enough constant $c = c_{p,\alpha}$, we now wish to pick parameters (specifically α_j's) that allow us to show that the code C_p constructed above is $(\alpha, 1/\varepsilon, c^p/\varepsilon)$-list recoverable in $2^{O(N^{1/p} \log(1/\varepsilon))}$ time where N is the blocklength of C_p. This can be achieved for $p = 1$ by a choice of C_1 with $\alpha_1 = \alpha$ and the rate of the code is an absolute constant (that depends on α). For $p > 1$, let us by induction pick C_{p-1} so that it is $(\alpha/2, 1/\varepsilon, c^{p-1}/\varepsilon)$-list recoverable. We will pick the "outermost" code G_p in the construction of C_p so that it is $(\alpha/2, c^{p-1}/\varepsilon, c^p/\varepsilon)$-list recoverable. By Lemma 9.11 we have such a G_p with rate

$$R(G_p) \geq \frac{1}{6}(\alpha/2 - 1/c) \ . \tag{9.6}$$

Now, applying a standard averaging argument one can combine the facts that C_{p-1} is $(\alpha/2, 1/\varepsilon, c^{p-1}/\varepsilon)$-list recoverable and G_p is $(\alpha/2, c^{p-1}/\varepsilon, c^p/\varepsilon)$-list recoverable to conclude that C_p is $(\alpha, 1/\varepsilon, c^p/\varepsilon)$-list recoverable. It remains to estimate the rate $R(C_p)$ of the code C_p (as a function f of α, p). By the above construction, we have

$$f(\alpha, p) = R(G_p) f(\alpha/2, p-1) \geq \frac{1}{6}\Big(\frac{\alpha}{2} - \frac{1}{c}\Big) f(\alpha/2, p-1) \qquad \text{(using (9.6))} \ .$$

Unwinding the recurrence, for α a fixed constant, like $\alpha = 1/2$ say, we can get C_p that is $(1/2, 1/\varepsilon, c^p/\varepsilon)$-list recoverable with $c \simeq O(2^p)$ and rate $R(C_p) = 2^{-O(p^2)}$. We have thus verified Property (i) for our code construction.

Decoding Complexity. The decoding of the code C_p proceeds inductively from the lowermost levels of the concatenation upwards. This is also best described inductively. For $p = 1$, as mentioned earlier, the decoding of C_1 proceeds by running over all $q^{O(m)} = q^{O(N)}$ codewords. For $p > 1$, given lists of size $1/\varepsilon$ at each position of the code, each of $O(m)$ codes C_{p-1} used to encode the symbols of G_p can be list recovered by induction in $2^{O(m \log(1/\varepsilon))}$ time. This passes a list of c^{p-1}/ε possible symbols for each of the $a_p m$ positions of the code G_p. The code G_p is then list recovered to produce a final set of c^p/ε messages as the answers. Since G_p is picked as guaranteed by Lemma 9.11, the list recovering of G_p can be performed in $(c^{p-1}/\varepsilon)^{O(m)} = 2^{O(m\log(1/\varepsilon))}$ time as well (absorbing factors which depend on c^{p-1} into the big-Oh notation, since we treat c, p as fixed constants). The overall decoding time is the sum of the decoding times for C_{p-1} and G_p, and is thus $2^{O(m\log(1/\varepsilon))}$. Since $n = 2m^p$ is the length of the message and the rate of C_p is $2^{-O(p^2)}$, we have the overall blocklength $N = 2^{O(p^2)}n = O(m^p)$. Therefore, the overall decoding complexity equals $2^{O(N^{1/p}\log(1/\varepsilon))}$, as claimed in Part (ii) of the lemma.

CONSTRUCTION COMPLEXITY. We finally verify the claimed construction complexity bounds for the code C_p. For $p = 1$, we appeal to Lemma 9.8 to conclude that C_1 can be constructed in $O(N^2\varepsilon^{-1}\log(1/\varepsilon))$ probabilistic time, or $2^{O(N\varepsilon^{-1}\log(1/\varepsilon))}$ deterministic time. For $p > 1$, the dominant component is the time to construct the outermost code G_p. Lemma 9.11 implies that G_p can be constructed in E_p can be constructed in $O(m^{2p}\log(1/\varepsilon))$ time probabilistically, and in $2^{O(m^p\varepsilon^{-1}\log(1/\varepsilon))}$ time deterministically. Since $m = O(N^{1/p})$, the construction time is $O(N^2\log(1/\varepsilon))$ probabilistically, and $2^{O(N\varepsilon^{-1}\log(1/\varepsilon))}$ deterministically. The encoding time is again dominated by the time to perform the outermost encoding according to G_p, and is therefore $O(m^{2p}\log^2 q) = O(N^2\log^2(1/\varepsilon))$. This completes the proof of Property (iii) in the statement of the lemma. □

9.5.2 Codes of rate $\Omega(\varepsilon)$ with sub-exponential list decoding for a fraction $(1-\varepsilon)$ of errors

We now use the multi-concatenated codes from the previous section to attain rate $\Omega(\varepsilon)$ for codes list decodable up to a fraction $(1-\varepsilon)$ of errors in sub-exponential time. Note that such a result was also discussed in Section 9.4.4, but we will now improve the decoding time from $2^{O(\sqrt{N})}$ to $2^{O(N^\gamma)}$ for each fixed $\gamma > 0$. Setting $\gamma = 1/2$ in the below theorem gives the result claimed in Section 9.4.4.

Theorem 9.16 *For every constant $\gamma > 0$ the following holds: for all sufficiently small $\varepsilon > 0$, there exists a code family with the following properties:*

(i) (Rate and alphabet size) The code has rate $\Omega(\varepsilon 2^{-O(\gamma^{-2})})$ and is defined over an alphabet of size $2^{O(\varepsilon^{-1}\log(1/\varepsilon))}$.

(ii) (Construction complexity) A description of a code of blocklength N in the family can be constructed in probabilistic $O(N^{2-2\gamma}\log(1/\varepsilon))$ time, or deterministically in time $2^{O(N^{1-\gamma}\varepsilon^{-1}\log(1/\varepsilon))}$. Moreover the code can be encoded in $O(N^{2(1-\gamma)}\log^2 N\log^{O(1)}(1/\varepsilon))$ time.

(iii) (List decodability) The code can be list decoded in $2^{O(N^\gamma\log(1/\varepsilon))}$ time from up to a fraction $(1-\varepsilon)$ of errors.

Proof: We use a construction quite similar to that of Theorem 9.14. Let $p' = \lceil 1/\gamma\rceil$ and $q_0 = O(1/\varepsilon^2)$ be a prime power. At the outermost level, we use a Reed-Solomon code C_{RS} of blocklength n_0 over a field of size $q_0^{n_0^{p'-1}}$ (instead of a field of size n_0 that we used in earlier constructions). Furthermore, the rate of the Reed-Solomon code will now be an absolute constant, say $1/4$ (as opposed to $O(\varepsilon)$ earlier). Each of the n_0 field elements (viewed as a string of length $n_0^{p'-1}$ over $\mathrm{GF}(q_0)$) is encoded using a *multi-concatenated* inner code C'_{in} that encodes $n_0^{p'-1}$ symbols into $2^{O(p^2)}n_0^{p'-1}$ symbols (over $\mathrm{GF}(q_0)$) and which has the properties guaranteed by Lemma 9.15 for $p = p'-1$. Denote by $C_{\mathrm{RS-in}}$ the resulting concatenated

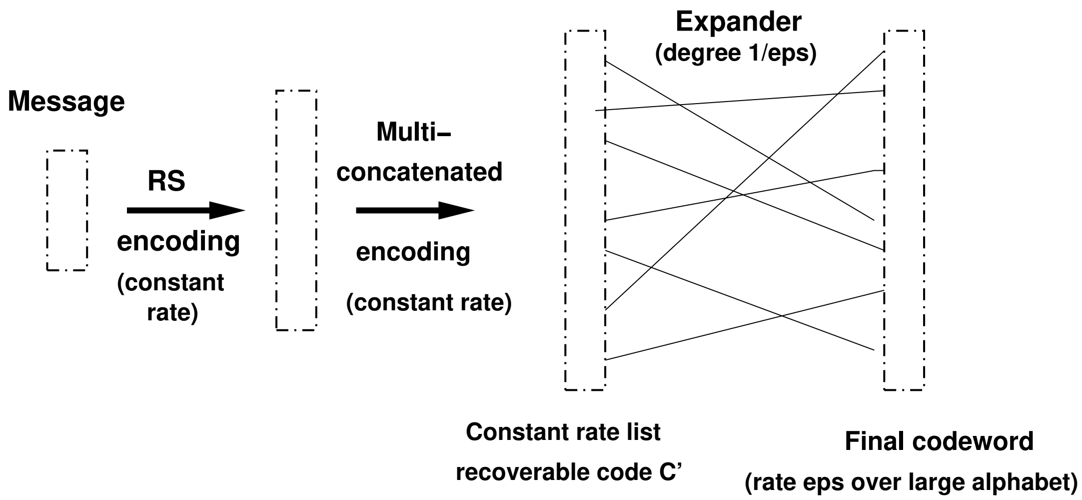

Figure 9-5: Basic structure of code construction that achieves rate $\Omega(\varepsilon)$ and list decoding radius $(1-\varepsilon)$. The list recoverability property of C' enables list decoding of the final code from a fraction $(1-\varepsilon)$ of errors.

code. The rest of the construction (i.e. obtaining the final code C^* from $C_{\text{RS-in}}$ using a expander) is the same as Theorem 9.14, and the claims about the rate and alphabet size follow similarly to Theorem 9.14. See Figure 9-5 for a sketch of the basic components in the construction.

About construction complexity, the significant component is finding the inner code C'_{in}, which can be done in $2^{O(n_0^{p'-1}\varepsilon^{-1}\log(1/\varepsilon))}$ time by Lemma 9.15, or in probabilistic $O((n_0^{p'-1})^2\log(1/\varepsilon))$ time. Since the overall blocklength of the code equals $N = n_0 2^{O(p^2)} n_0^{p'-1} = 2^{O(p^2)} n_0^{p'}$, we have $n_0 = O(N^{1/p'})$ and hence the claimed bounds on the construction time follow.

About list decoding complexity, we claim that $C_{\text{RS-in}}$ is $(1/2, O(1/\varepsilon), L)$-list recoverable in $2^{O(N^{1/p'}\log(1/\varepsilon))}$ time, for $L = 2^{O(N^{1/p'}\log(1/\varepsilon))}$. Now, appealing to Proposition 9.12 implies that our final code C^* will then be $((1-\varepsilon)N, L)$-list decodable in $2^{O(N^{1/p'}\log(1/\varepsilon))}$ time, which is what we would like to show.

To $(1/2, O(1/\varepsilon), L)$-list recover the concatenated code $C_{\text{RS-in}}$, we first use the decoding strategy guaranteed by Lemma 9.15 to $(1/4, O(1/\varepsilon), O(1/\varepsilon))$-list recover each of the n_0 inner codes. This takes a total of $n_0 2^{O((n_0^{p'-1})^{1/p}\log(1/\varepsilon))} = 2^{O(n_0\log(1/\varepsilon))}$ time (since $p = p'-1$), and passes lists of size $O(2^{O(p^2)}/\varepsilon)$ corresponding to each position of the Reed-Solomon code. Since we are thinking of p as a constant and ε sufficiently small, we can assume that lists of size $O(1/\varepsilon)$ are passed for each position of the Reed-Solomon code. For any message $\mathbf{x}$ that is a solution to the list recovering instance, at least a fraction $1/4$ of these lists contain the "correct" symbol of $C_{\text{RS}}(\mathbf{x})$. We now finish the decoding by a brute-force decoding of the outermost Reed-Solomon code as follows. Given lists of size $O(1/\varepsilon)$ for each of the n_0 codeword positions (these lists are the ones obtained after the independent decoding of the n_0 inner codes), for each subset of $n_0/4$ codeword positions and each possible choice of field element from the respective list (this involves considering

$\binom{n_0}{n_0/4} \cdot (O(1/\varepsilon))^{n_0/4} = 2^{O(n_0 \log(1/\varepsilon))}$ possibilities), we do the following. Determine if there is a Reed-Solomon codeword consistent with the $n_0/4$ field elements at the chosen positions (this can be performed using a straightforward polynomial interpolation since the rate of the Reed-Solomon code is $1/4$), and, if so, include that codeword in the list. Recalling that $n_0 = O(N^{1/p'})$, it is clear that the Reed-Solomon decoding can be performed in $2^{O(N^{1/p'} \log(1/\varepsilon))}$ time. Since $1/p' \leq \gamma$, this is consistent with our claimed runtime. □

Improvement to List Size

Note that the size of the list returned in decoding the above codes up to a fraction $(1-\varepsilon)$ of errors is $2^{O(N^\gamma \log(1/\varepsilon))}$. It might be of interest to keep this list size small, ideally a constant, even if the decoding algorithm itself runs in sub-exponential time. This can be achieved by skipping the use of the outermost Reed-Solomon code in the above construction and just using the $(1/2, O(1/\varepsilon), O(1/\varepsilon))$-list recoverable multi-concatenated code C'_{in} in the construction of Proposition 9.12. This will also give a code that is $((1-\varepsilon)N, O(1/\varepsilon))$-list decodable in time $2^{O(N^\gamma \log(1/\varepsilon))}$, at the expense of the construction times being slightly worse than those claimed in Theorem 9.16. Specifically, the probabilistic construction time will now be $O(N^2 \log(1/\varepsilon))$ and the deterministic construction time will be $2^{O(N\varepsilon^{-1} \log(1/\varepsilon))}$.

A version of Theorem 9.16 for list recoverability

We now state a variant of Theorem 9.16 that will be useful in getting binary codes in the next section.

Lemma 9.17 *For every constant $\gamma > 0$ the following holds: for all $\varepsilon > 0$, there exists a code family with the following properties:*

(i) (Rate and alphabet size) The code has rate $\Omega(\varepsilon 2^{-O(\gamma^{-2})})$ and is defined over an alphabet of size $2^{O(\varepsilon^{-1} \log(1/\varepsilon))}$.

(ii) (Construction complexity) A description of a code of blocklength N in the family can be constructed in probabilistic $O(N^{2-2\gamma} \log(1/\varepsilon))$ time, or deterministically in time $2^{O(N^{1-\gamma}\varepsilon^{-3} \log(1/\varepsilon))}$.

(iii) (List decodability) The code can be $(\varepsilon/2, O(1/\varepsilon^2), L)$-list recovered in $2^{O(N^\gamma \log(1/\varepsilon))}$ time (for $L = 2^{O(N^\gamma \log(1/\varepsilon))}$).

Proof (Sketch): The above result really follows using the same proof as that of Theorem 9.16. The point is that we we assume the code $C_{\text{RS-in}}$ to be $(1/2, O(1/\varepsilon^3), L)$-list recoverable (instead of $(1/2, O(1/\varepsilon), L)$-list recoverable). Accordingly we will have to change its parameters and replace each ε by ε^3. But this will still keep its alphabet size $q_0 = 1/\varepsilon^{O(1)}$ and its rate will be $2^{-O(\gamma^{-2})}$ which is a constant independent of ε. We will get our final code C^* from the code $C_{\text{RS-in}}$ by applying Lemma 9.13 (instead of Proposition 9.12), with the choice $\ell = O(1/\varepsilon^2)$. Thus we can get a code C^* of rate $\Omega(\varepsilon)$ over an alphabet of size $q_0^{O(1/\varepsilon)} = 2^{O(\varepsilon^{-1} \log(1/\varepsilon))}$ which is $(\varepsilon/2, O(1/\varepsilon^2), L)$-list recoverable. □

9.5.3 Binary codes of rate $\Omega(\varepsilon^3)$ with sub-exponential list decoding up to a fraction $(1/2-\varepsilon)$ of errors

We now use the code construction from Lemma 9.17 as outer codes in a concatenated scheme with a suitable binary inner code and obtain constructions of good list decodable binary codes. Our result is stated formally below.

Theorem 9.18 *For every constant $\gamma > 0$ the following holds: for all sufficiently small $\varepsilon > 0$, there exists a* binary *code family with the following properties:*

(i) (Rate) It has rate $\Omega(\varepsilon^3 2^{-O(\gamma^{-2})})$.

(ii) (Construction Complexity) A description of a code of blocklength N from the family can be constructed with high probability in randomized $O((N^{2(1-\gamma)} + \varepsilon^{-6})\log(1/\varepsilon))$ time or deterministically in time $2^{O(N^{1-\gamma}\varepsilon^{-3}\log(1/\varepsilon))}$. The code can be encoded in $O(N^{2(1-\gamma)}\log^2 N \log^{O(1)}(1/\varepsilon))$ time.

(iii) (List decodability) A code of blocklength N from the family can be list decoded from up to $(1/2-\varepsilon)N$ errors in $2^{O(N^{\gamma}\log(1/\varepsilon))}$ time.

Proof: The code will be obtained by concatenation of an outer code C_{out} over a large alphabet with a binary code C_{inner}. The code C_{out} will be picked as guaranteed by Lemma 9.17 and will be over an alphabet Σ_{out} of size $2^{O(\varepsilon^{-1}\log(1/\varepsilon))}$. Let m denote the blocklength of C_{out}. The code C_{inner} will be a binary code of rate $\Omega(\varepsilon^2)$, dimension $\lg|\Sigma_{\text{out}}| = O(\varepsilon^{-1}\log(1/\varepsilon))$, blocklength $t = O(\varepsilon^{-3}\log(1/\varepsilon))$ which is $((\frac{1}{2}-\frac{\varepsilon}{2})t, O(1/\varepsilon^2))$-list decodable. Such codes C_{inner} (in fact, linear codes) exist and can be found deterministically in $2^{O(t)}$ time (cf. Section 5.3.2 and Section 8.6.1). Alternatively, one can also pick a random rate $\Omega(\varepsilon^2)$ pseudolinear code by investing $O(\varepsilon^{-6}\log^2(1/\varepsilon))$ randomness. The fact that this such a code will be $((1/2-\varepsilon)t, O(1/\varepsilon^2))$-list decodable with high probability can be seen using Lemma 9.4 with the choice $q = 2$, $p = (1-\varepsilon)/2$ and $L = O(1/\varepsilon^2)$.

Let us call the entire concatenated binary code C_{bin} and let its blocklength be $N = m \cdot t$. A codeword in C_{bin} is comprised of m blocks (of t bits each) corresponding to the m codeword positions of the outer code C_{out}. Note that C_{bin} clearly has the claimed rate since C_{out} has rate $\Omega(\varepsilon \cdot 2^{-O(\gamma^{-2})})$ and C_{inner} has rate $\Omega(\varepsilon^2)$. The construction complexity of C_{bin} is the time required to construct C_{out} plus that required to construct the binary code C_{inner}. Therefore, using Lemma 9.17 and the above discussion concerning the construction of C_{inner}, the claimed bound on the construction complexity of C_{bin} follows. This proves Properties (i) and (ii) claimed in the theorem.

It remains to prove Property (iii) concerning the list decodability of C_{bin}. By the list decodability property of C_{out} guaranteed by Lemma 9.17, we may assume that there is an efficient algorithm A_{out} with runtime exponential in m^{γ} that, given as input lists L_i of size $O(1/\varepsilon^2)$ for $1 \le i \le m$, can find a list of all codewords of $\langle c_1, \ldots, c_m\rangle \in C_{\text{out}}$ such that $c_i \in L_i$ for at least a fraction $\varepsilon/2$ of the i's.

The list decoding algorithm for C_{bin} works as follows. Given a received word $\mathbf{r} \in \{0,1\}^N$, the algorithm finds, for each i, $1 \le i \le m$, a list L_i of all symbols β of Σ_{out} such that $C_{\text{inner}}(\beta)$ differs from the i'th block $\mathbf{r_i}$ of $\mathbf{r}$ in at most $\frac{t(1-\varepsilon)}{2}$ positions. Since C_{inner} is $((1-\varepsilon)t/2, O(1/\varepsilon^2))$-list decodable, each L_i has at most $O(1/\varepsilon^2)$ elements. Now we run the decoding algorithm A_{out} with input these m lists L_i. The runtime of the algorithm is dominated by that of A_{out}, which is $2^{O(m^{\gamma}\log(1/\varepsilon))}$, and is thus within the claimed bound.

To prove correctness of the algorithm, let $\mathbf{c} \in C_{\text{bin}}$ be any codeword which differs from $\mathbf{r}$ in at most $(1/2-\varepsilon)N$ positions (the list decoding algorithm must output every such $\mathbf{c}$). Let β_i, $1 \le i \le m$, be the i'th symbol of the codeword of C_{out} which upon concatenation with C_{inner} gives $\mathbf{c}$. By a simple averaging argument, one can show that for at least an $\varepsilon m/2$ values of i, $1 \le i \le m$, $\beta_i \in L_i$. By its claimed property, the decoding algorithm A_{out} will hence place $\mathbf{c}$ on the list it outputs. This completes the proof of Property (iii) claimed in the theorem as well. $\square$

9.6 Improving the alphabet size: Juxtaposed Codes

One drawback of the result of Theorem 9.14 (as well as that of Theorem 9.16) is that these give codes over an alphabet which is exponentially large in $1/\varepsilon$. In this section, we indicate how one can improve the alphabet size significantly at the expense of a moderate worsening of the rate, by using an entirely different technique (the technique is also somewhat simpler, as it avoids the use of expanders). The basic idea is use to several concatenated codes, each one of which is "good" for some error distribution, and then "juxtapose" symbols from these codes together to obtain a new code over a larger alphabet which has nice list decodability properties. The idea of juxtaposed codes is already used in this chapter in the proof of Lemma 9.15, where we juxtaposed a pseudolinear code with a linear code. But the use of juxtaposition there was for a largely "technical" reason. On the other hand, juxtaposition is fairly natural for the codes we construct in this section. The discussion in Section 9.2.2 already presented a high level discussion of the rationale behind juxtaposed codes; we further elaborate on this aspect below.

9.6.1 Intuition

The basic intuition for considering juxtaposed codes can be understood by considering the following very natural way of constructing a code list decodable up to a fraction $(1-\varepsilon)$ of errors. Namely, concatenate an outer Reed-Solomon code (call its blocklength n_0) with an inner code over an alphabet of size $O(1/\varepsilon^2)$ as guaranteed by Corollary 9.5. Each inner encoding by itself can tolerate a fraction $(1-O(\varepsilon))$ of errors via list decoding with lists of size $O(1/\varepsilon)$. Now consider a received word $\mathbf{r}$ and a codeword $\mathbf{c}$ of the concatenated code which agree on a fraction ε of symbols. If this agreement is evenly distributed among the n_0 blocks that correspond to the various inner encodings, then each of the n_0 inner codes can be decoded (by a simple brute-force search over all inner codewords) and return a list of $O(1/\varepsilon)$ Reed-Solomon symbols that includes the "correct" symbol. If the rate of the Reed-Solomon code is $O(\varepsilon)$, list recovering the Reed-Solomon using these lists is guaranteed to include the codeword $\mathbf{c}$ (cf. Chapter 6). The overall rate of the concatenated code can thus be $\Omega(\varepsilon^2)$, since both the Reed-Solomon and inner codes can have rate $\Omega(\varepsilon)$.

This seems to give us the desired construction with rate $\Omega(\varepsilon^2)$. There is a (big) problem, however. There is no guarantee that errors will be evenly distributed among the n_0 blocks. In fact, on the other extreme, it is possible that $\mathbf{c}$ and $\mathbf{r}$ agree completely on a fraction ε of the blocks, and differ completely on the remaining fraction $(1-\varepsilon)$ of the blocks. To tackle this case, the natural inner decoding to perform is to, for each block, simply return the symbol whose inner encoding is closest to that block of $\mathbf{r}$. Now the "correct" symbol (corresponding to $\mathbf{c}$) will be thus passed to the outer Reed-Solomon decoder for a fraction ε of the positions. To finish the decoding, we would need to be able to list decode the Reed-Solomon code for a $(1-\varepsilon)$ errors, and it is only known how to do so efficiently if the rate is $O(\varepsilon^2)$ (cf. Chapter 6, Theorem 6.13).

Thus the two widely differing (i.e. completely uniform and highly non-uniform) distributions of errors between the various inner codeword blocks require the rate of the Reed-Solomon code to be $O(\varepsilon)$ and $O(\varepsilon^2)$ respectively. Thus one has to conservatively pick the rate of the Reed-Solomon code to be $O(\varepsilon^2)$ to handle the highly non-uniform distribution of errors. The rate of the inner code has to be $O(\varepsilon)$ to handle the uniform distribution of errors. Therefore the overall rate has to be at most $O(\varepsilon^3)$.

A closer inspection of the question reveals that this limitation is due to our using a single outer code and single inner code, which can only be optimized for one error distribution, and suffers for a different error distribution. This suggests the use of several concatenated codes *in parallel*, each with its own outer and inner code rates that are optimized for a certain distribution of errors between the various inner codeword blocks. These concatenated codes can then be "put together" by juxtaposing their symbols together. Now, depending on how uniformly the errors are distributed, a certain concatenated code "kicks in" and enables recovery of the message. The use of multiple concatenated codes reduces the rate compared to the expander based constructions, and also increases the alphabet size compared to a single concatenated code. It turns out, however, that we can still do much better on alphabet size than the bound of $2^{O(\varepsilon^{-1}\log(1/\varepsilon))}$ that was achieved by the construction of Theorem 9.14.

9.6.2 The actual construction

We first discuss the basic code construction scheme, and then formally state the theorems we obtain for appropriate setting of parameters. Let $\varepsilon > 0$ be given; the goal being to construct a code family of good rate (as close to $\Omega(\varepsilon^2)$ as possible) that can be efficiently decoded from up to a fraction $(1-\varepsilon)$ of errors. Let $t \geq 1$ be an integer parameter (t will be the number of codes that will be juxtaposed together).

Let $\delta_0, \delta_1, \ldots, \delta_t$ be a sequence in geometric progression with $\delta_0 = \varepsilon/2$, $\delta_t = 1$, and $\delta_i/\delta_{i-1} = \Delta$ for $1 \leq i \leq t$. Note that these parameters must satisfy $\Delta^t = 2/\varepsilon$.

Fix $c > 1$ and let $q_0 = O(1/\varepsilon^c)$ be a prime power. Let m be a large enough integer. The juxtaposed code construction, say C^*, that we now give, will be parameterized by $(q_0, m, \varepsilon, t, \Delta)$.

For each i, $1 \leq i \leq t$, we will have one q_0-ary concatenated code $\mathbf{C_i}$ with outer code a Reed-Solomon code $C_i^{\rm RS}$ and inner code an appropriate q_0-ary pseudolinear code $C_i^{\rm in}$. The parameters of these codes will be as follows.

THE REED-SOLOMON CODES. The blocklength of each of the Reed-Solomon codes will be the same, $n_0 = q_0^m$. The code $C_i^{\rm RS}$ will be defined over the alphabet $\mathrm{GF}(q^{m_i})$ where $m_i = m\delta_i/\delta_0$. The rate of $C_i^{\rm RS}$ will be $R_i = \Theta(\varepsilon^2/(t^2\delta_i\Delta))$ and its dimension will be $k_i = R_i n_0$ (the reason for this choice of rate will be become clear once we specify the decoding algorithm in the proof of Theorem 9.19 below). Note that each message that is encoded by $C_i^{\rm RS}$ consists of k_i symbols over $\mathrm{GF}(q_0^{m_i})$, or equivalently, $k_i m_i = R_i n_0 m_i = \Theta(\frac{\varepsilon m n_0}{t^2\Delta})$ symbols over $\mathrm{GF}(q_0)$. This quantity is independent of i, and hence the number of q_0-ary symbols in the message of each $C_i^{\rm RS}$ can made equal. This is very useful for juxtaposing the codes together, as it makes sure that the dimension of each of the concatenated codes $\mathbf{C_i}$ will be the same.

THE INNER CODES. The blocklength of each inner code $C_i^{\rm in}$ will be the same, say n_1. Note that this ensures that each one of the concatenated codes $\mathbf{C_i}$ has identical blocklength $N \stackrel{\rm def}{=} n_0 n_1$. The dimension of $C_i^{\rm in}$ will be m_i, so that it can be concatenated with the Reed-Solomon code $C_i^{\rm RS}$ (that was defined over $\mathrm{GF}(q_0^{m_i})$). The code $C_i^{\rm in}$ will have the properties guaranteed by Corollary 9.5 – specifically, it will have rate $r_i = m_i/n_1 = \Omega(\delta_{i-1})$ and will

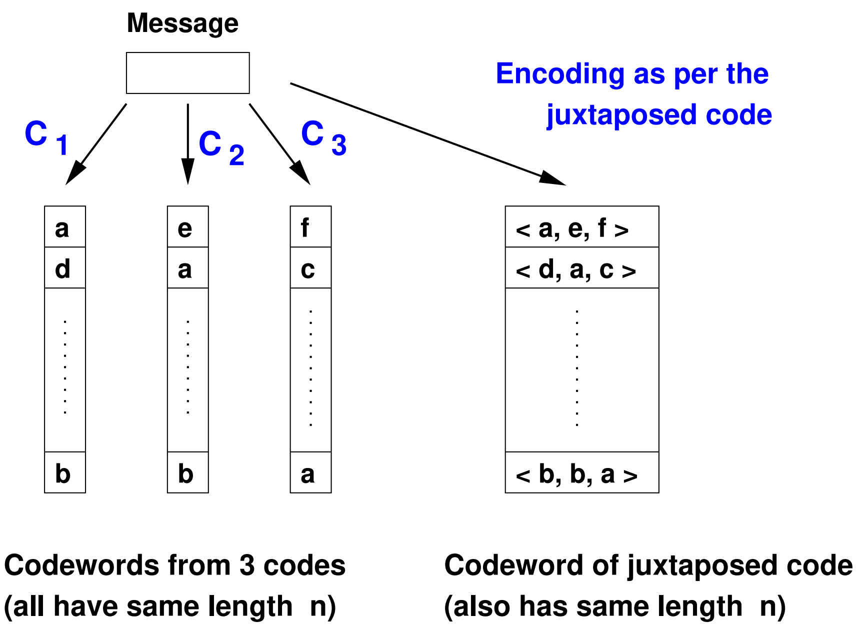

Figure 9-6: Illustration of juxtaposition of three codes C_1, C_2, C_3.

be $((1-\delta_{i-1})n_1, O(1/\delta_{i-1}))$-list decodable.[6] This implies that the blocklength n_1 equals

$$n_1 = O\Big(\frac{m_i}{\delta_{i-1}}\Big) = O\Big(\frac{m\delta_i}{\delta_0\delta_{i-1}}\Big) = O\Big(\frac{m\Delta}{\varepsilon}\Big) \ ,$$

which is independent of i and can thus be made identical for each of the inner codes C_i^{in}.

The construction time of $\mathbf{C_i}$ is dominated by the construction time for the inner code C_i^{in}. Now, using Lemma 9.9, we know that a C_i^{in} with the required properties can be constructed in deterministic $q_0^{O(n_1)} = q_0^{O(\frac{m\Delta}{\varepsilon})}$ time. Alternatively, a construction that works with high probability can be obtained in $O(n_1^2 \log q_0) = O(m^2\Delta^2\varepsilon^{-2}\log(1/\varepsilon))$ time.

THE JUXTAPOSED CODE. The final code, call it C^*, will be the juxtaposition of the codes $\mathbf{C_1}, \mathbf{C_2}, \ldots, \mathbf{C_t}$. Formally, by this we mean that to encode a message according to C^*, we will encode it according to each $\mathbf{C_i}$ to give a codeword, say $\langle c_{i1}, \ldots, c_{iN}\rangle \in \mathrm{GF}(q_0)^N$. The associated codeword of C^* will then be $\langle d_1, \ldots, d_N\rangle \in \mathrm{GF}(q_0^t)^N$ where $d_j = \langle c_{1j}, c_{2j}, \ldots, c_{tj}\rangle$ is interpreted as an element of $\mathrm{GF}(q_0^t)$. Figure 9-6 illustrates the juxtaposition operator applied to three codes.

We now pick parameters δ_i's appropriately in the above scheme and obtain the following theorem. (We use the notation developed above freely in the proof of the theorem.)

Theorem 9.19 *For every $\varepsilon > 0$, every integer $t \ge 1$ and each $b > t$, there exists a code family with the following properties:*

(i) It has rate $\Omega(t^{-3}\varepsilon^{2+2/t})$ and is defined over an alphabet of size $O(1/\varepsilon^b)$.

[6]The result of Corollary 9.5 will give such codes over an alphabet of size $O(1/\delta_{i-1}^a)$ for any $a > 1$, but it can be checked that it will equally work over the larger alphabet $\mathrm{GF}(q_0)$ — essentially the larger alphabet only helps that result.

(ii) *A code of blocklength N in the family can be constructed in $N^{O(1/\varepsilon^{1+1/t})}$ time deterministically, and a construction that has the list decodability property (iii) below with high probability can be obtained in probabilistic $O(\varepsilon^{-(2+2/t)}\log^2 N)$ time.*

(iii) *A code of blocklength N belonging to the family can be list decoded from up to a fraction $(1-\varepsilon)$ of errors in $N^{O(1/\varepsilon)}$ time using lists of size $O(t^2/\varepsilon^{1+1/t})$.*

Proof: Let $\varepsilon > 0$ be given. Pick $q_0 = O(1/\varepsilon^c)$ to be a power of 2 where $c = b/t > 1$. Let us pick the δ_i's in geometric progression with $\delta_0 = \varepsilon/2$, $\delta_t = 1$, and $\delta_i/\delta_{i-1} = \Delta$ for $1 \le i \le t$. Note that this implies $\Delta = (2/\varepsilon)^{1/t}$.

For every large enough integer m, we now apply the construction C^* discussed above with parameters $(q_0, m, \varepsilon, t, \Delta)$.

The code C^* is then clearly defined over an alphabet of size $q_0^t = O((1/\varepsilon)^{ct}) = O(1/\varepsilon^b)$ Recall that C^* is the juxtaposition of t codes $\mathbf{C_i}$, $1 \le i \le t$, each of which is obtained by the concatenation of a rate R_i Reed-Solomon code C_i^{RS} with a rate r_i inner code C_i^{in}, where $R_i = \Theta(\frac{\varepsilon^2}{t^2\delta_i\Delta})$ and $r_i = \Theta(\delta_{i-1})$. Therefore the rate of each $\mathbf{C_i}$ equals

$$R_i r_i = \Omega\Big(\frac{\varepsilon^2}{t^2\delta_i\Delta}\cdot\delta_{i-1}\Big) = \Omega\Big(\frac{\varepsilon^2}{t^2\Delta^2}\Big)\ . \tag{9.7}$$

Let K, N be the common dimension and blocklength respectively of the $\mathbf{C_i}$'s. The rate of the juxtaposed code C^* is $1/t$ times the rate of each $\mathbf{C_i}$ because of the juxtaposition operator and hence

$$R(C^*) = \Omega\Big(\frac{\varepsilon^2}{t^3\Delta^2}\Big) = \Omega(t^{-3}\varepsilon^{2+2/t})\ ,$$

as claimed in Part (i) of the theorem.

The dominant component in the construction of C^* is once again the construction of the inner codes C_i^{in} used in the concatenated codes $\mathbf{C_i}$. By the argument from the discussion preceding this theorem, we have that each C_i^{in} can be constructed in $q_0^{O(m\Delta/\varepsilon)}$ time deterministically, and $O(m^2\Delta^2\varepsilon^{-2}\log(1/\varepsilon))$ time probabilistically. Since the overall blocklength $N = n_0 n_1 = q_0^m n_1$, we have $m \le \log N/\log q_0$. Therefore the constructions times are $N^{O(\Delta/\varepsilon)} = N^{O(1/\varepsilon^{1+1/t})}$ for a deterministic construction, and $O(\Delta^2\varepsilon^{-2}\log^2 N) = O(\varepsilon^{-(2+2/t)}\log^2 N)$ for a probabilistic construction (that works with high probability). This proves Property (ii) claimed in the theorem.

It remains to prove the list decodability property of C^*. Specifically, we wish to prove that given a received word $\mathbf{r} \in \mathrm{GF}(q_0^t)^N$, we can output a list of all codewords of C^* that differ from $\mathbf{r}$ in at most $(1-\varepsilon)N$ positions, in $N^{O(1/\varepsilon)}$ time. Indeed let $\mathbf{c} = C^*(\mathbf{x})$ be a codeword of C^* that differs from $\mathbf{r}$ in at most a fraction $(1-\varepsilon)$ of places. Note that both $\mathbf{r}$ and $\mathbf{c}$ can be broken up into n_0 blocks of n_1 symbols each, corresponding to the n_0 inner encodings at the n_0 positions of the outer Reed-Solomon codes. (Here we are using the fact that all the Reed-Solomon codes C_i^{RS} and the inner codes C_i^{in} have the same blocklength, namely n_0 and n_1, respectively.)

Now comes the crucial part. Since the overall agreement between $\mathbf{c}$ and $\mathbf{r}$ is at least a fraction ε of symbols, a standard averaging argument implies that there exists a set B consisting of at least $\varepsilon n_0/2$ inner blocks, such that $\mathbf{c}$ and $\mathbf{r}$ agree on more than a fraction $\varepsilon/2 = \delta_0$ of symbols within each block in B. Now imagine partitioning the blocks in B into t parts P_i, $1 \le i \le t$, in the following way. The part P_i consists of all blocks in B for which the fractional agreement between the portions of $\mathbf{c}$ and $\mathbf{r}$ corresponding to that block lies in the interval $(\delta_{i-1}, \delta_i]$. One of these parts must have at least $|B|/t$ blocks. Let this part be P_{i^*}. Hence we conclude that there exists *some* i^*, $1 \le i^* \le t$, such that for at least a fraction $\frac{\varepsilon}{2t\delta_{i^*}}$ of the n_0 blocks, $\mathbf{c}$ and $\mathbf{r}$ agree on at least a fraction δ_{i^*-1} of positions within that block.

Now consider decoding the n_0 inner codes $C^{\text{in}}_{i^*}$ corresponding to this i^* up to a radius of $(1-\delta_{i^*-1})$ errors (here we focus attention on and use only the i^*'th symbol from each of the N "juxtaposed" symbols from the received word $\mathbf{r}$). This can be accomplished by brute-force in a total of $n_0 q_0^{m\delta_{i^*}/\delta_0} = q_0^{O(m/\varepsilon)} = n_0^{O(1/\varepsilon)}$ time. By the property of $C^{\text{in}}_{i^*}$, this decoding only outputs a list $L_j^{(i^*)}$ of $O(1/\delta_{i^*-1})$ codewords (or in other words Reed-Solomon symbols for the code $C^{\text{RS}}_{i^*}$) for each of the blocks j, $1 \le j \le n_0$. By our choice of i^*, at least $\frac{\varepsilon n_0}{2t\delta_{i^*}}$ of these lists have the "correct" symbol of $C^{\text{RS}}_{i^*}(\mathbf{x})$.

To finish the decoding, it suffices to be able to list decode $C^{\text{RS}}_{i^*}$ with these lists $L_j^{(i^*)}$, $1 \le j \le n_0$, as input, and find *all* messages $\mathbf{x}$ such that $L_j^{(i^*)}$ contains the j'th symbol of $C^{\text{RS}}_{i^*}(\mathbf{x})$ for at least $\frac{\varepsilon n_0}{2t\delta_{i^*}}$ values of j. We can now apply the list recovering algorithm for Reed-Solomon codes from Chapter 6 (specifically Theorem 6.17) to accomplish this decoding task in near-quadratic time. Specifically, this follows by applying Theorem 6.17 with the choice $n = n_0$, $k = k_{i^*} - 1 = O(n_0 \frac{\varepsilon^2}{t^2 \Delta \delta_{i^*}}) = O(n_0 \frac{\varepsilon^2 \delta_{i^*-1}}{t^2 \delta_{i^*}^2})$, $\ell = O(1/\delta_{i^*-1})$ and $\alpha = \frac{\varepsilon}{2t\delta_{i^*}}$. It can be verified that the condition $\alpha > \sqrt{2k\ell/n}$ can be satisfied with these setting of parameters. Moreover, by Theorem 6.17, the number of codewords output by the decoding algorithm will be $O(\sqrt{n\ell/k})$, which is $O(t\Delta/\varepsilon)$ for our choice of parameters.

Of course, the algorithm cannot know the value of i^* in the above description, but running the above decoding procedure for each $\mathbf{C_i}$, $1 \le i \le t$, will output a list of size at most $O(t^2\Delta/\varepsilon) = O(t^2 \varepsilon^{-(1+1/t)})$ that includes all codewords that differ from the received word $\mathbf{r}$ in at most a fraction $(1-\varepsilon)$ of the positions. The decoding time is dominated by the time to decode the inner codes, which, as discussed earlier, takes $n_0^{O(1/\varepsilon)} = N^{O(1/\varepsilon)}$ time. This completes the proof of Property (iii) of the theorem as well. $\square$

Comparison with Algebraic-geometric codes: Note that for $t \le 3$, the result of Theorem 9.19 is incomparable to AG-codes, since it gets a better alphabet size than AG-codes (which work over alphabet size of $O(1/\varepsilon^4)$), but the rate is worse than ε^2. Thus the above codes give some new, interesting trade-offs for codes that can be list decoded in polynomial time from a fraction $(1-\varepsilon)$ of errors.

By picking a fine "bucketing" with $\Delta = 2$ and $t = \lceil \lg(2/\varepsilon) \rceil$ in the above theorem, we can achieve a rate very close to ε^2 though the alphabet size becomes quasi-polynomial in $1/\varepsilon$. This gives us the following result.

Corollary 9.20 *For every $\varepsilon > 0$, there is a code family with the following properties:*

(i) (Rate and alphabet size) It has rate $\Omega(\varepsilon^2 \log^{-3}(1/\varepsilon))$ and is defined over an alphabet of size $2^{O(\log^2(1/\varepsilon))}$.

(ii) (Construction complexity) A code of blocklength N in the family can be constructed in $N^{O(1/\varepsilon)}$ time deterministically, and a construction that has the list decodability property (iii) below with high probability can be obtained in probabilistic $O(\log^2 N/\varepsilon^2)$ time.

(iii) (List decodability) A code of blocklength N in the family can be list decoded from up to a fraction $(1-\varepsilon)$ of errors in $N^{O(1/\varepsilon)}$ time using lists of size $O(\varepsilon^{-1}\log^2(1/\varepsilon))$.

9.7 Notes

The concept of good list recoverable codes, which was crucial to most of our results in this chapter, also appears in the work on extractor codes by Ta-Shma and Zuckerman [TZ01]. The terminology "list recoverable codes" itself was introduced for the first time by the author and Indyk in [GI01b]. Ta-Shma and Zuckerman also analyze the list recoverability of random codes. However, their results are for general random codes and their proof makes use of the complete independence of all the codewords. The result of Lemma 9.2, which appears in [GI01b], works for random pseudolinear codes and also gives bounds for list recovering with constant-sized lists. The result from [TZ01], as stated there, works for list size that depends on the blocklength of the code, since their target is a more general decoding situation when the input lists at each position could be of widely varying and potentially very large sizes. We, on the other hand, place a uniform upper bound on the size of each input list, and moreover are mainly interested in situations where this upper bound is a small fraction of the alphabet size of the code.

In recent years, there have been several papers which construct codes using expanders. These use expanders in one of two ways: either to construct the parity check matrix [SS96, Spi96, Z01́] or the generator matrix [ABN+92]. Our codes constructions follow the spirit of the second approach, in the sense that we also use expander-like graphs to distribute the symbols of the message. However, our constructions are more involved than the construction of [ABN+92], since we want to make the codes efficiently decodable. In particular there is a lot more algorithmic focus in our work than in [ABN+92].

There has also been work on sub-exponential time *unique* decoding algorithms. In particular, the algorithm of [ZP82] can unique decode certain large distance binary codes in $2^{O(\sqrt{n})}$ time. In fact it was this algorithm that motivated our discussion in Section 9.4.4. The quest for an improved decoding time led us to the constructions using multi-concatenated codes that were discussed in Section 9.5. The use of a sequence of inner codes in order to decrease the decoding time by paying only a constant factor in the rate at each level appears to be novel to the constructions in Section 9.5.

Except for the results of Section 9.6, the rest of the material discussed in this chapter appears in [GI01b]. The results of Section 9.6 will appear in an upcoming paper [GI01a].

Chapter 10

List Decoding from Erasures

We know accurately only when we know little,
with knowledge doubt increases.

Johann W. von Goethe

10.1 Introduction

The last two chapters presented a thorough investigation of the question of constructions of good codes, i.e. codes of high rate, which are list decodable from a very large, and essentially the "maximum" possible, fraction of errors. Utilizing the Reed-Solomon decoding algorithms from Chapter 6 at the core, we presented several novel constructions of efficiently constructible, encodable, and decodable codes that approach the (best possible) performance of random codes (for which we do not know of any efficient construction or decoding procedures). The focus was on the noise model where a certain adversarially chosen fraction of the symbols are in error (the actual errors can also be adversarially picked).

In this chapter, we consider the noise model of *erasures*. Under this model the symbols at an adversarially chosen set of positions are simply erased and the rest of the symbols are transmitted with no error. The receiver is assumed to know the positions where erasures have occurred. This is in fact a simpler situation to deal with, since any symbol received unerased is guaranteed to be correct, and reconstructing the codeword is more of an "interpolation-type" problem than an error-recovery problem. We already dealt with erasures in Chapter 6 where we discussed a decoding algorithm for Reed-Solomon codes in the presence of both errors and erasures. Also, under soft decoding, which was also discussed in Chapter 6, an erasure can be modeled by the weight (confidence rating) for the erased symbol being set to 0. Erasures are a popularly used method to model packet losses in the Internet, and indeed good erasure codes are useful tools to deal with packet losses in communication over the Internet.

Having already dealt with the more challenging model of errors, the reader might wonder why we are now moving to a "simpler" model. The reasons are two-fold. First, the fact that erasures are easier to deal with implies that it becomes possible to achieve better parameters (eg. rate) compared to the errors case, and approaching the optimal performance becomes a challenging question that is interesting by itself and is not subsumed by any of the results for the errors case. Second, many of the techniques developed in the previous two chapters apply to this chapter as well, and having already developed them for the errors case, the application to erasures becomes a lot easier to present now.

Following the spirit of the last two chapters, we will be interested in codes with non-trivial list decoding performance for erasures — specifically, codes of "large" rate that are list decodable using "small" lists from a "large" number of erasures. We consider combinatorial questions concerning list decoding from erasures and establish lower and upper bounds on the rate of a code that can be list decoded with list size L when up to a fraction p of its symbols are adversarially erased. Our results show that in the limit of large L, the rate of such a code approaches the capacity $(1-p)$ of the erasure channel. This is in the spirit of the results in Chapter 5. We then present results on efficiently constructible codes which approach the performance indicated possible by the combinatorial results. This is in the spirit of Chapters 8 and 9 where similar results were obtained for the errors case.[1]

One of the results of this chapter shows a *provable separation* between the *asymptotic* performance of linear and non-linear codes for erasure list decodability for certain settings of parameters. This result is quite surprising, at least to us, since such a situation is quite rare in coding theory.

10.2 Overview

We focus on binary codes for most of the chapter, except notably Section 10.7 where the larger alphabet size is critically used. The emphasis on binary codes is only to keep the presentation simple and all claims go through for codes over $\mathbb{F}_q$ for any fixed q.

We first present the basic definitions relating to list decodability from erasures in Section 10.3. The relation between erasure list-decodability and distance is studied in Section 10.4. In Section 10.5, we study the trade-off between erasure list decodability and the rate of a code, and obtain upper and lower bounds on the best possible rate of a code with certain erasure list decodability. We then move on to constructive results in Section 10.6, and present concatenated codes which get reasonably close to the combinatorial bounds. Finally, in Section 10.7 we present constructions of juxtaposed codes which almost achieve the best possible rate for a given erasure list-decodability, albeit over much larger alphabets than binary.

10.3 Definitions

We now present the basic definitions relating to list decoding from erasures. For $y \in [q]^n$ and $T \subseteq \{1, 2, \dots, n\}$, define $[y]_T \in [q]^{|T|}$ to be the projection of y onto the coordinates in T.

Definition 10.1 ((s, L)-erasure list-decodability) *A q-ary code C of blocklength n is said to be (s, L)*-erasure list-decodable *if for every $\mathbf{r} \in [q]^{n-s}$ and every set $T \subseteq \{1, 2, \dots, n\}$ of size $(n-s)$, we have $|\{\mathbf{c} \in C : [\mathbf{c}]_T = \mathbf{r}\}| \le L$. In other words, given any received word with at most s erasures, the number of codewords consistent with the received word is at most L.*

[1] The choice to discuss the combinatorial results relating to list decoding from erasures in this chapter as opposed to in Chapter 5 or in a separate chapter in Part I of the thesis was deliberate. The combinatorial results of this chapter have a "local" presence and are only used for the constructions in this chapter. Hence we felt there was no need to burden the reader with this material earlier on in the thesis. As a side benefit, this makes the current chapter quite self-contained and cohesive.

Note that a code of minimum distance d is $(d-1,1)$-erasure list-decodable, but is *not* $(d,1)$-erasure list-decodable.

Definition 10.2 (Erasure List Decoding Radius) *For an integer $L \geq 1$ and a code C, the* list-of-L erasure decoding radius *of C, denoted $\mathsf{ErasureRad}_L(C)$, is defined to be the* maximum *value of s for which C is (s,L)-erasure list-decodable. We also define the normalized list-of-L erasure decoding radius, denoted $\mathsf{ErasureLDR}_L(C)$, as*

$$\mathsf{ErasureLDR}_L(C) = \frac{\mathsf{ErasureRad}(C,L)}{n} ,$$

where n is the blocklength of C.

As before we would like to extend this definition for families of codes, since our aim is to study the asymptotic performance of codes.

Definition 10.3 (Erasure LDR for Code Families) *For an infinite family $\mathcal{C} = \{C_i\}_{i\geq 1}$ of codes and an integer $L \geq 1$, the* list-of-L erasure decoding radius *of $\mathcal{C}$, denoted $\mathsf{ErasureLDR}_L(\mathcal{C})$, is defined to be*

$$\mathsf{ErasureLDR}_L(\mathcal{C}) \stackrel{\text{def}}{=} \liminf_i \{\mathsf{ErasureLDR}_L(C_i)\} \ . \tag{10.1}$$

One can also allow a list size that is a growing function of the blocklength of the codes in the above definition (as we did in Definition 2.4 for the case of list decoding from errors). But all our results in this chapter will apply with a fixed list size independent of the blocklength. Hence, to keep things simple, we stick to a constant list size in the above definition.

We now define the function which measures the trade-off achievable between rate and erasure list decoding radius.

Definition 10.4 *For an integer L and $0 \leq p \leq 1$, the maximum rate of a q-ary code family with list-of-L erasure decoding radius at least p, denoted $\tilde{R}_{L,q}(p)$, is defined as*

$$\tilde{R}_{L,q}(p) \stackrel{\text{def}}{=} \sup_{\mathcal{C}:\ \mathsf{ErasureLDR}_L(\mathcal{C}) \geq p} R(\mathcal{C}) \ . \tag{10.2}$$

where the supremum is taken over all q-ary code families $\mathcal{C}$ with $\mathsf{ErasureLDR}_L(\mathcal{C}) \geq p$.
If the supremum is taken over all linear *q-ary code families with $\mathsf{ErasureLDR}_L(\mathcal{C}) \geq p$, then we denote the above rate function by $\tilde{R}^{\text{lin}}_{L,q}(p)$.*
For the binary case $q = 2$, we will denote $\tilde{R}_{L,2}(p)$ and $\tilde{R}^{\text{lin}}_{L,2}(p)$ as simply $\tilde{R}_L(p)$ and $\tilde{R}^{\text{lin}}_L(p)$, respectively.

Note: Recall that we used $R_L(p)$ for the similar quantity for the case of errors. To avoid conflict with that notation, we have now used $\tilde{R}_L(p)$ to represent the corresponding function for erasures.

10.3.1 Comment on combinatorial vs. algorithmic erasure list-decodability

For linear codes, list decoding from erasures is algorithmically easy as it just amounts to finding the space of solutions to a linear system. Thus, if a linear code is (s, L)-erasure list-decodable for some small L, then a list of size at most L can also be *efficiently output* given any received word with at most s erasures.[2] Thus for linear codes, it suffices to find codes with good combinatorial erasure list-decodability, and this automatically implies a polynomial time list decoding algorithm from a large number of erasures. However, it might still be possible to give a faster algorithm than the one that solves a linear system and outputs all its solutions. Such a reduction from algorithmic to combinatorial erasure list-decodability does not hold for general, non-linear codes. Therefore, in Section 10.7 where we give non-linear codes, there is a need to explicitly argue that a fast erasure recovery algorithm exists, in addition to the fact that the list size will be small for the concerned number of erasures.

10.4 Erasure list-decodability and minimum distance

We now study the erasure list-decodability of a code *purely* as a function of its minimum distance. This can be viewed as the analog of Chapters 3 and 4 for the case of erasures. The situation for erasures is generally much easier to analyze, though.

The two results below together show that, purely as a function of the relative distance, the best bound on the erasure list decoding radius of a q-ary code of relative distance δ is $q\delta/(q-1)$. This is the analog of the "Johnson radius" for the case of erasures.

Proposition 10.1 *Let C be a q-ary code of blocklength n and relative distance δ. Then, for any $\varepsilon > 0$, C is $((\frac{q}{q-1} - \varepsilon)\delta n, \frac{q}{(q-1)\varepsilon})$-erasure list-decodable.*

Proof: Define $s = (\frac{q}{q-1} - \varepsilon)\delta n$, $t = n - s$, and let $\mathbf{r} \in [q]^t$ be a received word with s erasures. Without loss of generality, assume that the first s symbols of $\mathbf{r}$ have been erased. Let c_i, $1 \le i \le M$, be all the codewords of C that agree with $\mathbf{r}$ in the last $(n-s)$ positions. Hence they all agree with each other in the last t positions. For each i, $1 \le i \le M$, define $\tilde{c}_i$ to be the truncation of c_i to the first s positions. We have $\Delta(\tilde{c}_i, \tilde{c}_j) \ge \delta n = (\frac{q}{q-1} - \varepsilon)^{-1} s$.

Thus we have M strings of length s with fractional distance between any pair strictly larger than $(1 - 1/q)$. It is a folklore fact there can be at most a constant (independent of s) number of such strings. One way to prove this is as follows. As in the proof of Proposition 8.1 from Chapter 8, we can associate sq-dimensional real unit vectors v_i with each $\tilde{c}_i$ such that

$$\langle v_i, v_j \rangle = \Big(1 - \frac{q}{q-1} \frac{\Delta(\tilde{c}_i, \tilde{c}_j)}{s}\Big) .$$

Since $\Delta(\tilde{c}_i, \tilde{c}_j) \ge (\frac{q}{q-1} - \varepsilon)^{-1} s$ for $i \neq j$, we have

$$\langle v_i, v_j \rangle \le -\varepsilon / (\frac{q}{q-1} - \varepsilon) .$$

[2]This is not true for the case when there are errors, where algorithmic list decodability is potentially a lot more difficult to achieve than combinatorial list decodability. In fact, though we presented several list decoding algorithms in this thesis that decode up to the combinatorial (Johnson) bounds, there remain constructions of codes with good combinatorial list decodability, but for which efficient list decoding algorithms are not known.

By Lemma 3.5, the number of such vectors is at most

$$1 + \frac{q/(q-1) - \varepsilon}{\varepsilon} = \frac{q}{(q-1)\varepsilon} \ .$$

Hence $M \le \frac{q}{(q-1)\varepsilon}$, and therefore C is $(s, \frac{q}{(q-1)\varepsilon})$-erasure list-decodable. □

Proposition 10.2 *For every q, and every δ, $0 < \delta < (1-1/q)$, and all small enough $\varepsilon > 0$, the following holds. For all large enough n, there exists a q-ary code C of relative distance at least δ which is not $((\frac{q}{q-1} + \varepsilon)\delta n, 2^{\Omega(\varepsilon^2 \delta n)})$-erasure list-decodable.*

Proof: Define $n' = (\frac{q}{q-1} + \varepsilon)\delta n$. Pick a q-ary code C' of blocklength n' that meets the Gilbert-Varshamov bound and has minimum distance at least $\delta n = (\frac{q}{q-1} + \varepsilon)^{-1} n'$. The relative distance δ' of this code is at most $(1 - 1/q - O(\varepsilon))$, and therefore the code has rate $r \ge 1 - H_q(\delta') = \Omega(\varepsilon^2)$. Now form a code C of blocklength n from C' by just padding each codeword with $(n - n')$ zeroes. The minimum distance of C is still at least δn, and hence its relative distance is at least δ. Moreover, consider the received word $\mathbf{r}$ for C whose first n' positions are erased and the last $(n - n')$ positions contain zeroes. Clearly all codewords in C agree with $\mathbf{r}$ in the unerased positions. Thus, C is not $(n', |C| - 1)$-erasure list-decodable. Recalling that $n' = (\frac{q}{q-1} + \varepsilon)\delta n$ and $|C| = |C'| \ge 2^{rn'} = 2^{\Omega(\varepsilon^2 \delta n)}$, we get the claimed result. □

The above two results indicate that in order to construct, say binary, codes which are erasure list-decodable up to a fraction $(1 - \varepsilon)$ of erasures, it suffices to construct codes which have relative distance $(1/2 - O(\varepsilon))$. Moreover, if one only uses the distance to bound the erasure list decoding radius, then a relative distance of $(1/2 - O(\varepsilon))$ is also necessary. Since there is an upper bound of $O(\varepsilon^2 \log(1/\varepsilon))$ on the rate of such large distance binary codes [MRRW77], this indicates that the best rate for codes erasure list-decodable from a fraction $(1 - \varepsilon)$ of erasures that one can obtain by this method is also $O(\varepsilon^2 \log(1/\varepsilon))$. However, it turns out that a much better rate of $\Omega(\varepsilon)$ is achievable for such codes by directly studying the trade-off between erasure list-decodability and rate. The detailed investigation of such a trade-off is the subject of the next section.

10.5 Combinatorial Bounds for erasure list-decodability

10.5.1 Discussion

We now proceed to establish lower and upper bounds on this function $\tilde{R}_L(p)$. Since the list-of-1 ELDR of a code family equals its relative distance δ, $\tilde{R}_1(p) = R(\delta)$ is the central function in coding theory that studies the trade-off between the rate and relative distance of a code. One of the consequences of results of this chapter (specifically Theorems 10.3 and 10.8) is that in the limit of large L, the function $\tilde{R}_L(p)$ (as well as $\tilde{R}_L^{\text{lin}}(p)$) tends to $1 - p$, thus matching the singleton bound. This result has the following nice interpretation. It is a classical result in coding theory that the capacity of the erasure channel where each codeword symbol is randomly and independently erased with probability p, equals $(1 - p)$ [Eli55]. Thus our results show that using list decoding with large enough (but constant-sized) lists, one can approach capacity even if the symbols that are erased are *adversarially* (and not randomly) chosen.

Our upper bound on $\tilde{R}_L(p)$ also shows that $\tilde{R}_L(p) < 1 - p$ for every p and every fixed list size L (we stress that this result holds even for general, non-linear codes). Thus one *needs* unbounded list size in order to approach the capacity of the adversarial erasure channel using list decoding. We point out that similar statements also held for the errors case — the results of Chapter 5 imply that, in the limit of large L, there exists binary linear codes with rate approaching $(1 - H(p))$ and list decodable using lists of size L up to a fraction p of errors, and also that one requires *unbounded* L in order to get arbitrarily close to the "capacity" $1 - H(p)$.

10.5.2 Lower bound on $\tilde{R}_L(p)$

Theorem 10.3 *For every integer $L \geq 1$ and every p, $0 \leq p \leq 1$, we have*

$$\tilde{R}_L(p) \geq \frac{L}{L+1}(1-p) - \frac{H(p)}{L+1} \ . \tag{10.3}$$

Proof: The proof follows by a straightforward application of the probabilistic method. Pick a random binary code C of blocklength n and with 2^{rn} codewords, where the rate r will be specified shortly. The number of received words with $s = pn$ erasures is exactly $\binom{n}{pn}2^{(1-p)n}$, and thus at most $2^{(H(p)+1-p)n}$. For each such received word, the probability that there exist *some* $L+1$ codewords all of which agree with it in every unerased position is at most

$$\binom{2^{rn}}{L+1}(2^{-(1-p)n})^{L+1} \ .$$

The probability that C is not (pn, L)-erasure list-decodable is thus at most

$$2^{(H(p)+1-p)n} \cdot 2^{rn(L+1)}2^{-(1-p)n(L+1)}$$

which is $o(1)$ for

$$r = \frac{L}{L+1}(1-p) - \frac{H(p)}{L+1} - o(1) \ .$$

Therefore, the lower bound on $\tilde{R}_L(p)$ claimed in (10.3) holds. □

Erasure list-decodability of pseudolinear codes

The above analyzes the performance of general, random codes. It is desirable to achieve similar lower bounds using much less randomness, say by using random pseudolinear or random linear codes. We defer the latter to the next section and now state a result for random pseudolinear codes. The proof follows along the same lines as the above proof – we use the fact any L non-zero codewords of a random $(L, 2)$-pseudolinear code are mutually independent, and hence the probability that they all agree with some received word with s erasures, is easy to compute. Thus, one can conclude that or every p, $0 < p < 1$, and every integer $L \geq 2$, there exists an infinite family $\mathcal{C}_L$ of binary $(L, 2)$-pseudolinear codes of rate r given by

$$r = (1-p)\frac{L-1}{L} - \frac{H(p)}{L} \ ,$$

with the property that the list-of-L erasure decoding radius of $\mathcal{C}_L$ is at least p. Applying this to the case $p = 1-\sigma$ and using the upper bound $H(p) = H(1-\sigma) \leq \sigma \lg(e/\sigma)$ gives the result below. The claimed construction times follows since a random $(L, 2)$-pseudolinear code of dimension k and blocklength n can be picked using a random $n \times O(kL)$ Boolean matrix, and the construction can be derandomized (by using techniques similar to those from Section 9.3.3, but now applied to the erasure setting) in $2^{O(kL)}$ time. We omit the by now standard details.

Lemma 10.4 *For every σ, $0 < \sigma < 1/2$, there exist $L = O(\log(1/\sigma))$ and a family of $(L, 2)$-pseudolinear codes with the following properties:*

(i) It has rate $\Theta(\sigma)$.

(ii) A code of blocklength n in the family is $((1-\sigma)n, L)$-erasure list-decodable.

(iii) A code of blocklength n in the family can be constructed in $O(n^2\sigma\log(1/\sigma))$ time probabilistically and in $2^{O(\sigma\log(1/\sigma)n)}$ time deterministically.

We will use the codes guaranteed by the above lemma in Section 10.7 as inner codes in a suitable concatenation scheme. We next turn to linear codes.

10.5.3 Lower bound on $\tilde{R}_L^{\text{lin}}(p)$

A lower bound using Theorem 10.3

For every integer $L \geq 1$ and every p, $0 \leq p \leq 1$, it is easy to deduce the lower bound

$$\tilde{R}_L^{\text{lin}}(p) \geq \frac{J-1}{J}(1-p) - \frac{H(p)}{J}\ , \tag{10.4}$$

where $J = \lceil \lg(L+1) \rceil$, using a proof similar to that the result of Theorem 10.3. Indeed, we can pick a random linear code and analyze the probability that there are $(L+1)$ codewords all agreeing with some received word that has a fraction p of symbols erased. Now, any set of $(L+1)$ codewords must have at least $J = \lceil \lg(L+1) \rceil$ codewords that correspond to encodings of linearly independent messages. Therefore we can simply consider each set of J linearly independent messages and analyze the probability that they are mapped to a set of J codewords all of which agree with a received word that has a fraction p of erasures. Owing to the mutual independence of these J codewords, an analysis similar to Theorem 10.3 can be carried out with every occurrence of $(L+1)$ being replaced by J. This argument has already been used in a couple of places in the thesis, namely in the proofs of Theorem 5.3 and Lemma 9.6.

While this lower bound will suffice for our applications to concatenated schemes (in Section 10.6), we now present a more direct analysis that gives a better lower bound, which shaves off the $(1 - 1/J)$ factor multiplying $(1-p)$ in the bound (10.4). The difference between the new bound and (10.4) above becomes negligible for large L.

A better lower bound on $\tilde{R}_L^{\text{lin}}(p)$

The following folklore combinatorial lemma gives a useful linear-algebraic characterization of when a linear code is (s, L)-erasure list-decodable.

Lemma 10.5 *An $[n, k]_2$ linear code C is (s, L)-erasure list-decodable if and only if its $n \times k$ generator matrix G has the property that every $(n-s) \times k$ submatrix of G has rank at least $(k - \lfloor \lg L \rfloor)$.*

Proof: Let $T \subseteq \{1, 2, \ldots, n\}$ with $|T| = n - s$ and $r \in \{0,1\}^{n-s}$, the number of codewords $c \in C$ with $[c]_T = r$ is precisely the number of solutions $x \in \{0,1\}^k$ to the system $G_T x = r$ where G_T is the submatrix of G consisting of all rows indexed by elements in T. By standard linear algebra, the number of solutions x to the linear system $G_T x = \mathbf{0}$ is precisely 2^ℓ where $\ell = k - \text{rank}(G_T)$, and for any $r \in \{0,1\}^{n-s}$, the number of solutions x to $G_T x = r$ is at most 2^ℓ (in fact, it is always either 0 or 2^ℓ). Hence C is (s, L)-erasure list-decodable if and only if for every $T \subseteq \{1, \ldots, n\}$ with $|T| = n - s$, G_T has rank at least $k - \lfloor \lg L \rfloor$. $\square$

Theorem 10.6 *For every integer $L \geq 1$ and every p, $0 \leq p \leq 1$, we have*

$$\tilde{R}_L^{\rm lin}(p) \geq 1 - p - \frac{H(p)}{\lceil \lg(L+1) \rceil} \ . \tag{10.5}$$

The above theorem follows from Part (i) of the stronger claim of the lemma below. The second part of the lemma will be used in the application to concatenated codes in Section 10.6. It asserts the existence of codes that exhibit a "gradual" increase in list size as more and more codeword symbols are erased, up to a total of $(1-\varepsilon)n$ erasures.

Lemma 10.7 *For all large enough n, the following hold:*

(i) For every integer $L \geq 1$ and every p, $0 \leq p \leq 1$, there exists an $[n,k]_2$ linear code C with $k = \lfloor (1 - p - \frac{H(p)}{\lceil \lg(L+1) \rceil})n - \sqrt{n} \rfloor$ that is (pn, L)-erasure list-decodable.

(ii) There exists absolute constants $A, B > 0$ such that for every ε, $0 \leq \varepsilon \leq 1$, there exists an $[n,k]_2$ linear code C with $k = \lfloor \frac{\varepsilon n}{\lg(A/\varepsilon)} \rfloor$ that is $(s, \frac{Bn}{n-s})$-erasure list-decodable for every *$s \leq (1-\varepsilon)n$.*

Proof: We first prove Part (i) of the lemma. The proof is based on the probabilistic method. We will pick a code C generated by a random $n \times k$ generator matrix G where k is as specified in the statement of the lemma.[3] We will prove that such a random code is (pn, L)-erasure list-decodable with high probability. By Lemma 10.5, this amounts to estimating the probability that a random $n \times k$ matrix over $\mathbb{F}_2$ has the property that every $t \times k$ submatrix has rank at least $(k - \lfloor \lg L \rfloor)$, where $t = (1-p)n$. For any J', $0 \leq J' \leq k$, the probability that a random matrix M of order $t \times k$ has rank $(k - J')$ is at most $2^{kJ'}2^{-tJ'}$. This follows since for a fixed subspace S of $\mathbb{F}_2^k$ of dimension $(k - J')$, the probability that all t rows of M lie in S is at most $2^{-J't}$, and furthermore the number of subspaces of $\mathbb{F}_2^k$ of dimension $(k-J')$ is at most $2^{kJ'}$ as one can specify such a subspace as the null-space of a $J' \times k$ matrix over $\mathbb{F}_2$. Let $J \stackrel{\rm def}{=} \lceil \lg(L+1) \rceil = 1 + \lfloor \lg L \rfloor$. Then, for $t \geq k$, the probability that a random matrix M of order $t \times k$ has rank $(k - J)$ or less is at most

$$\sum_{J'=J}^{k} 2^{(k-t)J'} \leq k 2^{(k-t)J} \ .$$

Now, by a union bound, the probability that some $t \times k$ submatrix of G has rank at most $(k - J)$ is at most

$$\begin{aligned}
\binom{n}{t} \cdot k 2^{(k-t)J} &= \binom{n}{(1-p)n} \cdot k 2^{(k-t)J} \\
&\leq k \cdot 2^{H(p)n} \cdot 2^{kJ - (1-p)nJ} \\
&\leq k \cdot 2^{H(p)n} \cdot 2^{\left((1-p)J - H(p)\right)n} \cdot 2^{-J\sqrt{n}} \cdot 2^{-(1-p)nJ} \\
&= k \cdot 2^{-J\sqrt{n}} \\
&= o(1) \ ,
\end{aligned}$$

[3] We will assume for simplicity that C has dimension k, i.e. G has full column rank, since this happens with very high probability.

where in the second step we have used the fact that $\binom{n}{(1-p)n} = \binom{n}{pn} \le 2^{H(p)n}$ (cf. [vL99, Chap. 1, Theorem 1.4.5]), and in the third step we substituted the value of k from the statement of Part (i) of the lemma.

Hence an $n \times k$ matrix G every $t \times k$ submatrix of which has rank at least $k - J + 1 = k - \lfloor \lg L \rfloor$ exists, and by Lemma 10.5, therefore, there exists a (pn, L)-erasure list-decodable code of blocklength n and dimension k. This proves Part (i) of the lemma.

To prove Part (ii), we apply the above proof for $p = (1-\varepsilon)$, and $k = \lfloor \frac{\varepsilon}{\lg(8/\varepsilon)} n \rfloor$. Let us estimate the probability that for a fixed s, where $n/2 \le s \le (1-\varepsilon)n$, C is not $(s, \frac{8n}{n-s})$-erasure list-decodable. By Lemma 10.5, this happens only if some $(n-s) \times k$ submatrix of G has rank less than $k - \lg(\frac{8n}{n-s})$. Let $\sigma = (n-s)/n$; we have $\varepsilon \le \sigma \le 1/2$. As in the proof of Part (i), the probability that some $(n-s) \times n$ submatrix of G has rank less than $k - \lg(8/\sigma)$ is at most

$$\begin{aligned}
\binom{n}{n-s} \cdot k \cdot 2^{(k-n+s)\lg(8/\sigma)} &\le k \cdot \left(\frac{e}{\sigma}\right)^{\sigma n} \cdot 2^{\left((\varepsilon/\lg(8/\varepsilon)) - \sigma\right)\lg(8/\sigma) n} \\
&= k \cdot 2^{-n\left(\sigma \lg(8/e) - \frac{\varepsilon \lg(8/\sigma)}{\lg(8/\varepsilon)}\right)} \\
&= 2^{-\Theta(n)}
\end{aligned}$$

where the last step follows since $\sigma \ge \varepsilon$, and in the first step we used the inequality $\binom{n}{\sigma n} \le (e/\sigma)^{\sigma n}$ for $\sigma \le 1/2$.

Now, by a union bound, the probability that for some s, $n/2 \le s \le (1-\varepsilon)n$, C is not $(s, \frac{8n}{n-s})$-erasure list-decodable is also exponentially small. Hence there exists a linear code C of blocklength n and rate $\varepsilon/\lg(8/\varepsilon)$ that is $(s, \frac{8n}{n-s})$-erasure list-decodable for every s that satisfies $n/2 \le s \le (1-\varepsilon)n$. Since the list size for $s < n/2$ erasures is at most the list size for $n/2$ erasures, such a code is also $(s, \frac{16n}{n-s})$-erasure list-decodable for every s, $0 \le s \le (1-\varepsilon)n$. This proves Part (ii) of the claim (with the choice $A = 8$ and $B = 16$ for the absolute constants). □

Remark: Note that the above lemma not only guarantees the existence of codes with the required properties, but also proves that a random code has these properties with very high probability.

10.5.4 Upper bound on $\tilde{R}_L(p)$

We now turn to upper bounds on $\tilde{R}_L(p)$. It is easy to prove that for any fixed L (in fact even for a list size L that is allowed to grow polynomially in the blocklength), we must have $\tilde{R}_L(p) \le 1 - p$. Indeed, let C be a code of blocklength n and rate r, and let $T = \{1, 2, \ldots, (1-p)n\}$. Pick a *random* $y \in \{0,1\}^{(1-p)n}$ and consider the set S_y of codewords $c \in C$ that satisfy $[c]_T = y$. The expected number of such codewords equals $2^{rn}2^{-(1-p)n}$, and hence there must exist a $y \in \{0,1\}^n$ for which $|S_y| \ge 2^{(r-(1-p))n}$. Since we want $|S_y| \le L \le \text{poly}(n)$, we must have $r \le (1-p)$. Hence $\tilde{R}_L(p) \le 1 - p$. Below, we are interested in a better upper bound on $\tilde{R}_L(p)$, which in particular bounds it strictly away from $(1-p)$ for every fixed L (and thereby shows that one requires unbounded list size in order to approach the "capacity" $(1-p)$ of the erasure channel).

Theorem 10.8 *For every integer $L \ge 1$ and every p, $0 \le p \le 1 - 2^{-L}$, we have*

$$\tilde{R}_L(p) \le \min\left\{1 - H(\lambda), 1 - \frac{p}{1 - 2^{-L}}\right\}, \tag{10.6}$$

where λ is the unique root of the equation $\lambda^{L+1}+(1-\lambda)^{L+1}=1-p$ in the range $0\le\lambda\le 1/2$. For $p\ge 1-2^{-L}$, we have $\tilde{R}_L(p)=0$.

Proof: We will first prove that $\tilde{R}_L(p)\le 1-H(\alpha)$ for *any* $0\le\alpha\le 1/2$ that satisfies $\alpha^{L+1}+(1-\alpha)^{L+1}\ge 1-p$. We will later deduce the claimed upper bounds on $\tilde{R}_L(p)$ from this fact.

Let α such that $\alpha^{L+1}+(1-\alpha)^{L+1}\ge 1-p$; we wish to prove that $\tilde{R}_L(p)\le 1-H(\alpha)$. Let C be a (pn,L)-erasure list-decodable code of blocklength n with M_0 codewords. Our goal is to prove an upper bound on M_0.

The proof follows along the lines of the Elias-Bassalygo upper bound on the rate of a code in terms of its minimum distance (cf. [vL99, Section 5.2]). Pick a random $v\in\{0,1\}^n$ and consider the subset C' of all codewords of C that are at a Hamming distance αn from v. The expected size of C' equals $|C|\binom{n}{\alpha n}2^{-n}\ge M_0 2^{(H(\alpha)-1)n-o(n)}$. Hence there exists such a code C' with

$$|C'|=M\ge M_0 2^{(H(\alpha)-1)n-o(n)}\ . \tag{10.7}$$

By shifting the origin to v, we can assume that all codewords of C' have Hamming weight exactly αn. We will prove an upper bound on M, and by Equation (10.7) this will imply an upper bound on M_0 as well.

We will prove an upper bound on M by counting the total number N of pairs (S,i) such that S is an $(L+1)$-element subset of C' and $1\le i\le n$, and all codewords in S agree in position i. Since C is (pn,L)-erasure list-decodable and C' is a subset of C, C' is also (pn,L)-erasure list-decodable. Hence for any such subset S of $(L+1)$ codewords from C', the number of codeword positions where all codewords in S agree is at most $((1-p)n-1)$. We therefore have

$$N\le\binom{M}{L+1}((1-p)n-1)\ . \tag{10.8}$$

We next establish a lower bound on N. Arrange the codewords in C' in the form of a $M\times n$ matrix in the obvious way with the rows being the various codewords. Let a_i be the fraction of 1's in the i'th column of this matrix. Since each codeword of C' has weight exactly αn, we have $\sum_{i=1}^n a_i=\alpha n$. Also, by definition we have

$$\begin{aligned} N &= \sum_{i=1}^{n}\left[\binom{a_iM}{L+1}+\binom{(1-a_i)M}{L+1}\right] \\ &\ge n\left[\binom{\alpha M}{L+1}+\binom{(1-\alpha)M}{L+1}\right] \end{aligned} \tag{10.9}$$

where we have used the fact that $\sum_i a_i=\alpha n$, and hence the minimum value of

$$\sum_{i=1}^{n}\left[\binom{a_iM}{L+1}+\binom{(1-a_i)M}{L+1}\right]$$

is achieved when each $a_i=\alpha$. We now claim that any $0\le\beta\le 1$ and large enough M,

$$\binom{\beta M}{L+1}\ge\Big(\beta^{L+1}-\frac{2L^2\beta^L}{M}\Big)\binom{M}{L+1}\ . \tag{10.10}$$

The above is clearly true if $\beta = 0, 1$, so assume $0 < \beta < 1$. Also assume $M \geq 2L/\beta$ (as otherwise we will already have a good upper bound on M). Now,

$$\begin{aligned}\binom{\beta M}{L+1}\binom{M}{L+1}^{-1} &= \frac{\beta M(\beta M - 1)\cdots(\beta M - L)}{M(M-1)\cdots(M-L)} \\ &\geq \beta\Big(\beta - \frac{L}{M-L}\Big)^L \\ &\geq \beta^{L+1}\Big(1 - \frac{2L}{\beta M}\Big)^L \quad \text{(since } M \geq 2L/\beta \geq 2L) \\ &\geq \beta^{L+1}\Big(1 - \frac{2L^2}{\beta M}\Big) \quad \text{(using } (1-x)^L \geq 1 - xL \text{ for } x \leq 1 \text{ and } L \geq 1),\end{aligned}$$

giving Equation (10.10).

Now, combining (10.8), (10.9) and (10.10), we get, assuming $M \geq 2L/\alpha$, that

$$M(\alpha^{L+1} + (1-\alpha)^{L+1} - (1-p) + 1/n) \leq 2L^2(\alpha^L + (1-\alpha)^L) .$$

Thus as long as $\alpha^{L+1} + (1-\alpha)^{L+1} \geq 1 - p$, we have $M \leq \max\{2L/\alpha, 2L^2 n\}$. Recalling Equation (10.7), the size M_0 of the original code C satisfied $M_0 \leq M 2^{(1-H(\alpha))n + o(n)}$. Hence we get $M_0 \leq 2^{(1-H(\alpha))n+o(n)}$, and $R(C) = \frac{\lg M_0}{n} \leq 1 - H(\alpha) + o(1)$. Since C was an arbitrary (pn, L)-erasure list-decodable code, this proves that $\tilde{R}_L(p) \leq 1 - H(\alpha)$, as we set out to prove.

Recall that α was any real number in the range $[0, 1/2]$ that satisfied $\alpha^{L+1} + (1-\alpha)^{L+1} \geq 1 - p$. Now, if $p \geq 1 - 2^{-L}$, we can pick $\alpha = 1/2$ and this will imply $\tilde{R}_L(p) = 0$. For the case when $p < 1 - 2^{-L}$, we note that the function $f(\alpha) = \alpha^{L+1} + (1-\alpha)^{L+1}$ is decreasing in the range $0 \leq \alpha \leq 1/2$ with $f(0) = 1 \geq 1 - p$ and $f(1/2) = 2^{-L} < 1 - p$, and thus the equation $f(\alpha) = 1 - p$ has a unique solution, say λ, in the range $0 \leq \lambda < 1/2$. We can then use $\alpha = \lambda$ and conclude $\tilde{R}_L(p) \leq 1 - H(\lambda)$, which gives the first upper bound claimed in Equation (10.6).

It now remains to prove the second upper bound $\tilde{R}_L(p) \leq 1 - p/(1 - 2^{-L})$ in the range $0 \leq p \leq 1 - 2^{-L}$. We know that $\tilde{R}_L(0) = 1$ and $\tilde{R}_L(1 - 2^{-L}) = 0$, and this upper bound simply amounts to proving that $\tilde{R}_L(p)$ always lies on or below the straight line joining the points $(0, 1)$ and $(1 - 2^{-L}, 0)$. We prove this by a standard "puncturing" argument. Let C be a (pn, L)-erasure list-decodable code. Let $\gamma = 1 - \frac{p}{1-2^{-L}}$. For each $a \in \{0,1\}^{\gamma n}$, define C_a to be the subcode of C consisting of all codewords whose first γn positions agree with a, and let C'_a be the code obtained from C_a by puncturing the first γn positions. The blocklength of C'_a equals $n' = (1-\gamma)n$. Since C is (pn, L)-erasure list-decodable, for each $a \in \{0,1\}^{\gamma n}$, C'_a must also be (pn, L)-erasure list-decodable. Now $pn = pn'/(1-\gamma) = (1 - 2^{-L})n'$, and since $\tilde{R}_L(1 - 2^{-L}) = 0$, we have $|C_a| = |C'_a| = 2^{o(n)}$ for each $a \in \{0,1\}^{\gamma n}$. Hence $|C| \leq 2^{\gamma n + o(n)}$. Recalling that $\gamma = 1 - \frac{p}{1-2^{-L}}$, we have $\tilde{R}_L(p) \leq 1 - \frac{p}{1-2^{-L}}$, as claimed. □

Corollary 10.9 *For every integer $L \geq 1$ and every p, $0 < p < 1$, we have $\tilde{R}_L(p) < 1 - p$.*

Comment on the bound (10.6)

For $L = 1$, the two bounds proven in Theorem 10.8 are precisely the Elias-Bassalygo and Plotkin bounds on the rate of a code in terms of its minimum distance, which state that $R(\delta) \leq 1 - H(\frac{1-\sqrt{1-2\delta}}{2})$ and $R(\delta) \leq 1 - 2\delta$, respectively. For large L, the bound $\tilde{R}_L(p) \leq 1 - p/(1 - 2^{-L})$ is better than the other bound $\tilde{R}_L(p) \leq 1 - H(\lambda)$ except for *very small* p (less than about $L/2^L$).

10.5.5 Improved upper bound for $\tilde{R}_L^{\text{lin}}(p)$

The previous upper bound applied to general binary codes. We next ask the question whether a better upper bound is possible if one restricts attention to binary linear codes, i.e. whether an upper bound better than (10.6) exists for $\tilde{R}_L^{\text{lin}}(p)$. In this section, we obtain a different upper bound that implies that, for at least some settings of parameters, the answer to this question is yes. As a consequence, we will in fact be able to exhibit a *provable separation* between the power of linear and non-linear codes with respect to erasure list-decodability.

Lemma 10.10 *For every integer L and every p, $0 \le p \le (1 - 1/2L)$,*

$$\tilde{R}_L^{\text{lin}}(p) \le \frac{L(\lg L + 2)}{2L - 1}\Big(1 - p - \frac{1}{2L}\Big) .$$

Proof: Let C_1 be a linear code of blocklength n and dimension $k = rn$ that is (pn, L)-erasure list-decodable. We will use C_1 to construct a non-linear (in fact, pseudolinear) code C_2 of smaller, but related, dimension but which has much better erasure list-decodability. We will then appeal to the bound of Theorem 10.8 to conclude an upper bound on the rate of C_2, which in turn will imply an upper bound on the rate r of C_1.

We will take C_2 to be an $(n, k')_2$ pseudolinear code that encodes $\mathbf{x} \in \mathbb{F}_2^{k'}$ into $C_1(H_{\mathbf{x}})$, where H is a $k \times (2^{k'} - 1)$ parity check matrix of a BCH code of designed distance $d = \lfloor \lg L \rfloor + 3$, and $H_{\mathbf{x}}$ denotes the column number $\mathbf{x}$ of H (here we view $\mathbf{x}$ as an integer in the range $[0, 2^{k'} - 1]$ and use the convention $H_{\mathbf{0}} = \mathbf{0}$). Thus any $d - 1 = \lfloor \lg L \rfloor + 2$ columns of H are linearly independent. Define $\tilde{L} = \lfloor \lg L \rfloor + 1 = \lceil \lg(L+1) \rceil$. Then any $\tilde{L} + 1$ columns of H are linearly independent. Such a matrix H exists with $k \le k' \frac{\tilde{L}+1}{2}$ (cf. [MS81], [AS92, Chap. 15]).

We will now prove that if C_1 is (pn, L)-erasure list-decodable, then C_2 is $(pn, \tilde{L})$-erasure list-decodable. Suppose this is not the case. This means that there exists $T \subset [n]$ with $|T| = (1-p)n$ and $\mathbf{r} \in \mathbb{F}_2^{(1-p)n}$, such that at least $m \stackrel{\text{def}}{=} \tilde{L} + 1$ codewords of C_2 agree with $\mathbf{r}$ on the positions in T. Let these codewords be $C_2(\mathbf{x_1}), C_2(\mathbf{x_2}), \ldots, C_2(\mathbf{x_m})$. By the definition of C_2 from C_1, this implies that the messages $H_{\mathbf{x_j}}$ for $1 \le j \le m$ are all solutions to $C_1(\mathbf{y})_{|T} = \mathbf{r}$ (i.e. their encodings by C_1 agree with $\mathbf{r}$ in every position in T).

We now consider two cases: $\mathbf{r} = \mathbf{0}$ and $\mathbf{r} \ne \mathbf{0}$. In the first case, we can assume that $\mathbf{x_m}$ (and hence $H_{\mathbf{x_m}}$) is the zero vector. The set of solutions $\mathbf{y} \in \mathbb{F}_2^k$ to $C_1(\mathbf{y})_{|T} = \mathbf{0}$ form a linear subspace W that includes the vectors $H_{\mathbf{x_j}}$ for $1 \le j \le m - 1$. By the property of the matrix H, any $\tilde{L} + 1$ columns of it are linearly independent, and hence the vectors $H_{\mathbf{x_j}}$, $1 \le j \le m - 1$ form a set of linearly independent vectors. Hence the dimension of W as a vector space over $\mathbb{F}_2$, $\dim(W)$, is at least $m - 1 = \tilde{L}$, and hence $|W| \ge 2^{\tilde{L}} \ge L + 1$ (recalling that $\tilde{L} = \lceil \lg(L+1) \rceil$). This contradicts the (pn, L)-erasure list decodability of C_1.

Let us now consider the case $\mathbf{r} \ne \mathbf{0}$. In this case the set of solutions to $C_1(\mathbf{y})_{|T} = \mathbf{r}$ form an affine space (i.e. is of the form $\mathbf{w} + W$ for some subspace W). By the list decodability property of C_1 we know that $|W| \le L$, and hence $\dim(W) \le \lfloor \lg L \rfloor = \tilde{L} - 1$. The maximum number of linearly independent vectors in $\mathbf{w} + W$ can be at most $\dim(W) + 1 \le \tilde{L}$. This contradicts the fact that $H_{\mathbf{x_j}}$, $1 \le j \le \tilde{L} + 1$, all lie in $\mathbf{w} + W$, since, by the property of the matrix H, these form a set of $\tilde{L} + 1$ mutually linearly independent vectors.

Thus we get a contradiction in either case, and hence C_2 is $(pn, \tilde{L})$-erasure list-decodable. Appealing to Theorem 10.8, the rate of C_2, $R(C_2)$, satisfies

$$R(C_2) \le 1 - \frac{p}{1 - 2^{-\tilde{L}}} .$$

Now, since the dimension of C_1 is at most $\frac{\tilde{L}+1}{2}$ times that of C_2, we have

$$
\begin{aligned}
R(C_1) &\le \frac{\tilde{L}+1}{2}\Big(1 - \frac{p}{1-2^{-\tilde{L}}}\Big) \\
&\le \frac{\tilde{L}+1}{2}\Big(1 - \frac{p}{1-1/2L}\Big) \qquad (\text{using } \tilde{L} = 1 + \lfloor \lg L \rfloor) \\
&= \frac{L(\lg L + 2)}{2L-1}\Big(1 - p - \frac{1}{2L}\Big) ,
\end{aligned}
$$

where the last step follows since $\tilde{L} = 1 + \lfloor \lg L \rfloor \le \lg L + 1$. □

10.5.6 Provable separation between erasure list-decodable linear and non-linear codes

The following result shows that list decoding up to a fraction $(1-\varepsilon)$ of erasures can be accomplished using non-linear codes with an exponentially smaller list size compared to that achievable using linear codes.

Theorem 10.11 *Let $\varepsilon > 0$ and $\mathcal{C}^{\text{lin}}$ be a binary linear code family of positive rate with* $\mathsf{ErasureLDR}_L(\mathcal{C}^{\text{lin}}) \ge 1-\varepsilon$. *Then $L = \Omega(1/\varepsilon)$. On the other hand, there exists a binary code family $\mathcal{C}$ of positive rate (in fact, rate $\Omega(\varepsilon)$) with* $\mathsf{ErasureLDR}_{L'}(\mathcal{C}) \ge 1-\varepsilon$ *for $L' = O(\log(1/\varepsilon))$.*

Proof: The lower bound on list size for linear codes follows from Lemma 10.10 with the setting $p = 1-\varepsilon$. The claim about general, non-linear codes follows from Theorem 10.3 with the setting $p = 1-\varepsilon$ (and using $H(\varepsilon) \le O(\varepsilon \log(1/\varepsilon))$). In fact by Lemma 10.4, we can achieve a similar performance with a family of $(L', 2)$-pseudolinear codes as well. □

10.6 A good erasure list-decodable binary code construction

10.6.1 Context

We have so far investigated the best rate possible for codes with a certain erasure list-decodability. These results were all probabilistic and demonstrated that good codes exist in abundance. However, they did not give an efficient procedure to construct them deterministically. We now move on to the question of efficient constructions of such codes and efficient algorithms to decode them from erasures. We will focus on binary codes for this section.

As for the errors case, we will again focus on the high-noise regime to state and prove our results. For the erasures case, this means list decoding when up to a fraction $(1-\varepsilon)$ of symbols could be adversarially erased. We loosely give such codes the label "highly erasure list-decodable codes".

The existential results state that the best rate one can hope for such codes is $\Omega(\varepsilon)$ (with a list size of $O(\log(1/\varepsilon))$ for non-linear codes and $1/\varepsilon^{O(1)}$ for linear codes). The relation between the erasure list decoding radius and minimum distance established in Section 10.4, implies that we can construct highly erasure list-decodable codes by using binary linear codes of relative distance $1/2 - O(\varepsilon)$. The best explicit constructions of such large distance binary codes achieves a rate of $\Omega(\varepsilon^3)$ [ABN+92, She93]. Thus, we can also construct explicit codes of rate $\Omega(\varepsilon^3)$ which are efficiently list decodable from a fraction $(1-\varepsilon)$ of erasures. (Since the code is linear, the fact that the list size is small up to a fraction $(1-\varepsilon)$ of erasures immediately gives an at most cubic time algorithm to find and output the list.)

We now present a concatenated code construction that improves this bound and achieves a rate of $\Omega(\varepsilon^2/\log(1/\varepsilon))$. We stress that our result is **not** obtained by appealing to the above mentioned distance to erasure list decoding radius relation. Indeed no polynomial time constructions of binary code families of relative distance $(1/2 - O(\varepsilon))$ and rate about ε^2 are known. In fact such a construction, which will asymptotically match the Gilbert-Varshamov bound at low rates, will be a *major* breakthrough in coding theory.[4]

10.6.2 The formal result

Theorem 10.12 *For every $\varepsilon > 0$, there exists a family of binary linear codes of rate $\Omega(\varepsilon^2/\log(1/\varepsilon))$ such that a code of blocklength in the family can be constructed in $2^{\varepsilon^{-O(1)}} + \text{poly}(N, 1/\varepsilon)$ time, and can be list decoded in polynomial time using lists of size $O(1/\varepsilon)$ when up to a fraction $(1-\varepsilon)$ of its symbols are erased.*

The above result is the main result of this section. It follows immediately from the following lemma.

Lemma 10.13 *There exist absolute constants b, d such that for all large enough integers K and all small enough $\varepsilon > 0$, there exists a binary linear code C_K that has the following properties:*

(i) C_K has dimension K and blocklength $N \le \frac{bK\log(1/\varepsilon)}{\varepsilon^2}$.

(ii) The generator matrix of C_K can be constructed in $2^{\varepsilon^{-O(1)}} + \text{poly}(N, 1/\varepsilon)$ time.

(iii) C_K is $((1-\varepsilon)N, \frac{d}{\varepsilon})$-erasure list-decodable (and since C_K is linear there is an $O(N^3)$ time list decoding algorithm to decode up to $(1-\varepsilon)N$ erasures using lists of size $O(1/\varepsilon)$).

Proof: The code C_K is constructed by concatenating an outer code C_{out} over $\text{GF}(2^m)$ of blocklength n_0, dimension $k_0 = K/m$ and minimum distance d_0, with an inner code C_{in} as guaranteed by Part (ii) of Lemma 10.7. By using a construction in [ABN+92], we can pick parameters so that $k_0 = K/m = \Omega(\varepsilon n_0)$, $m = O(1/\varepsilon)$ and $d_0 = (1 - O(\varepsilon))n_0$ (for convenience, we hide constants using big-Oh notation, but we stress that these are absolute constants that do not depend on ε).[5] We note that this choice of C_{out} is not linear over $\text{GF}(2^m)$ (though it *is* additive), but a (succinct) description of C_{out} can be constructed in time $\text{poly}(n, 1/\varepsilon)$. Moreover, after concatenation with an inner binary linear code, the overall concatenated code will be a binary linear code.

The inner code C_{in} will be a code as guaranteed by Lemma 10.7 of dimension m and blocklength $n_1 = O(\frac{m\lg(1/\varepsilon)}{\varepsilon})$ that is $((1-\sigma)n_1, B/\sigma)$-erasure list-decodable for every $\sigma \ge \varepsilon/2$. We can construct such a code by a brute-force search in $2^{O(mn_1)} = 2^{\varepsilon^{-O(1)}}$ time.

[4]The reader might recall that the same applied to our construction in Chapter 8 of binary codes of rate $\Omega(\varepsilon^4)$ list decodable up to a fraction $(1/2-\varepsilon)$ of errors.

[5]Actually, we can use algebraic-geometric codes (specifically those discussed in Section 6.3.9) and even have $m = O(\log(1/\varepsilon))$. But we prefer to use the codes from [ABN+92] as they are easier to construct and have lower construction complexity, and we do not want to give the impression that we need complicated AG-codes for our construction. We will return to the performance attainable using AG-codes in Section 10.6.3.

Once we have descriptions of C_{out} and C_{in}, one can construct the generator matrix of the concatenated code C_K in $\text{poly}(N, 1/\varepsilon)$ time where $N = n_0 n_1$ is the blocklength of C_K. Now $N = n_0 \cdot n_1 = O(\frac{K}{m\varepsilon}) \cdot O(\frac{m \lg(1/\varepsilon)}{\varepsilon}) = O(\frac{K \lg(1/\varepsilon)}{\varepsilon^2})$. This proves Parts (i) and (ii) of the statement of the lemma.

We now prove that C_K is $((1-\varepsilon)N, O(1/\varepsilon))$-erasure list-decodable. Let y be a received word with $(1-\varepsilon)N$ erasures, and let $c'_1, c'_2, \ldots, c'_M$ be the codewords in C_K "consistent" with y, and let c_j be the codeword of C_{out} corresponding to c'_j, for $1 \le j \le M$. We wish to prove that $M = O(1/\varepsilon)$. Let y_i be the portion of y corresponding to the i'th outer codeword position, for $1 \le i \le n_0$. Let the number of symbols in y_i be $\sigma_i n_1$. We have $\sum_i \sigma_i = \varepsilon n_0$. Define $Q = \{i : \sigma_i \ge \varepsilon/2\}$. Clearly we have

$$\sum_{i \in Q} \sigma_i \ge \frac{\varepsilon n_0}{2} \ . \tag{10.11}$$

Now define "weights" $w_{i,\beta}$ for $1 \le i \le n_0$ and $\beta \in \text{GF}(2^m)$ as follows. If $i \notin Q$, set $w_{i,\beta} = 0$ for all $\beta \in \text{GF}(2^m)$. If $i \in Q$, then $\sigma_i \ge \varepsilon/2$ and hence by construction C_{in} is $((1-\sigma_i)n_1, B/\sigma_i)$-erasure list-decodable. In other words, if $J_i = \{\beta : C_{\text{in}}(\beta) \text{ is consistent with } y_i\ \}$, then $|J_i| \le B/\sigma_i$. We set (for $i \in Q$):

$$w_{i,\beta} = \begin{cases} \sigma_i & \text{if } \beta \in J_i \\ 0 & \text{if } \beta \notin J_i \end{cases}$$

Since $|J_i| \le B/\sigma_i$, our choice of weights clearly satisfy

$$\sum_{i,\beta} w_{i,\beta}^2 \le B \sum_{i \in Q} \sigma_i \ . \tag{10.12}$$

We now use the combinatorial result of Corollary 3.7 from Chapter 3 to obtain an upper bound on the number of codewords of C_{out} that satisfy a certain weighted condition depending on the distance d_0 of C_{out}. Recalling the result of Corollary 3.7, for any $\varepsilon' > 0$, we have that the the number of codewords $(\alpha_1, \alpha_2, \ldots, \alpha_{n_0}) \in C_{\text{out}}$ that satisfy

$$\sum_{i=1}^{n_0} w_{i,\alpha_i} \ge \Big((n_0 - d_0(1-\varepsilon')) \sum_{i,\beta} w_{i,\beta}^2 \Big)^{1/2} \tag{10.13}$$

is at most $1/\varepsilon'$.

Now for each c_j, $1 \le j \le M$, and each $i \in Q$, we have $c_{j,i} \in J_i$, where $c_{j,i} \in \text{GF}(2^m)$ denotes the i'th symbol of c_j. Now, $w_{i,c_{j,i}} = \sigma_i$ for every $i \in Q$ and $w_{i,c_{j,i}} = 0$ for $i \notin Q$. Thus, we have

$$\sum_{i=1}^{n_0} w_{i,c_{j,i}} = \sum_{i \in Q} \sigma_i \ . \tag{10.14}$$

Combining Equations (10.11), (10.12) and (10.14), we have that the codeword c_j satisfies Condition (10.13) as long as

$$\frac{\varepsilon n_0}{2} \ge (n_0 - d_0(1-\varepsilon'))B \ ,$$

which can be satisfied provided $d_0 \ge n_0(1 - \frac{\varepsilon}{2B})(1-\varepsilon')^{-1}$. Picking $\varepsilon' = \frac{\varepsilon}{4B}$, we only need $d_0 = n_0(1 - O(\varepsilon))$ which is satisfied for our choice of C_{out}. Hence the number of codewords consistent with the received word y is at most $1/\varepsilon' = O(1/\varepsilon)$, thus proving Part (iii) of the lemma as well. □ *(Lemma 10.13)*

Remark: The best previous results deducible by using codes of large $(1/2 - O(\varepsilon))$ relative minimum distance and appealing to the erasure list decoding radius to minimum distance relation from Section 10.4 can achieve a blocklength of $N = \min\{O(\frac{K^2}{\varepsilon^2}), O(\frac{K}{\varepsilon^3})\}$. Thus, our result achieves the best features of both these bounds and obtains a blocklength N that has a linear dependence on K and a near-quadratic dependence on ε.

10.6.3 Some improvements

Obtaining near-linear encoding and near-quadratic decoding times

The codes constructed in Lemma 10.13 being linear can be encoded in $O(N^2)$ time and can be list decoded from a fraction $(1 - \varepsilon)$ of erasures in $O(N^3)$ time by solving a linear system. By using Reed-Solomon codes as outer code, the encoding can be accomplished in $O(N \log^{O(1)} N)$ time. This is because the outer encoding can be performed using $O(N \log N)$ field operations by employing FFT based methods, and then the inner encoding just takes $O(\log^2 N)$ time for each of the N outer codeword symbols. For decoding, the inner codes can be decoded in $O(\log^3 N)$ time by solving a linear system, and then the Reed-Solomon code can be decoded from a set of weights that satisfy Condition (10.13) in near-quadratic time (cf. Chapter 6). There is, however, a potential problem in using Reed-Solomon codes as outer codes. In such a case, the inner code needs to be a linear code of dimension $\Omega(\log N)$, and hence a brute force search for the inner code will take time which has a quasi-polynomial dependence on N. Therefore, if we seek a deterministic construction, then the construction time appears to be no longer polynomial in N.

Nevertheless there is a way to obtain a polynomial time construction of the necessary inner code. A random linear code will have the erasure list-decodability property required by the inner code with high probability (cf. Lemma 10.7 and the remark following it). This randomized construction can be derandomized using the method of conditional expectations. The approach is similar to that taken to derandomize the construction of pseudolinear codes from Section 9.3.3, though the specific implementation details are now different.

To pick an $[n, k]_2$ linear code, we pick the n rows of the $n \times k$ generator matrix G in sequence, each time going through all possible choices in $\mathbb{F}_2^k$ and picking the one which minimizes a certain conditional expectation. The relevant conditional expectation in question concerns the expected number of $t \times k$ submatrices of G which have rank at most $k - \log(n/t) - c$, summed over all t, $\varepsilon n \le t \le n$ (here c is a large enough *absolute* constant). Lemma 10.5 implies that finding a generator matrix where there are no such submatrices is equivalent to the construction of an $[n, k]_2$ linear code that satisfies the requirement of Lemma 10.7, Part (ii). To be able to compute the relevant conditional expectations easily, we will introduce indicator random variables whose expectation provides a pessimistic estimation of the true expectation, and compute their expectations instead.

We will introduce an indicator random variable $I(T, S)$ for each $T \subseteq [n]$ with $|T| = t \ge \varepsilon n$ and each $S \subseteq \mathbb{F}_2^k$ of linearly independent vectors with $|S| = \log(n/t) + c$. We will define $I(T, S) = 1$ if the span of the rows of the generator matrix G that are indexed by T is contained in the nullspace of the span of S, and $I(T, S) = 0$ otherwise. Thus if $I(T, S) = 1$, then the $|T| \times k$ submatrix of G corresponding to rows in T has rank at most $k - |S|$, and provides a "counterexample" to the code constructed having the desired erasure list-decodability property. The event "$I(T, S) = 1$" is therefore one that we would like to avoid, for every choice of T, S. It is easy to see that for each fixed T, S, the conditional expectation of $I(T, S)$ after the first few rows of G have been fixed (taken over the random choice of the remaining rows) can be computed exactly, since it is simply the probability that each of the rows in T that have not been fixed yet lie in the nullspace of S.

Now, define the random variable $X = \sum_{T,S} I(T, S)$. By linearity of expectation, the conditional expectation of X can be computed exactly for each choice of a subset of rows of G. The initial expectation of X, say $\mathbf{E}$, is exponentially small in n provided $k = \Theta(\varepsilon n/\log(1/\varepsilon))$ (this follows from the proof of Part (ii) of Lemma 10.7).

Hence the above derandomization procedure will find a matrix G whose X value is at most $\mathbf{E}$, and since for each fixed G, the random variable X takes on an integer value, we must have $X(G) = 0$. This implies in turn that G has the required rank property for its submatrices, as desired.

The runtime of this derandomization procedure is dominated by the time to consider all possible candidates for the sets T and S above. Therefore, it takes

$$2^{O(H(\varepsilon)n)} \cdot 2^{O(k \log(1/\varepsilon))} = 2^{O(\varepsilon \log(1/\varepsilon)n)} = 2^{O(k \log^2(1/\varepsilon))}$$

time (since $k = \Theta(\varepsilon n/\log(1/\varepsilon))$.)

Applying this to the context of concatenation with an outer Reed-Solomon code where the dimension $k = O(\log N)$, we conclude that the generator matrix of a linear inner code that has the required properties can be computed in $N^{O(\log^2(1/\varepsilon))}$ time. This is dominant component in the construction of the concatenated code, and therefore the overall code can be constructed in $N^{O(\log^2(1/\varepsilon))}$ time.

To summarize, we can prove the result of Lemma 10.13 with codes that have a near-linear time encoder, a quadratic time list decoder, and an $N^{O(\log^2(1/\varepsilon))}$ time deterministic construction.

Improving construction time using AG-codes

Despite the impressive encoding and decoding times, the drawback of the above construction with Reed-Solomon codes as the outer code is the rather high exponent of N in the construction time. In particular, the code family ceases to be *uniformly constructive*, since the construction time is no longer of the form $O(f(\varepsilon)N^a)$ where a is an absolute constant independent of ε.

Instead of Reed-Solomon codes, we can use AG-codes of distance $(1 - O(\varepsilon))$ as the outer code. Such codes of rate $\Omega(\varepsilon)$ exist over an alphabet size of $O(1/\varepsilon^2)$ (cf. Section 6.3.9). Now, the dimension k of the inner binary code is only $O(\log(1/\varepsilon))$. Applying the same derandomization procedure discussed above, we can now deterministically find a binary linear inner code which has the required erasure list-decodability properties in $2^{O(\log^3(1/\varepsilon))}$ time.

Since the overall code is linear, the claims of quadratic encoding time and cubic decoding time still holds. Thus, we can obtain the same properties as the codes from Lemma 10.13 with an improved construction time of $2^{O(\log^3(1/\varepsilon))} + \mathrm{poly}(N, 1/\varepsilon)$. We formally record this improvement below.

Proposition 10.14 *There exist absolute constants b, d such that for all large enough integers K and all sufficiently small $\varepsilon > 0$, there exists a binary* linear *code C_K that has the following properties:*

(i) C_K *has dimension* K *and blocklength* $N \leq \frac{bK \lg(1/\varepsilon)}{\varepsilon^2}$.

(ii) The generator matrix of C_K *can be constructed in* $2^{O(\log^3(1/\varepsilon))} + \mathrm{poly}(N, 1/\varepsilon)$ *time.*

(iii) C_K *is* $((1-\varepsilon)N, \frac{d}{\varepsilon})$*-erasure list-decodable. In other words, the* $N \times K$ *generator matrix* G *of* C_K *has the property that every* $(\varepsilon N \times K)$ *submatrix has rank at least* $(k - \lfloor \lg(d/\varepsilon) \rfloor)$*. Also, since since* C_K *is linear, there is an* $O(N^3)$ *time decoding algorithm to list decode it from up to* $(1-\varepsilon)N$ *erasures.*

10.6.4 The ε^2 "rate barrier" for binary linear codes

We now present a connection between good erasure list-decodable binary linear codes and certain bipartite Ramsey graphs. This indicates that improving the rate in Proposition 10.14 to, say ε^a for some $a < 2$, without worsening the other parameters significantly, is likely to be very difficult. This connection was pointed out to us by Noga Alon [Alo99].

For a Boolean matrix, a submatrix is said to be *monochromatic* if either all its entries equal 0 or all of them equal 1. The bipartite Ramsey problem asks for a construction of $n \times n$ Boolean matrices which have no $p \times p$ monochromatic submatrices (where p is a certain function of n), for infinitely many values of n. Such a matrix has an obvious connection to $n \times n$ bipartite graphs which have no complete bipartite subgraph $K_{p,p}$ or its complement as an induced subgraph. Hence the terminology "bipartite Ramsey problem" is used to refer to this problem. A straightforward application of the probabilistic method shows that, for all large enough n, there exist such matrices which achieve $p = O(\log n)$. However, an explicit or polynomial time construction of such a matrix is a much harder task. The best known polynomial time constructions of $n \times n$ 0-1 matrices only rule out the existence of $p \times p$ monochromatic submatrices for p about $\sqrt{n}$. A polynomial time construction with p much smaller than $\sqrt{n}$ is a folklore open problem. The following result shows a connection between certain erasure list-decodable codes and bipartite Ramsey graphs. It shows that achieving a rate of $\Omega(\varepsilon^a)$ for some $a < 2$ for our linear code construction from the previous sections, will, under some conditions, imply an improvement to the "$\sqrt{n}$ bound" for the bipartite Ramsey problem.

Proposition 10.15 (Alon) *Assume that for every large enough integer* k *and every* $\varepsilon > 0$*, there exists an* $[n,k]_2$ *linear code* C *of rate* $k/n = \varepsilon^a$*,* $a > 1$*, such that (a) the generator matrix of* C *can be constructed in* $O(n^c f(\varepsilon))$ *time where* c *is an absolute constant and* f *is an arbitrary real-valued function, and (b)* C *is* $((1-\varepsilon)n, 1/\varepsilon^{O(1)})$*-erasure list-decodable. Then, for any* $\gamma > 0$*, for infinitely many* n*, there exists an* $O(n^c f(n^{-(1/a-\gamma)}))$ *time construction of an* $n \times n$ *matrix over* $\mathbb{F}_2$ *which has no monochromatic* $p \times p$ *submatrix, i.e., no* $p \times p$ *submatrix that consists of all* 0*'s or all* 1*'s, for* $p = n^{1-1/a+\gamma}$.

Proof: For a large enough integer n and small enough $\gamma > 0$, pick $\varepsilon = n^{-(1/a-\gamma)}$. Then by the hypothesis, there exists an $[n,k]_2$ linear code C with the claimed properties for $k = \varepsilon^a n = n^{a\gamma}$. Let A be the $n \times k$ generator matrix of C. Define $p = \varepsilon n = n^{1+\gamma-1/a}$. By Lemma 10.5, the assumed erasure list-decodability property of C implies that the rank of each $p \times k$ submatrix of A is at least $(k - O(\log 1/\varepsilon)) = k - O(\log n)$. Define j to be the smallest integer for which the number of subsets of $\{1, 2, \ldots, k\}$ of size exactly j is at least n. Since $k = n^{a\gamma}$, we have $j \leq \frac{2}{a\gamma}$ (for large enough n). .

Construct an $n \times n$ matrix B by having its columns be all the linear combinations (over $\mathbb{F}_2$) of exactly j distinct columns of A. (This might create more than n columns since the number of such combinations could exceed n; in this case we just pick a subset of n linear combinations arbitrarily.)

We will prove that B has no $p \times p$ monochromatic submatrix consisting of all 0's or all 1's. Suppose not, and let B have a $p \times p$ submatrix consisting of only 0's (the case of all 1's can be dealt with similarly). Let I be the set of rows involved in this submatrix, and let $\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_k}$ be the restrictions of the columns of A to the rows in I. Let M be the $p \times k$ submatrix of A indexed by the rows in I.

By assumption about the $p \times p$ all 0s submatrix, we have at least p linear combinations, each of exactly j of the vectors $\mathbf{v_i}$, all giving the all 0's vector. This implies that the nullspace of M (i.e. the space $\{\mathbf{x} \in \mathbb{F}_2^k : M\mathbf{x} = \mathbf{0}\}$) has at least p non-zero vectors of Hamming weight exactly j. It is an easy exercise to show that the dimension of such a space must be at least $p^{1/j}$. Hence we have

$$\mathrm{rank}(M) \le k - p^{1/j} = k - n^{(1+\gamma-1/a)/j} \le k - n^{(a\gamma(1+\gamma)-\gamma)/2} , \tag{10.15}$$

since $j \le 2/(a\gamma)$. On the other hand, by hypothesis we have that every $p \times k$ submatrix of A has rank at least $(k - O(\log n))$ and hence we must have $\mathrm{rank}(M) \ge k - O(\log n)$. This contradicts Equation (10.15) for large enough n.

Hence the matrix B has no $p \times p$ monochromatic submatrix as claimed. The claimed construction time for B follows easily from the choice of parameters, and the proof is complete. $\square$

Corollary 10.16 *Suppose we could prove the result of Proposition 10.14 with a rate of $\Omega(\varepsilon^a)$ for some $1 \le a < 2$. Then there exists a quasi-polynomial time construction of $n \times n$ matrices over $\{0, 1\}$ which have no monochromatic $n^b \times n^b$ submatrix for some $b < 1/2$.*

Proof: Pick b so that $1 - 1/a < b < 1/2$ (this is possible since $a < 2$), and let $\gamma = b - (1 - 1/a)$. For $\varepsilon = n^{-(1/a-\gamma)} = n^{b-1}$, the construction time of the code guaranteed by Proposition 10.14 is $2^{O(\log^3 n)}$, and hence by Proposition 10.15, we have a $2^{O(\log^3 n)}$ time construction of an $n \times n$ 0-1 matrix with no monochromatic $p \times p$ submatrix where $p = n^{1-1/a+\gamma} = n^b$. $\square$

As remarked earlier, a polynomial time construction of bipartite Ramsey graphs with p much smaller than $\sqrt{n}$ is a folklore open problem. It is not known how to achieve p significantly smaller than $\sqrt{n}$ even if one allows quasi-polynomial time to construct the matrix. Therefore, Corollary 10.16 indicates that improving the rate in our construction from Proposition 10.14 to sub-quadratic in ε is likely to be quite difficult.[6]

There is, however, some hope to beat the "ε^2 barrier" without having to confront some major open problem in constructive Ramsey theory. For our coding applications we have always been thinking of ε as a small positive *constant* (independent of n), and hence we allow for a construction that runs in time exponential in $1/\varepsilon$ and/or a list size that is exponential in $1/\varepsilon$. It is an interesting open question whether this can be exploited to improve the rate beyond ε^2. Also, the results of the next section get a rate better than ε^2 by using a slightly larger alphabet size. They also critically exploit non-linearity. It will also be interesting to get a better than ε^2 rate for *binary* codes using some non-linear construction. Such a construction would escape the confines of the relation to bipartite Ramsey graphs, and thus may not be very hard to obtain (and our results of the next section provides some cause for optimism that this is indeed the case).

[6]On the other hand, optimistically speaking this provides further motivation to try and improve the ε^2 rate in Proposition 10.14.

10.7 Better results for larger alphabets using juxtaposed codes

We now proceed to highly erasure list-decodable codes over alphabets which are slightly larger than binary. We will use the larger alphabet size to get rates better than the ε^2 barrier we highlighted for binary linear codes in the previous section.

There are two main tools used in our construction. The first one is the use of pseudolinear codes (as guaranteed by Lemma 10.4) as inner codes instead of linear codes as in the previous section. The provably better bound on list size for pseudolinear codes compared to linear codes translates into some quantitative advantage for the erasure list-decodability of the concatenated code. The second tool is the use of symbol juxtaposition to combine together several binary codes into a single code over a larger alphabet. This uses a similar approach to and is based on the same intuition as the code constructions in Section 9.6, where we presented juxtaposed codes with good list decodability from errors.

10.7.1 Main theorem

Theorem 10.17 *For every $\varepsilon > 0$ and every integer $t \geq 1$, there exists a code family with the following properties:*

- *(i) (Rate and alphabet size) It has rate $\Omega(\varepsilon^{1+1/t}/(t^2\log(1/\varepsilon)))$ and is defined over an alphabet of size 2^t.*
- *(ii) (Construction complexity) A code of blocklength N can be constructed in $N^{O(\varepsilon^{-1}\log(1/\varepsilon))}$ time deterministically and $O(\log^2 N\log(1/\varepsilon)/\varepsilon^{2+1/t})$ time probabilistically.*
- *(iii) (Erasure list-decodability) A code of blocklength N in the family can be list decoded from up to a fraction $(1-\varepsilon)$ of erasures using lists of size $O(t\log(1/\varepsilon))$ in $N^{O(1/\varepsilon)}$ time.*

Proof (Sketch): The construction and proof are very similar to those in Section 9.6.2 (specifically Theorem 9.19) from the previous chapter where we proved a similar result for the the errors case. Familiarity with the contents of Section 9.6.2 will be helpful to understand what follows.

Let $\delta_0, \delta_1, \ldots, \delta_t$ be a sequence in geometric progression with $\delta_0 = \varepsilon/2$, $\delta_t = 1$, and $\delta_i/\delta_{i-1} = \Delta$ for $1 \leq i \leq t$. Note that this implies $\Delta = (2/\varepsilon)^{1/t}$.

Let m be a large enough integer and let $n_0 = 2^m$. Our code C^* will be the juxtaposition of t codes $\mathbf{C_i}$, each of which is the concatenation of an $[n_0, k_i]_{2^{m_i}}$ Reed-Solomon code C_i^{RS} with an $(n_1, m_i)_2$ pseudolinear code C_i^{PL}, where $m_i = m\delta_i/\delta_0$ and

$$k_i = \Theta\Big(\frac{\varepsilon \cdot n_0}{t\delta_i\log(1/\varepsilon)}\Big) .$$

The pseudolinear code C_i^{PL} will have the properties guaranteed by Lemma 10.4 with the setting $\sigma = \delta_{i-1}$. Hence it will have rate $r_i = m_i/n_1 = \Omega(\delta_{i-1})$. Note that each $\mathbf{C_i}$ is a binary code of blocklength $N \stackrel{\mathrm{def}}{=} n_0 n_1$ and dimension

$$k_i m_i = \Omega\Big(\frac{\varepsilon n_0}{t\delta_i\log(1/\varepsilon)} \cdot \delta_{i-1}n_1\Big) = \Omega\Big(\frac{\varepsilon N}{t\Delta\log(1/\varepsilon)}\Big) .$$

This is independent of i making it possible for the codes $\mathbf{C_i}$ to be juxtaposed together to give C^*. The rate of C^* is $1/t$ times the rate of each individual $\mathbf{C_i}$, and thus

$$R(C^*) = \Omega\Big(\frac{\varepsilon}{t^2\Delta\log(1/\varepsilon)}\Big) = \Omega\Big(\frac{\varepsilon^{1+1/t}}{t^2\log(1/\varepsilon)}\Big) .$$

Being the juxtaposition of t binary codes, C^* is a code over an alphabet of size 2^t. This verifies Property (i) claimed in the theorem.

The dominant component in the construction of C^* is the construction of the inner codes $C_i^{\rm PL}$ used in the concatenated codes $\mathbf{C_i}$. By Lemma 10.4, each $C_i^{\rm PL}$ can be constructed in $2^{O(m_i \log(1/\delta_{i-1}))} = 2^{O(m\varepsilon^{-1}\log(1/\varepsilon))}$ time deterministically (since $m_i = m\delta_i/\varepsilon = O(m\varepsilon^{-1})$. It can also be constructed in $O(m_i^2 \log(1/\varepsilon)/\delta_{i-1}) = O(m^2\varepsilon^{-2}\Delta\log(1/\varepsilon))$ time probabilistically. Since the overall blocklength $N = n_0 n_1 = 2^m n_1$, we have $m \le \lg N$. Therefore the constructions times are $N^{O(\varepsilon^{-1}\log(1/\varepsilon))}$ for a deterministic construction, and $O(\log^2 N \log(1/\varepsilon)/\varepsilon^{(2+1/t)})$ for a probabilistic construction (that works with high probability). This proves Property (ii) claimed in the theorem.

It remains to prove the erasure list-decodability property of C^*. Given a received word $\mathbf{r}$ with symbols at a $(1-\varepsilon)N$ positions erased, we wish to find all codewords $\mathbf{c} \in C^*$ that agree with $\mathbf{r}$ in the unerased positions. Suppose that $\mathbf{c} = C^*(\mathbf{x})$ is any such codeword. Both $\mathbf{r}$ and $\mathbf{c}$ can be broken up into n_0 blocks of n_1 symbols each, corresponding to the n_0 inner encodings at the n_0 positions of the outer Reed-Solomon codes. By the same "bucketing" argument that we used in the proof of Theorem 9.19, we know that there exists *some* i^*, $1 \le i^* \le t$ for which the following holds: there exists a subset B of the n_0 blocks, with $|B| \ge n' \stackrel{\rm def}{=} \frac{\varepsilon n_0}{2t\delta_{i^*}}$, such that for every block in B at least a fraction δ_{i^*-1} of positions within that block are *not* erased in $\mathbf{r}$.

Pick an arbitrary subset B' of B with $|B'| = n'$ and consider the list decoding of each of the n' inner codes corresponding to the blocks in B' from the at most $(1-\delta_{i^*-1})$ fraction of erasures. (Here we focus attention on and use only the i^*'th symbol from each of the N "juxtaposed" symbols from the received word $\mathbf{r}$.) Each inner decoding can be accomplished by a brute-force search over all codewords in $2^{m_i} \le 2^{m/\varepsilon} = O(n_0^{1/\varepsilon})$ time, and in fact this is the dominant component of the decoding time. By the property of $C_{i^*}^{\rm PL}$, each of the n' inner decodings only outputs a list $L_j^{(i^*)}$ of $O(\log(1/\delta_{i^*-1})) = O(\log(1/\varepsilon))$ codewords (or in other words Reed-Solomon symbols for the code $C_{i^*}^{\rm RS}$), for each of the blocks $j \in B'$. By our choice of i^*, each of these lists must have the "correct" symbol of $C_{i^*}^{\rm RS}(\mathbf{x})$.

To finish the decoding, it suffices to be able to list decode $C_{i^*}^{\rm RS}$ with these lists $L_j^{(i^*)}$, $j \in B'$, as input and find *all* messages $\mathbf{x}$ such that $L_j^{(i^*)}$ contains the j'th symbol of $C_{i^*}^{\rm RS}(\mathbf{x})$ for *every* $j \in B'$. This is exactly the setup of the list decoding from uncertain receptions we considered in Chapter 6 (specifically, Theorems 6.15 and 6.17). Since the rate of $C_{i^*}^{\rm RS}$ is picked to be $O(\varepsilon/(t\delta_{i^*}\log(1/\varepsilon)))$, it follows that one can find all such $\mathbf{x}$ in near-quadratic time by running the algorithm of Theorem 6.17 (with $k = k_{i^*} - 1$, $n = n'$, $\ell = O(\log(1/\varepsilon))$, and $\alpha = 1$). Also the number of solutions output will be at most $O(\sqrt{n\ell/k})$, which is $O(\log(1/\varepsilon))$ for our choice of parameters. The $O(n_0^{1/\varepsilon})$ time required to decode the inner codes thus dominates the runtime of the algorithm.

Of course, the algorithm cannot know the value of i^* in the above description. But, one can run the above decoding procedure for each $\mathbf{C_i}$, $1 \le i \le t$, and then output the union of the lists output by each of the decodings. This will give us a list of size at most $O(t\log(1/\varepsilon))$ that includes *all* codewords that agree with the received word $\mathbf{r}$ in every unerased position. This completes the proof of Property (iii) claimed in the theorem as well. □

10.7.2 Discussion of some consequences

Theorem 10.17 with the choice $t = 1$ gives binary codes with the same rate as Theorem 10.12, except that the construction is no longer linear, and the list size is much better (namely, $O(\log(1/\varepsilon))$ instead of $O(1/\varepsilon)$). Applying Theorem 10.17 with $t = \lceil \lg(1/\varepsilon) \rceil$, we get the following corollary:

Corollary 10.18 *For every $\varepsilon > 0$, there exists a polynomial time constructible code family of of rate $\Omega(\varepsilon \log^{-3}(1/\varepsilon))$ over an alphabet of size $O(1/\varepsilon)$ such that each code in the family can be list decoded from up to a fraction $(1-\varepsilon)$ of erasures in polynomial time using lists of size $O(\log^2(1/\varepsilon))$.*

Note that the rate is off only by a $\log^{O(1)}(1/\varepsilon)$ factor from the *optimal* $\Omega(\varepsilon)$ rate (which is achieved by random codes, cf. Lemma 10.4 and Theorem 10.11). However, while the random construction works even over the binary case, we need larger and larger alphabet sizes in order to approach the optimal $\Omega(\varepsilon)$ rate.

AG-codes which achieve the best known trade-offs between rate and minimum distance provide constructions of highly erasure list decodable codes of the optimal rate $\Omega(\varepsilon)$, but they require an alphabet size of $O(1/\varepsilon^2)$. The constructions implied by Theorem 10.17 are incomparable to AG-codes since they achieve a better alphabet size at the expense of a moderate worsening of the rate. Also, the list size achieved by them is $O(\log(1/\varepsilon))$ when t is a constant independent of ε, where as AG-codes being linear necessarily require a list size of $\Omega(1/\varepsilon)$ in order to list decode a fraction $(1-\varepsilon)$ of erasures.

Thus, the results of this section give polynomial time constructions of codes that achieve some new, interesting trade-offs between rate and erasure list-decodability.

10.7.3 Improving the decoding time in Theorem 10.17

One of the drawbacks of the result of Theorem 10.17 is that the decoding time is quite high (namely, $N^{O(1/\varepsilon)}$). Incidentally, this was also the case for the result for errors from Theorem 9.19, though we did not point it out explicitly then! The high decoding time resulted from the brute-force decoding of the inner codes. For example, the inner code C_t^{PL} had a dimension of $m\delta_t/\delta_0 = \Omega(m/\varepsilon)$, and decoding it by searching over all codewords requires $2^{O(m/\varepsilon)} = N^{O(1/\varepsilon)}$ time. To reduce the decoding time for each inner code C_i^{PL}, we further juxtapose it with a linear code of C_i^{lin} of the same dimension and blocklength. This is similar to the approach taken in Lemma 9.11 of Chapter 9. We now briefly review that technique when applied to the setting of erasure codes. C_i^{lin} will be a binary linear code of rate $\Omega(\delta_{i-1})$ which is $((1-\delta_{i-1})n_1, O(1/\delta_{i-1}^2))$-list decodable, as guaranteed by Lemma 10.7, Part (i). The necessary linear codes C_i^{lin} can be constructed within the time bounds required to construct the codes C_i^{PL} (for both probabilistic and deterministic constructions — for deterministic constructions we can use the derandomization procedure discussed in Section 10.6.3). Hence the asymptotic construction time of the codes is not altered by the addition of the linear component to the inner codes.

The inner decoding now uses the linear component of the received word to first perform erasure decoding of the codes C_i^{lin}. Owing to the linearity, this can be accomplished by solving a linear system in $O(n_1^3)$ time. By the erasure list-decodability property of C_i^{lin}, this step will return $L_i \le O(1/\delta_{i-1}^2)$ messages. It now suffices to check which subset of these L_i messages are consistent under encoding by C_i^{PL} with the pseudolinear component of the received word. This task can of course be done in time polynomial in n_1.

Thus, using this trick all the inner decodings can be performed in $O(n_0 \mathrm{poly}(n_1)) = O(N \log^{O(1)} N)$ time. Since the outer Reed-Solomon decoding, as per the bounds stated in Theorem 6.17, takes at most $O(n_0^2 \log^3 n_0 \varepsilon^{-O(1)})$ time, the overall decoding time is now $O(N^2 \varepsilon^{-O(1)} \log N)$ (since $n_0 \le N/\log N$). The juxtaposition with the linear code reduces the rate by a factor of 2 and also squares the alphabet size. We omit further details, and below we simply state the final result that can be obtained after applying this modification.

Theorem 10.19 *For every $\varepsilon > 0$ and every integer $t \ge 1$, there exists a code family with the following properties:*

(i) (Rate and alphabet size) It has rate $\Omega(\varepsilon^{1+1/t}/(t^2 \log(1/\varepsilon)))$ and is defined over an alphabet of size 2^{2t}.

(ii) (Construction complexity) A code of blocklength N can be constructed in $N^{O(\varepsilon^{-1}\log(1/\varepsilon))}$ time deterministically and in $O(\log^2 N \log(1/\varepsilon)/\varepsilon^{2+1/t})$ time probabilistically.

(iii) (Erasure list-decodability) A code of blocklength N in the family can be list decoded from up to a fraction $(1-\varepsilon)$ of erasures using lists of size $O(t\log(1/\varepsilon))$ in $O(N^2\varepsilon^{-O(1)} \log N)$ time.

10.8 Concluding Remarks

Our lower bound on $\tilde{R}_L(p)$ from Theorem 10.6 guarantees the *existence* of binary linear code families of rate $\Omega(\varepsilon)$ which can be list decoded from up to a fraction $(1-\varepsilon)$ of erasures, using lists of size, say $O(\varepsilon^{-2})$. The construction of Theorem 10.12, however, only achieves a rate of about ε^2. Now the result of Theorem 10.11 implies that for linear code families of positive rate, one requires a list size of at least $\Omega(1/\varepsilon)$ to list decode from $(1-\varepsilon)$ erasures. This implies that our concatenated code constructions from Section 10.6 cannot be improved by the choice of a better linear code as inner code. Moreover, the connection to bipartite Ramsey graphs from Section 10.6.4 indicates that either the list size or the construction time will probably be exponential in $1/\varepsilon$ if the rate of the code is asymptotically better than ε^2. In this sense, the rate in our result of Proposition 10.14 might be hard to improve without worsening some of the other parameters.

In Section 10.7, by resorting to pseudolinear codes at the inner level, together with the technique of juxtaposed code constructions, we were able to closely approach the optimal rate of $\Omega(\varepsilon)$ by allowing a gradual increase in the alphabet size. These results achieve a very small (about $O(\log(1/\varepsilon))$) list size for list decoding up to a fraction $(1-\varepsilon)$ of erasures. The fact that this is impossible to achieve with linear codes (Theorem 10.11) is also one of the surprising results of this chapter.

Below we list some open questions relating to the contents of this chapter.

Open Question 10.1 *Is there a* poly$(n, 1/\varepsilon)$ *time construction of* binary linear *codes with the properties guaranteed in Lemma 10.13, namely codes which have rate close to $\Omega(\varepsilon^2)$ and which are efficiently list decodable up to a fraction $(1-\varepsilon)$ of erasures?*

Open Question 10.2 *Is there a polynomial time construction of* binary *codes of rate $\Omega(\varepsilon^{2-a})$ for some $a > 0$ which are efficiently list decodable up to a fraction $(1-\varepsilon)$ of erasures? More ambitiously, can one construct such codes with close to the optimal $\Omega(\varepsilon)$ rate? (We allow for the construction time to depend exponentially on $1/\varepsilon$, and the codes to be non-linear provided they have efficient encoding and decoding algorithms.)*

Open Question 10.3 *Is there a polynomial time construction of q-ary codes that have (the optimal) $\Omega(\varepsilon)$ rate and which are efficiently list decodable up to a fraction $(1-\varepsilon)$ of erasures, for some q that is a fixed constant independent of ε? (Certain constructions of AG-codes achieve the optimal $\Omega(\varepsilon)$ rate, but require an alphabet size of $O(1/\varepsilon^2)$. Our juxtaposed codes achieve a rate of about $\Omega(\varepsilon^{1+1/\lg q})$ and thus approach the optimal $\Omega(\varepsilon)$ rate for large q.)*

10.9 Bibliographic Notes

The concept of erasure list-decodability seems to have been studied explicitly for the first time in [GS00]. Their motivation for studying list decoding under erasures in turn came from [GHLP99, KS99], where the role of such codes in results about the hardness of partial computation of NP-witnesses was highlighted (more on this in Section 12.2.5 of Chapter 12). The relation between minimum distance and erasure list-decodability (Proposition 10.1) is folklore and is also implicit in [GS00].

The notion of erasure list decoding radius and the rate function $\tilde{R}_L(p)$ were first introduced by the author in [Gur01b]. The lower bound on $\tilde{R}_L^{\mathrm{lin}}(p)$ (Theorem 10.6) and the upper bound on $\tilde{R}_L(p)$ (Theorem 10.8) also appear in [Gur01b]. The specific use of the probabilistic method in the proof of Lemma 10.7 was inspired by a similar proof in [GHLP99].

The lower bound on $\tilde{R}_L(p)$ (Theorem 10.3) as well as the "improved" upper bound on $\tilde{R}_L^{\mathrm{lin}}(p)$ (Lemma 10.10), which in particular implies a provable separation between the performance of linear and non-linear codes with respect to erasure list-decodability, appear for the first time in this thesis. They will be submitted as part of an upcoming paper [GI01a].

The concatenated code construction from Section 10.6 appears in [Gur01b]. The interesting connection to Ramsey graphs, which indicated the difficulty of improving our result for binary linear codes, was communicated to us by Noga Alon [Alo99]. The improvements in rate by resorting to pseudolinear (as opposed to linear) inner codes, and the idea of code juxtaposition from Section 10.7, appear in published form for the first time in this thesis. They will be also be part of the upcoming submission [GI01a].

Part III

Applications

INTERLUDE

The first two parts of the thesis have explored the combinatorial and algorithmic aspects of list decoding in detail and have presented list decoding algorithms that correct a large fraction of errors for certain classical codes as well as some novel code constructions.

In addition to their inherent interest to the subject of list decoding, some of the results and techniques developed so far have also found numerous applications outside the immediate domain of list decoding and even coding theory. The next couple of chapters provide a glimpse of some of these applications.

The next chapter will present an application of the techniques used in Chapter 9 to the construction of binary linear codes which can be uniquely decoded up to the maximum possible fraction $(1/4 - \varepsilon)$ of errors. The highlight of the construction will be the extremely fast encoding and decoding algorithms that we will achieve. In fact, we will achieve codes that are both encodable and decodable in linear time.

Chapter 12 discusses applications of list decoding to problems outside coding theory. This will include a brief discussion of and pointers to the reasonably large number of complexity-theoretic applications of list decoding, as well as some cryptographic applications and an interesting algorithmic application called "Guessing Secrets".

After presenting these applications, we will conclude the thesis with some closing remarks and a list of open problems.

Chapter 11

Linear-time codes for unique decoding

Warning: *This chapter contains material of an explicit nature.*

11.1 Context

The goal of this chapter is also to construct codes which can be decoded from a large, and essentially up to a "maximum" possible, fraction of errors. The difference is that we are now interested in unique decoding as opposed to list decoding. Nevertheless we will now be able to use some of the results and techniques from the previous chapters on list decodable codes to construct good, uniquely decodable codes.

For unique decoding, the maximum number of errors that can be corrected is limited by half the minimum distance of the code. In particular, it is never possible to correct more than a fraction $1/2$ of errors (while list decoding allowed us to correct up to a fraction $(1-\varepsilon)$ of errors, for any desired $\varepsilon > 0$). In this chapter, we will be interested in both binary codes and codes over large alphabets. Since binary code families of positive rate have relative distance less than $1/2$,[1] the half-the-minimum-distance barrier implies that the maximum possible fraction of errors that can be uniquely decoded is $(1/4 - \varepsilon)$ for binary codes. For codes over large alphabets, the maximum unique decoding radius is $(1/2 - \varepsilon)$ (this requires an alphabet size of $\Omega(1/\varepsilon)$, though).

Therefore, we are interested in codes over large alphabets that are uniquely decodable up to a fraction $(1/2 - \varepsilon)$ of errors, and in binary codes that are uniquely decodable up to a fraction $(1/4 - \varepsilon)$ of errors. These represent the "high-noise regime" for unique decoding. For such code constructions, our goal is to achieve good rate together with highly efficient encoding and decoding algorithms.

Due to the half-the-distance limitation of unique decoding, the relative distance of such uniquely decodable codes must be at least $(1 - O(\varepsilon))$ for large alphabet codes and $(1/2 - O(\varepsilon))$ for binary codes. The best known explicit constructions of codes with such a large relative distance achieve a rate of $\Omega(\varepsilon)$ and $\Omega(\varepsilon^3)$ respectively. However, such codes either used algebraic-geometric codes [She93] and hence suffered from complicated decoding procedures (in particular the decoding time was at least cubic in the blocklength), or as in the constructions of [ABN$^+$92], were explicitly constructible but lacked a polynomial time

[1] This is a well-known bound in coding theory that follows for example from the "Plotkin bound", cf. [vL99, Section 5.2]

algorithm to decode up to half the minimum distance (or for that matter to decode up to any positive constant, no matter how small, fraction of errors). In short, codes of good rate with a "maximum" relative distance (i.e., $(1-\varepsilon)$ for codes over large alphabet, and $(1/2-\varepsilon)$ for binary codes) were not known together with very efficient decoding procedures, for both the list and unique decoding versions.

In this chapter, we present constructions of large distance codes with the above rates (i.e., match the best known rates), together with *near-linear* time (i.e., $O(n^{1+\beta})$ time for any desired $\beta > 0$) list decoding algorithms to decode well beyond half the distance. We then focus on unique decoding of such large distance codes up to close to half the distance. The goal here is to obtain better runtimes than for the list decoding case, and in particular to obtain the optimal linear decoding time.[2] We succeed in this goal, and present novel constructions of large distance codes of the above "best known" rates, together with *linear time* unique decoding algorithms, as well as linear time encoding algorithms. Specifically, we can construct codes over a constant-sized alphabet that have the optimal (up to constant factors) $\Omega(\varepsilon)$ rate and which are uniquely decodable from a fraction $(1/2-\varepsilon)$ of errors in linear time and which are further encodable in linear time. We also construct similar linear-time binary codes that can correct a fraction $(1/4-\varepsilon)$ of errors and have $\Omega(\varepsilon^3)$ rate (which matches the best known rate for constructive codes of relative distance $(1/2 - O(\varepsilon))$).

These are the first linear-time codes which are uniquely decodable up to the maximum possible radius. The result for binary codes dramatically improves the fraction of errors corrected by the *only other* construction of linear-time codes known prior to our work, namely the codes due to Spielman [Spi96]. The large alphabet result is also striking since it matches the optimal performance (in terms of rate) achieved by AG-codes, but comes equipped with linear time encoding and decoding algorithms!

Though the unique decoding algorithms for large distance codes form the crux of this chapter, we also discuss the list decoding results in this chapter since the techniques used are similar to our unique decoding results. All our constructions share the common thread of using expander-like graphs as a component, and there is a strong overlap in techniques between this chapter and portions of Chapter 9 (specifically, Section 9.4). The expander graphs enable the design of efficient decoding algorithms through various forms of voting procedures. The result on near-linear time list decodable codes also uses the idea of multi-concatenated codes from Chapter 9. Nevertheless, except for this one dependence on multi-concatenated codes, the presentation in this chapter should be reasonably self-contained and should allow the reader to read and appreciate the chapter on its own. Also, we should mention that our unique decoding results can be read independently of the list decoding results (and in fact we would recommend this for a first reading of this chapter).

We elaborate on the results of this chapter and compare them with previously known bounds in the next section. Most of the material in this chapter appears in the paper [GI01b], except for the results in Section 11.6 which appear for the first time in this thesis, and will be part of an upcoming submission [GI01a].

11.2 Main Results

We first study the problem of constructing efficiently *list decodable* codes of good rate and "maximum" possible relative distance. The parameters of these constructions are tabulated

[2]The goal of linear time list decoding algorithms for any non-trivial code family seems very difficult and out of reach at this point.

in Figure 11-1.

No	Alphabet	Relative distance	LDR	Rate	Encoding time	Decoding time	Const. time (probabilistic)
1a	$2^{1/\varepsilon}$	$1-\varepsilon$	$\alpha < 1$	ε	$n \log^{O(1)} n$	$n^{1+\beta}$	explicit
1b	2	$1/2-\varepsilon$	$\eta < 1/2$	ε^3	$n \log^{O(1)} n$	$n^{1+\beta} + 2^{1/\varepsilon^3}$	$1/\varepsilon^3$

Figure 11-1: The parameters of our near-linear time list decodable codes. n stands for the length of the code. For readability, the $O(\cdot)$ and $\Omega(\cdot)$ notation have been omitted. The value of β is in the interval $(0,1]$; its value influences the rate by a constant factor (which admittedly depends rather badly on β). The LDR (list decoding radius) shows the fraction of errors which the list decoding algorithms can correct.

We now elaborate on the performance achieved by these constructions. The first construction (1a) presents explicit codes with relative distance $(1-\varepsilon)$ and an optimal (up to constant factors) rate of $\Omega(\varepsilon)$, together with a *near-linear time* (i.e., $O(n^{1+\beta})$ time for any desired $\beta > 0$) list decoding algorithms to correct a fraction α of errors, for any desired constant $\alpha < 1$ (the constant in front of the ε in the rate depends on α, β). Concatenation of these codes with a constant-sized large distance binary code gives a constructive family of binary codes of relative distance $(1/2-\varepsilon)$ and rate $\Omega(\varepsilon^3)$, which is further list decodable up to a fraction η of errors in near-linear time for any desired constant $\eta < 1/2$ (this is our construction (1b)). These constructions match the best known rate for constructive families of codes with such a large relative distance. In addition, we are able to give very efficient list decoding procedures for such codes. Thus they provide a more efficient alternative to codes based on concatenation schemes involving AG-codes, which achieve a similar rate, but only have complicated construction algorithms and high decoding complexity (the best known runtime is cubic in the blocklength, even if we are only interested in unique decoding, cf. [FR93]). We also mention that our codes are very similar to the codes constructed in [ABN+92], where constructions with similar rate and relative distance are given. However, the constructions in [ABN+92] lacked a polynomial time decoding algorithm to (unique) decode up to any constant fraction of errors, let alone list decode beyond half the distance. In fact, the authors of [ABN+92] pose the question of whether there exists a polynomial time decoding algorithm for their codes as an open problem. Our results can be viewed as a positive resolution of the main open problem from their paper.

We next discuss our results for unique decoding. The results are tabulated in Figure 11-2. We actually obtain two classes of constructions, each with constructions of binary and large alphabet codes. The constructions (3a,3b) essentially subsume (2a,2b) (except for the slightly larger alphabet size in (3a) compared to construction (2a)). However, the constructions (2a,2b) are simpler to describe and nicely illustrate the main idea behind our linear-time code constructions, and therefore we include a discussion of them in this chapter (we also recommend reading them first before reading about constructions (3a,3b)). Note also that as far as unique decoding is concerned, the constructions (3a,3b) also subsume the constructions (1a,1b).

No	Alphabet	Decoding radius	Rate	Encoding time	Decoding time	Construction time (probabilistic)
2a	$2^{1/\varepsilon^2}$	$1/2-\varepsilon$	ε^2	n/ε^2	n/ε^2	explicit
2b	2	$1/4-\varepsilon$	ε^4	$n+2^{1/\varepsilon^2}$	$n/\varepsilon^2+2^{1/\varepsilon^4}$	$1/\varepsilon^4$
3a	$2^{\varepsilon^{-2}\log(1/\varepsilon)}$	$1/2-\varepsilon$	ε	n/ε^2	n/ε^2	explicit
3b	2	$1/4-\varepsilon$	ε^3	$n+2^{1/\varepsilon^2}$	$n/\varepsilon^2+2^{1/\varepsilon^4}$	$\log(1/\varepsilon)/\varepsilon^4$

Figure 11-2: The parameters of our linear-time codes for unique decoding. n stands for the length of the code. For readability, the $O(\cdot)$ and $\Omega(\cdot)$ notation have been omitted, and $\log(1/\varepsilon)$ factors have been omitted in the encoding and decoding times. The decoding radius shows the fraction of errors which the unique decoding algorithms can correct.

Note that both classes of uniquely decodable codes come with linear-time encoding and decoding algorithms (we call such codes *linear-time codes*). We use the same approach as in constructions (1a,1b) to obtain constructions (2a,2b). The rate of these codes is an ε factor worse compared to the best known rate for constructive codes with the necessary relative distance. However, these codes are simple to describe, and can correct a fraction of errors arbitrarily close to the maximum possible (i.e., $1/4$ for binary codes and $1/2$ for large alphabet codes) in *linear* time. In fact, by decoding up to the maximum possible radius, these codes exhibit a *dramatic* improvement in error-correction capability over that of the only prior linear-time binary codes (due to Spielman [Spi96, Spi95]), or for that matter over that of the linear-time decodable binary codes of Sipser and Spielman [SS96], Zémor [Z0́1], and Barg and Zémor [BZ01]. To make a concrete comparison, the largest fraction of errors corrected by the linear-time codes in [Spi96] (at least the bound directly deducible from the paper) seems to be about 4.6×10^{-7}. The linear-time decodable codes from [Z0́1] can correct about a fraction 3.1×10^{-3} of errors with positive rate. In comparison, we can correct a fraction $(1/4-\varepsilon)$ of errors for arbitrary $\varepsilon>0$.

Using a slightly different construction, we get the linear-time codes (3a,3b), which can also correct a fraction of errors arbitrarily close to the maximum possible, and in addition achieve rates of $\Omega(\varepsilon)$ and $\Omega(\varepsilon^3)$ for codes over a large alphabet and binary alphabet respectively (thereby improving the constructions (2a,2b)). We stress again that these are the *best known rates* for codes of relative distance $(1-\varepsilon)$ and binary codes of relative distance $(1/2-\varepsilon)$ respectively, and are we are able to achieve these rates *together with linear time encoding and decoding procedures.*

A Remark on Construction Times. Our constructions (1a,2a,3a) of codes over a large alphabet are *explicit.*[3] Our binary codes (1b,2b,3b) are obtained by concatenating the codes (1a,2a,3a) with suitable binary codes. These binary codes can either be picked randomly or found deterministically by a constant amount of brute-force search. If one considers codes with only a constant amount of "non-constructivity" as still being "explicit", then our codes (1b,2b,3b) are also explicit. Moreover, we can make the construction completely explicit at the expense of a (moderate) worsening of the rate, by picking some explicit binary code for the final concatenation instead of the "best" non-explicit ones. We point out here that the linear-time codes of Spielman [Spi96] also require a constant amount of non-constructivity,

[3]We do not try to formally define the notion of a construction being "explicit". We refer to a construction as explicit only if it is "constructive" in the strictest sense of the term, namely if it is specifiable by a formula of an acceptable type.

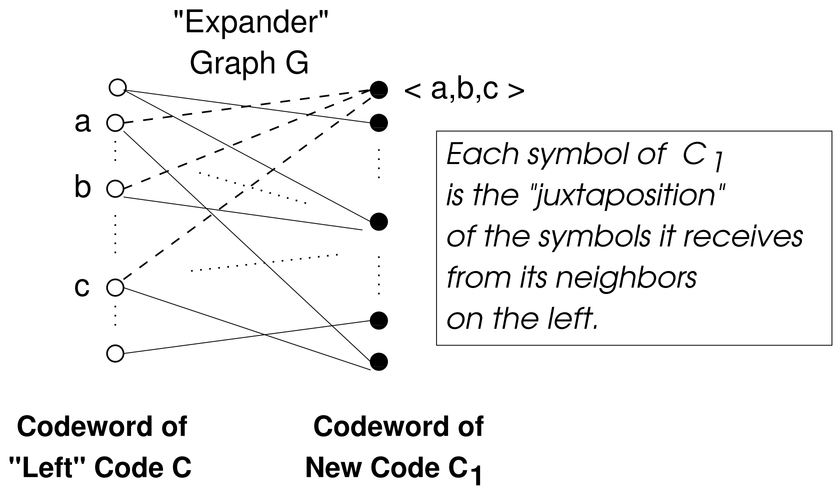

Figure 11-3: Basic structure of our code constructions. For binary codes, each symbol of C_1 is further concatenated with a good, constant-sized binary code.

which can eliminated at the expense of worsening the parameters (see the discussion in [Spi96, Sec. 5, Remark 20]).

Comment on meaning of "linear time": When we claim that our encoding and decoding algorithms run in linear time, we must be careful to specify exactly what is meant, since the meaning of "linear time" depends on the model of sequential computation considered. We do not get into a further discussion of this aspect here, but instead point the interested reader to the excellent discussion at the beginning of the papers [SS96, Spi96]. We just remark that our codes use the linear-time codes due to Spielman [Spi96] as a black-box, and use expanders on top of it to boost the number of errors tolerated by the code. Our algorithms run in linear time in *any* model under which the algorithms of Spielman run in linear time. In particular, they can be implemented to run in linear time on a RAM in the uniform cost model, or on a Pointer Machine (see [SS96, Sec. 2] for details and pointers on these models).

11.3 Techniques

At a high level, our codes are all constructed using the same scheme (which in turn is similar to the scheme used in the expander-based constructions of Chapter 9). The basic components of the constructions are: a "left" code (say, C), an expander-like graph G, and (in case of binary codes) an efficient "right" binary code (say C').

Given the above components, the codes are constructed as follows. For each codeword x of C, we construct a new codeword y by distributing the symbols of x from left to right according to the edges in G. The juxtaposition of symbols "sent" to each right node of G forms a symbol of the codeword y of the final large alphabet code C_1. This is illustrated in Figure 11-3. If we want to construct a binary code, we add a final level of concatenation with the binary code C'.

The specifics of the implementation of the above ideas depend on the actual code construction. For the constructions (1a,1b), we use for left code a list decodable code that can correct an α fraction of errors for an appropriate constant $\alpha < 1$ — such a code of constant rate and with a near-linear time list decoding algorithm can be built using a concatenation of several levels of Reed-Solomon codes followed by any explicitly specified large distance code.

Our codes (2a,2b), as well as (3a,3b), use the linear-time encodable/decodable codes of Spielman [Spi96] as the left code. The properties of the expander graph G allows us to transform a heavily corrupted codeword y of C_1 to a much less corrupted codeword x of C. The latter can be decoded using the linear-time decoding algorithm for C. The encoding can also be accomplished in linear time using the linear-time encoder for C.

Our constructions are similar to those in [ABN$^+$92] where expanders were used to "amplify" the distance and thus construct codes of large minimum distance. However, while the results of [ABN$^+$92] as well as our results in Chapter 9 use only the vertex-expansion properties of the graph G, here we use much stronger *isoperimetric* properties offered by expander graphs. We need these stronger properties because we are also interested in good decoding algorithms for our codes. Informally, the isoperimetric property just states that the fraction of edges between two sufficiently large subsets of vertices is approximately the product of their densities. The isoperimetric property of expanders is implied by the spectral definition of expanders in terms of the second largest eigenvalue of the adjacency matrix. It is this definition of "expanders" that we use in this chapter.

11.4 Near-linear time list decodable codes of large distance

In this section we present our constructions (1a,1b). The codes come in two versions: binary and over a large alphabet. The latter ones have relative designed distance $(1-\varepsilon)$, rate $\Omega(\varepsilon)$ and alphabet size $2^{O(1/\varepsilon)}$. The binary codes have relative designed distance $(1/2-\varepsilon)$ and rate $\Omega(\varepsilon^3)$. In both cases, we will present an algorithm to list decode up to any desired radius that is less than the maximum one could hope for. Specifically, for any fixed $\alpha < 1$ (resp. $\eta < 1/2$), our codes over a large alphabet (resp. binary codes) will be list decodable up to a fraction α (resp. η) of errors. A straightforward implementation of our idea, i.e., using a "simple" left code, will lead to near-quadratic list decoding times for our constructions. By complicating the choice of the left code C, we can obtain a decoding time of $O(n^{1+\beta})$ for any desired constant $\beta > 0$.

11.4.1 Combinatorial objects necessary for the constructions

We start the description from the codes over large alphabet that are uniquely decodable up to a fraction $(1/2-\varepsilon)$ of errors. We will then concatenate this code with a suitable inner code to get the desired binary codes. We need the following combinatorial objects for our construction:

(1) For any $\alpha, \beta > 0$, an $(n,k)_q$ code C with minimum distance at least $n/2$ and constant rate $r_{\alpha,\beta} > 0$ such that C is list decodable in $O(n^{1+\beta})$ time from up to a fraction $(1-\alpha)$ of errors. (Here q, r are constants that depend only on α, β and *not* on n or ε.)

(2) A bipartite regular graph $G = (A, B, E)$ with $|A| = |B| = n$ and degree Δ, such that for parameters $(\varepsilon, \delta, \delta')$ where $\varepsilon \le \delta' \le \delta$, G has the following properties:

(i) For $X \subset A$, $|X| = n/2$, we have $|\Gamma(X)| > (1 - \varepsilon)|B|$ where $\Gamma(X) \subseteq B$ is the neighborhood of X (i.e., the graph has good *vertex-expansion* and is in particular an $(n, \Delta, 1/2, \varepsilon)$-expander, in the terminology of Chapter 9, Definition 9.5).

(ii) For $X \subset A$, $|X| = n/2$, and $Y \subseteq B$, $|Y| \geq \delta|B|$, we have $\frac{|E(X:Y)|}{\Delta|X|} \geq \delta'$ (here $E(X : Y)$ denotes the set of edges in the $X : Y$ cut).

We next specify how the above objects can be constructed.

Explicit construction of the list decodable code C

It is not difficult to see that an explicit code construction C satisfying the above requirements exists together with a near-quadratic time algorithm to list decode up to a fraction $(1 - \alpha)$ of errors.

For example, one can use constructions similar to those in Chapter 8 which concatenate an outer Reed-Solomon code with an inner code over a large, constant-sized alphabet. The inner code can be picked to be any explicitly constructible code of large minimum distance. Appealing to the Johnson bounds, the inner code will have large list-of-ℓ decoding radius for some fixed constant ℓ. The inner codes can be (list) decoded by a brute-force search for all codewords within a certain radius (this will take quadratic time to perform for all inner codes put together). The lists of size at most ℓ obtained from the list decoding of the inner codes can be passed to an outer decoder. The outer decoder will be a (near-quadratic time) list recovering algorithm for Reed-Solomon codes that finds all Reed-Solomon codewords which agree with an element of the list passed by the inner decoder for at least a certain fraction of the positions. Note that unlike the results presented in Chapter 8, where our goal was to maximize the fraction of errors corrected for a fixed code and hence we used the soft list decoding algorithms for Reed-Solomon codes, in the above we only need a list recovering algorithm for Reed-Solomon codes (Theorem 6.17). We omit the details, but picking a large enough alphabet size (that depends on α), this procedure can decode up to a fraction $(1-\alpha)$ of errors, and the rate of the concatenated code will be a positive constant that depends only on α.

To speed up the decoding time further and achieve near-linear decoding time as required by (1), we need a more complicated construction. The quadratic time in the previous algorithm comes from two sources. First there are $\Omega(n)$ inner codes, and a brute-force decoding of each of them takes $\Omega(n)$ time (where n is the blocklength of the Reed-Solomon code). Secondly, the Reed-Solomon decoding takes $\Omega(n^2)$ time. In order to reduce the decoding time, we need to reduce the blocklength of the Reed-Solomon code. This can be done by considering Reed-Solomon codes over a very large alphabet size (for example, we can have a blocklength of $O(\sqrt{n})$ using a field of size $2^{O(\sqrt{n})}$). The dependence of the runtime of the Reed-Solomon decoding algorithm (specifically the one from Theorem 6.17) on the field size is pretty good, and this will permit list decoding the Reed-Solomon code in near-linear time. However, the dimension of the inner code is no longer logarithmic, and in fact brute-force decoding of each inner decoding will take super-polynomial time, which is much worse than the quadratic time we began with! To tackle this problem, we do not apply the inner code immediately to the outermost Reed-Solomon code, but instead use a multi-concatenated scheme to gradually reduce the alphabet size, till it is finally about $\mathrm{poly}(\log n)$ (this is similar to the construction in Section 9.5). At this stage, the brute-force decoding of each inner code will in fact take only $\log^{O(1)} n$ time. Since we want each of the intermediate codes used in the multi-concatenated scheme to also be efficiently list decodable, we will use Reed-Solomon codes with appropriate parameters over fields of gradually decreasing size at the various levels of the concatenation.

The result thus obtained is formally stated below. We only sketch the proof since the techniques used are similar to those used in the proof of Lemma 9.15 from Chapter 9.

Lemma 11.1 *For every $\alpha > 0$ there exists a prime power $q = q_\alpha$, which may be assumed to be a power of two, such that for all $\beta > 0$ the following holds. There is an* explicitly specified *code family with constant rate $r_{\alpha,\beta} > 0$ and relative distance at least $1/2$ over an alphabet of size q, with the property that a code of blocklength N from the family can be list decoded from up to a fraction $(1-\alpha)$ of errors in $O(N^{1+\beta})$ time, and can be encoded in $O(N \log^{O(1)} N)$ time.*

Proof Sketch: Let $q = q_\alpha$ be a large enough power of two ($q = O(1/\alpha^2)$ will suffice, for example). The basic idea is to use a concatenated scheme where an outer code C_{out} is concatenated with a large distance q-ary inner code C_{in}. But in order to reduce the decoding time, instead of using as C_{out} a Reed-Solomon code, we use several Reed-Solomon codes over decreasing alphabet sizes in a multi-concatenated scheme similar to the one in Lemma 9.15. The change compared to Lemma 9.15 is that instead of using the (non-explicit) large alphabet pseudolinear codes guaranteed by Lemma 9.11 at each stage, we use (explicitly specified) Reed-Solomon codes with appropriate parameters. Since the innermost code C_{in} can be any explicit large distance q-ary code, the overall code construction is explicit.

We now sketch some details of the actual code construction. Define $p = \lceil 2/\beta \rceil$ and let n be any large enough integer of the form m^p, for some integer m. We now specify how a message will be encoded by our multi-concatenated code. Given a message $x \in \mathbb{F}_q^n$ where $n = m^p$, we first encode it using $(p-1)$ levels of Reed-Solomon codes, the i'th level being a code of dimension m and blocklength $a_i m$ over $\text{GF}(q^{m^{p-i}})$ for $1 \le i < p$ (the constant a_i will be chosen large enough so that the concerned Reed-Solomon code can be efficiently list decoded from a certain fraction of errors, and its exact value will depend upon α, p). It is easy to check that after these $(p-1)$ levels of encoding we will be left with $m^{p-1} \prod_{i-1}^{p-1} a_i = O(m^{p-1})$ symbols over $\text{GF}(q^m)$. We then encode each of these $O(m^{p-1})$ symbols of $\text{GF}(q^m)$ resulting from this $(p-1)$ stage encoding by a further two-levels of Reed-Solomon encoding, this time over fields of size $O(m)$ and $O(\log m)$ respectively. After all this, the encoding of x is comprised of at most $O(m^p \log m)$ strings of length $O(\log \log m)$ over $\text{GF}(q)$. Finally, each of these strings is further encoded by an arbitrary explicit q-ary linear code $\tilde{C}$ of dimension $O(\log \log m)$, positive rate and large enough minimum distance (as a function of α).[4]

The above gives an explicit specification of our final code, call it C^*. The distance of the code is at least the product of the component codes used in the multi-concatenated scheme. Therefore, by picking low enough rates, and hence large enough distances, for the Reed-Solomon codes and the innermost code $\tilde{C}$, we can ensure that the final code C^* has relative distance at least $1/2$, as required in the statement of the lemma.

We next sketch the list decoding procedure for C^*. The innermost code $\tilde{C}$ can be list decoded to the necessary radius in $\log^{O(1)} m$ time by a brute-force search over all the codewords. Since there are $O(m^p \log m)$ such decodings to be performed, all these can

[4]Explicit constructions of such $\tilde{C}$ are known by concatenating suitable algebraic codes with every code from a so-called "Wozencraft ensemble" of codes (most codes in which meet the Gilbert-Varshamov bound). See, for example, the discussion in the proofs of Theorem 8.8 and Proposition 8.10 from Chapter 8.

be done within $O(n \log^{O(1)} n)$ time (recall that $n = m^p$). For decoding the Reed-Solomon codes, we use the list recovering algorithm from Chapter 6 (specifically that of Theorem 6.17) which runs in $O(N^2 \log N \log Q \log^{O(1)} \log Q)$ time when decoding an $[N, K]_Q$ Reed-Solomon code. In this way, the cost of decoding each codeword at the i-th level, for $1 \le i < p$ is $O(m^2 m^{p-i} \log^{O(1)} n)$ (as the blocklength is $O(m)$ and the alphabet size is $q^{m^{p-i}}$). Since there are $O(m^{i-1})$ Reed-Solomon decodings to be performed at the i'th stage, the total decoding time for level i is $O(m^{p+1} \log^{O(1)} n)$ which is certainly $O(n^{1+\beta})$ (since $p \ge 2/\beta$). Similarly the decoding of the last two levels of Reed-Solomon codes can also be performed in $O(n^{1+\beta})$ time. The overall decoding can therefore be accomplished in $O(n^{1+\beta})$ time. (The reader might note the similarity between the above procedure and the decoding algorithm for multi-concatenated codes from Lemma 9.15. The main difference is that, at each level of the multi-concatenation, instead of using the codes and decoding algorithms guaranteed by Lemma 9.11, we use Reed-Solomon codes and the very efficient list recovering algorithm for Reed-Solomon codes from Theorem 6.17.)

The exact rate of each of the Reed-Solomon codes and $\tilde{C}$ must be chosen small enough so that the above recursive decoding strategy will work and actually list decode the entire code from a fraction $(1-\alpha)$ of errors. This can be done similarly to the proof of Lemma 9.15, and we omit the tedious details. But the important thing is that all rates can be picked to be a positive constant (solely as a function of α, p), and thus the overall rate will be a constant depending upon α, β (though it will have a rather bad dependence on β, and not better than around $\alpha^{-O(1/\beta^2)}$ since the rate goes down exponentially in p^2, cf. Lemma 9.15). Since we only claimed a positive rate that depends exclusively on α, β, we do not have to bother about a precise calculation of the rate. The claimed encoding time also follows easily from the construction (by using FFT based techniques to encode the Reed-Solomon codes and an $O((\log \log m)^2)$ time encoding algorithm for the innermost code $\tilde{C}$). □

Remark on the necessity of a strongly polynomial time Reed-Solomon decoding algorithm: Note that the near-linear dependence of the Reed-Solomon decoding algorithms over GF(Q) on $\log Q$ was *critical* for the above application (due to the very large alphabets over which the Reed-Solomon codes were defined). In particular, we had to use the result of Theorem 6.17, which in turn followed from the work of Augot and Pecquet [AP00], as opposed to, for example, the result of Theorem 6.16. The above application is thus nice in the sense that it highlights an important difference between the various Reed-Solomon list decoding algorithms, and crucially exploits a distinguishing feature, namely the strongly polynomial runtime, of one of the algorithms.

Construction of the necessary "expander"

We now move on to the explicit construction of a graph with the properties required in 2(i) and 2(ii) (stated at the beginning of Section 11.4.1). The existence of such a bipartite graph follows from the existence of certain expander graphs. There are several ways in which expander graphs are defined in the literature. The definition of expanders we gave in Chapter 9 is the common definition when one only needs vertex expansion. For our application here we will also need some isoperimetric properties offered by expanders, and therefore we use a spectral definition of expanders based on the second largest eigenvalue of the normalized adjacency matrix. Under this definition a Δ-regular graph H on n vertices with adjacency matrix $\mathbf{A}$ is an expander if $\lambda(H) < 1$, where $\lambda(H) \stackrel{\text{def}}{=} \max\{\lambda_2, |\lambda_n|\}$ is defined to be the second largest eigenvalue in magnitude and $1 = \lambda_1 \ge \lambda_2 \ge \cdots \lambda_n \ge -1$

are the n eigenvalues of $\frac{1}{\Delta} \cdot \mathbf{A}$.

The following result relating the second eigenvalue to vertex expansion is well-known and has appeared in many places (see, for example, Theorem 2.4 of [AS92, Chap. 9]).

Lemma 11.2 *Let $H = (V, E)$ be a Δ-regular graph with $n = |V|$ and $\lambda(H) = \lambda$, and let $T \subseteq V$ with $|T| = bn$. Let $t = |v \in V : N(v) \cap T = \emptyset\}|$ be the number of vertices of H that have* no *neighbors in T. Then*

$$t \leq \frac{\lambda^2(1-b)n}{b} . \tag{11.1}$$

The above lemma applies to a general graph while we are interested in bipartite graphs. But this is easily fixed. One can define a $n \times n$ bipartite graph $G = (A, B, E')$ from the above graph H by letting A, B to be copies of V and connecting a vertex $a \in A$ with $b \in B$ iff the corresponding vertices in V are adjacent in H. We call such a graph G the *double cover* of H. Together with the above lemma, this gives us the desired bipartite expanders, stated in the form of the following corollary.

Corollary 11.3 *Let H be a Δ-regular graph on n vertices with $\lambda(H) = \lambda$. Let $G = (A, B, E)$ be the double cover of H. Then for every subset $X \subseteq A$ with $|X| \geq bn$, we have $|\Gamma(X)| \geq (1 - \frac{\lambda^2}{b})n$ where $\Gamma(X) \subseteq B$ is the set of all nodes with some neighbor in X.*

Expander graphs with $\lambda \ll 1$ also have good *isoperimetric* properties. Loosely speaking this means that the fraction of edges between two large sets of vertices approximately equals the product of the densities of those sets. The formal lemma, stated below, is folklore (see for example Corollary 2.5 in [AS92, Chap. 9]).

Lemma 11.4 *Let H be a Δ-regular graph with $\lambda(H) = \lambda < 1$. Let $G = (A, B, E)$ be the double cover of H. Then for every pair of subsets $X \subseteq A$ and $Y \subseteq B$, we have*

$$\left| \frac{E(X:Y)}{\Delta|X|} - \frac{|Y|}{|B|} \right| \leq \lambda \sqrt{\frac{|Y|}{|X|}} .$$

Thus a low value of λ achieves both good vertex expansion and isoperimetric properties. It is known, however, that the best value of λ one can hope for in an infinite family of Δ-regular graphs is $\frac{2\sqrt{\Delta-1}}{\Delta} - o(1)$. Amazingly enough, there are explicitly known constructions of an infinite family of Δ-regular graphs $\{G_i\}_{i \geq 1}$ with $\limsup_{i \to \infty} \lambda(G_i) = \frac{2\sqrt{\Delta-1}}{\Delta} < \frac{2}{\sqrt{\Delta}}$. These graphs, which are called Ramanujan graphs, were constructed independently in [LPS88] and [Mar88].

Using Corollary 11.3 (with the choice $b = 1/2$) and Lemma 11.4, we conclude that for $\Delta = O(1/\varepsilon + 1/\delta)$, Ramanujan graphs of degree Δ give us an infinite family of the desired bipartite graphs H that satisfy both the expansion property (i) and the isoperimetric property (ii), for parameters $(\varepsilon, \delta, \delta' = \delta/4)$. We record this in the lemma below. Note that we are only using one direction of the isoperimetric property (namely the lower bound on $\frac{E(X:Y)}{\Delta|X|}$) from Lemma 11.4. We will need the other direction of the isoperimetric property (namely an upper bound on $\frac{E(X:Y)}{\Delta|X|}$) in the construction of linear-time binary codes in Sections 11.5 and 11.6.

Lemma 11.5 *For every $\varepsilon > 0$ and $\delta \le \varepsilon$, there exists $\Delta = O(1/\varepsilon)$ such that for infinitely many n there exists an* explicitly constructible *$n \times n$ Δ-regular bipartite graph G with the necessary properties (i) and (ii) defined at the beginning of Section 11.4.1, for parameters $(\varepsilon, \delta, \delta' = \delta/4)$.*

11.4.2 Codes with rate $\Omega(\varepsilon)$, relative distance $(1-\varepsilon)$ and alphabet size $2^{O(1/\varepsilon)}$

Theorem 11.6 *For every $\beta, \delta > 0$ there is a constant $B > 1$ such that for all small enough $\varepsilon > 0$ there is an explicitly specified code family with the properties:*

(i) It has rate ε/B, relative distance at least $(1-\varepsilon)$ and alphabet size $2^{O(1/\varepsilon)}$.

(ii) A code of blocklength n in the family can be list decoded in $O(n^{1+\beta})$ time from up to a fraction $(1-\delta)$ of errors, and can be encoded in $O(n \log^{O(1)} n)$ time.

Proof: Let $\beta, \delta > 0$ be given. We will construct an $(n,k)_Q$ code C_1 with $Q = q^\Delta$ starting from the code C guaranteed by (1) at the beginning of Section 11.4.1, for the choice $\alpha = \delta/8$ and β as given. We will use the code C together with a Δ-regular bipartite expander as guaranteed by Lemma 11.5. The expander will be used to redistribute the symbols in a codeword of C to give a codeword in the final code C_1. To encode a message x, we first encode x by C, and then we push copies of each symbol of $C(x)$ along the edges of G and form $C_1(x)$. Formally, the i'th symbol of $C_1(x)$, $1 \le i \le n$, will be the juxtaposition of $C(x)_{i_1}, \ldots, C(x)_{i_\Delta}$ where $i_1, i_2, \ldots, i_\Delta$ are the Δ neighbors on the left side of G, in some fixed order, of the vertex number i on the right side of G.[5]

From the vertex-expansion property (i) of G and the fact that C has distance at least $n/2$, it is easy to see that C_1 has distance at least $(1-\varepsilon)n$. Since the rate of C is a constant (independent of ε) and $\Delta = O(1/\varepsilon)$, it is clear that the rate of C_1 is $\Omega(\varepsilon)$. This proves Part (i) of the theorem.

The decoding algorithm that list decodes C_1 up to a $(1-\delta)n$ errors works as follows. Let $z \in [Q]^n$ be a received word and x be any message such that $\Delta(C_1(x), z) \le (1-\delta)n$.[6] Let Y be the set positions on which z agrees with $C_1(x)$; therefore $|Y| \ge \delta n$. The algorithm tries to recover the codeword of C that corresponds to z. This is done by constructing, for each $i = 1, 2, \ldots, n$ (i.e., each node on the left side of the graph G), a list $L(i)$ of t most popular symbols among the corresponding values stored in z on positions in the neighborhood of i, i.e. those in $\{j : (i,j) \in E\}$. (For each such j, z_j is a Δ-tuple over $[q]$, one of whose components comes from the vertex number i on the left side of G — it is this component that votes for the i'th symbol of $C(x)$. The final list $L(i)$ is comprised of the t symbols that receive the most votes.) The crucial point is that if $C(x)_{|i} \notin L(i)$, then the number of edges in $\{(i,j) \in E : j \in Y\}$ is at most $\frac{\Delta}{t+1}$. Let X be the set of i's such that $C(x)_{|i} \notin L(i)$. It follows that

$$\frac{|E(X:Y)|}{\Delta|X|} \le \frac{1}{(t+1)} .$$

[5]The construction requires that the blocklength of C is exactly equal to the number of vertices in G. Since G is constructed using Ramanujan graphs, this means that the blocklength of C should be such that an explicit Ramanujan graph exists on that many vertices. As with the constructions in Chapter 9, this will also be an assumption in our code constructions in this chapter. As discussed in Section 9.4.3 of Chapter 9, Ramanujan graphs occur in sufficient density that it is not a problem to achieve this. Therefore, to keep the presentation simple, for the rest of the chapter we simply assume that the blocklength of our "left code" and the number of vertices in the "expander" graph match exactly.

[6]The double use of Δ for both the Hamming distance and the degree of the expanders is unfortunate. We hope that it will not cause any confusion.

Let us pick $t = \lfloor \frac{1}{\delta'} \rfloor$. Then $\frac{1}{t+1} < \delta'$, and therefore, by Property 2(ii) of G (stated at the beginning of Section 11.4.1), we must have $|X| < n/2$. So, we can pick one symbol from each $L(i)$ at random and we get a string s with expected fractional distance from $C(x)$ at most $\frac{1}{2} + \frac{1}{2} \cdot (1 - \frac{1}{t}) = 1 - \frac{1}{2t}$.[7] Since $t = \lfloor \frac{1}{\delta'} \rfloor$, we have

$$1 - \frac{1}{2t} \leq 1 - \frac{\delta'}{2} = 1 - \frac{\delta}{8} = 1 - \alpha \ .$$

Therefore $\Delta(C(x), s) \leq (1 - \alpha)n$, and we can decode x successfully using the list decoding algorithm for C. The running time of this procedure is $O(n/\varepsilon)$ plus the time to decode C, which is $O(n^{1+\beta})$, for a total of $O(n^{1+\beta})$ time. The encoding time is the time required to encode a message by C, which is $O(n \log^{O(1)} n)$, plus the time required to redistribute the symbols using the expander, which is $O(n/\varepsilon)$, for a total of $O(n \log^{O(1)} n)$ time. □

Corollary 11.7 *For every $\varepsilon > 0$, there exists a rate $\Omega(\varepsilon)$ code family over an alphabet of size $2^{O(1/\varepsilon)}$ with relative distance at least $(1-\varepsilon)$ that can be uniquely decoded from a fraction $(1 - \varepsilon)/2$ of errors in near-linear time.*

We stress that even to perform unique decoding of the final code C_1 as stated in the above corollary we need the list decodability of C.

11.4.3 Binary codes of rate $\Omega(\varepsilon^3)$, relative distance $(1/2 - \varepsilon)$

We now concatenate the code C_1 from Theorem 11.6 with a binary code to get binary codes of rate $\Omega(\varepsilon^3)$ and relative distance at least $(1/2 - \varepsilon)$ and which have efficient algorithms to decode up to a fraction $(1/4 - \varepsilon)$ of errors.

Theorem 11.8 *For any $\beta > 0$ and any η, $0 < \eta < 1/2$, there is a constant $B > 1$ such that for all small enough $\varepsilon > 0$ there exists a* binary *code family with the following properties:*

- *It has rate ε^3/B and relative distance at least $(1/2 - \varepsilon)$.*
- *Every code in the family can be constructed in probabilistic $O(1/\varepsilon^3)$ or deterministic $2^{O(1/\varepsilon^3)}$ time.*
- *A code of blocklength N from the family can be list decoded from up to a fraction η of errors in $O(N^{1+\beta} + 2^{O(1/\varepsilon^3)})$ time, and can be encoded in $O(N \log^{O(1)} N)$ time.*

Proof: The basic idea is to concatenate the $(n, k)_Q$ code C_1 from Theorem 11.6 with a suitable binary inner code. For the inner code, take an arbitrary $[n', k']_2$ binary linear code C' with $k' = \lg Q = O(1/\varepsilon)$, $n' = O(k'/\varepsilon^2) = O(1/\varepsilon^3)$ and relative distance $(1 - \varepsilon)/2$. A random $[n', k']_2$ linear code will meet the Gilbert-Varshamov bound and have the necessary relative distance with high probability. Hence the code C' can be found with high probability in randomized $O(1/\varepsilon^3)$ time or by brute-force search in deterministic $2^{O(1/\varepsilon^3)}$ time. Pick η' to be a constant that satisfies $\eta < \eta' < 1/2$. The Johnson bounds for list decoding discussed in Chapter 3 (specifically Theorem 3.2) imply that, for all small enough ε, such a code C' is $(\eta' n', A)$-list decodable, where A is a constant that depends on η but is independent of ε.

[7] It is trivial to derandomize this and deterministically generate a list of at most t strings $s_1, s_2, \ldots, s_t$, one of which satisfies $\Delta(C(x), s_j) \leq (1 - 1/2t)n$. This can be done, for instance, by taking s_j to comprise of the j'th symbols from each of the lists $L(i)$, $1 \leq i \leq n$. We describe the randomized variant for ease of presentation.

We obtain the desired binary code C_2 by concatenating the code C_1 from Theorem 11.6 with C'. Clearly, C_2 has blocklength $N = nn' = O(n/\varepsilon^3)$, rate $\Omega(\varepsilon^3)$, and minimum distance at least $d = \frac{(1-\varepsilon)^2 N}{2}$.

The decoding of C_2 proceeds as follows. Let $\mathbf{r}$ be a received word and let x be any message that satisfies $\Delta(C_2(x), \mathbf{r}) \leq \eta N$. The goal of the list decoding algorithm for C_2 is to find a list that includes the message x, and it proceeds as follows. We can write the received word as $\mathbf{r} = \mathbf{r_1}, \mathbf{r_2}, \ldots, \mathbf{r_n}$, where each $\mathbf{r_i}$, $1 \leq i \leq n$, is the portion of the received word corresponding to the i'th inner block. For $1 \leq i \leq n$, let $L(\mathbf{r_i})$ be a list of all symbols $\gamma \in [Q]$ such that $\Delta(C'(\gamma), \mathbf{r_i}) \leq \eta' n'$. By the $(\eta' n', A)$-list decodability property of C', we know that each list $L(\mathbf{r_i})$ contains at most A symbols. Moreover, each $L(\mathbf{r_i})$ can be found by a brute-force decoding of C' which will take $Q = 2^{O(1/\varepsilon)}$ time, and this can be speeded up by table look-up to $O(1/\varepsilon^3)$ time (building up the table takes up a *one-time* cost of $2^{O(1/\varepsilon^3)}$ time and space).

Now, if $C(x)_{|i} \notin L(\mathbf{r_i})$ for some i, then we must have $\Delta(C_2(x)_{|i}, \mathbf{r_i}) \geq \eta' n'$ where $C_2(x)_{|i}$ denotes the i'th block of the encoding of x by C_2. Together with $\Delta(C_2(x), \mathbf{r}) \leq \eta n n'$, this implies, by an averaging argument, that the number of i's such that $C(x)_{|i} \notin L(\mathbf{r_i})$ is at most $\mu \stackrel{\text{def}}{=} \frac{\eta}{\eta'} < 1$. Therefore, if we choose one of the symbols from each $L(\mathbf{r_i})$ at random, the resulting string s has expected fractional distance to $C_1(x)$ smaller than $\mu + (1-\mu)(1 - 1/A) = (1 - \frac{1-\mu}{A})$.[8] If we pick δ (for the construction of the outer code C_1 from Theorem 11.6) to be smaller than $(1-\mu)/A$, we will be able to successfully decode x by running the list decoding algorithm for C_1 on the string s. The decoding time includes the time to decode the inner codes, which takes the one-time cost of $2^{O(1/\varepsilon^3)}$ to build the look-up table and $O(n/\varepsilon^3) = O(N)$ time to actually perform the n inner decodings, and the time to list decode the outer code C_1 up to a fraction $(1-\delta)$ of errors, which takes $O(n^{1+\beta})$ time by Theorem 11.6. The encoding time of C_2 is dominated by the encoding time of C_1, which is $O(n \log^{O(1)} n)$. Since the overall blocklength N of the concatenated code C_2 is at least n, the claimed encoding and decoding times follow. This completes the proof of the theorem. □

Remark: In the construction of Theorem 11.6, we can start with C which is an additive code over a field $\mathbb{F}_q$ of characteristic 2. In this case, the final code C_1 obtained in Theorem 11.6 will also be an additive code over a field $\mathrm{GF}(Q)$ of characteristic 2. Concatenating an additive code over $\mathrm{GF}(Q)$ with a binary linear code of dimension $\lg Q$ will result in a concatenated code which is a binary *linear* code. Hence, we can get the above theorem with the added requirement of a linear code family.

We also note that even in order to get unique decoding of the above codes up to a fraction $(1/4 - \varepsilon)$ of errors, we would need the list decodability of the code C_1 from Theorem 11.6.

11.5 Linear-time encodable and decodable codes: Construction I

In this section, we present another version of the above constructions, which by worsening the rate by a factor of ε, reduces the decoding time to linear in the blocklength. Moreover, we can also make the codes linear-time encodable. However, we can only perform unique decoding of these codes. This gives our codes (2a,2b) mentioned in Section 11.2. In the

[8] As in the proof of Theorem 11.6, it is easy to find such an s deterministically.

next section, we will improve the rate to match those from the previous section and thus get linear-time codes with the best known rates (specifically the codes (3a,3b) mentioned in Section 11.2). The codes of this section are however easier to describe and contain the main idea behind the constructions (3a,3b), and hence we discuss these first. We would recommend reading this section before reading the improved constructions in Section 11.6.

11.5.1 Codes with rate $\Omega(\varepsilon^2)$ decodable up to a fraction $(1/2-\varepsilon)$ of errors

Theorem 11.9 *For any $\varepsilon > 0$ there is an explicitly specified code family with rate $\Omega(\varepsilon^2)$, relative distance at least $(1-\varepsilon)$ and alphabet size $2^{O(1/\varepsilon^2)}$, such that a code of blocklength n from the family can be (a) encoded in $O(n/\varepsilon^2)$ time, and (b) uniquely decoded from up to a fraction $(1/2-\varepsilon)$ of errors in $O(n/\varepsilon^2)$ time.*

Proof: We need the following two combinatorial objects for our code construction:

(1) A binary asymptotically good $[n,k]_2$ linear code C, encodable and uniquely decodable from a fraction $\gamma > 0$ of errors in *linear time* (here γ is an absolute positive constant). An explicit construction of such a code is known [Spi96, Spi95].

(2) A Δ-regular bipartite graph $G = (A, B, E)$ with $|A| = |B| = n$, such that:

  (a) for every set $X \subset A$ with $|X| \geq \gamma n$, if Y is the set of neighbors of X in G, then $|Y| \geq (1-\varepsilon)|B|$.

  (b) for every set $Y \subset B$ with $|Y| \geq (1/2+\varepsilon)n$, the set $X' \subseteq A$ defined by

$$X' = \{x \in A : \ x \text{ has as many neighbors in } B \setminus Y \text{ as in } Y\} \tag{11.2}$$

  has size at most γn.

A graph as in (2) above with $\Delta = O(\frac{1}{\gamma\varepsilon^2})$ can be obtained from a Ramanujan graph (i.e., an expander with second largest eigenvalue $O(1/\sqrt{\Delta})$). Indeed let $H = (V, E')$ be a Δ-regular Ramanujan graph with $\lambda(H) = \lambda = O(1/\sqrt{\Delta})$. Take G to be the double cover of H. We will prove that G both the properties (a) and (b) described above. For property (a), we apply Corollary 11.3 with the choice $b = \gamma$. This gives that for all $X \subseteq A$ with $|X| \geq \gamma n$, the set $Y \subseteq B$ of all nodes with neighbors in X satisfies

$$|Y| \geq \Big(1 - \frac{\lambda^2}{\gamma}\Big)n \geq \Big(1 - O\Big(\frac{1}{\Delta\gamma}\Big)\Big)n \geq (1-\varepsilon^2)n > (1-\varepsilon)n \ ,$$

for $\Delta = \Omega(\frac{1}{\gamma\varepsilon^2})$.

For the second property (b), assume that $|Y| \geq (1/2+\varepsilon)n$ and let X' be defined as in (11.2). We need to prove that $|X'| \leq \gamma n$. By the definition of X', we have $E(X' : Y) \leq \Delta|X'|/2$. Applying the result of Lemma 11.4, we know that

$$\begin{aligned} \frac{E(X' : Y)}{\Delta|X'|} &\geq \frac{|Y|}{n} - \lambda\sqrt{\frac{|Y|}{|X'|}} \\ &\geq \Big(\frac{1}{2}+\varepsilon\Big) - \lambda\sqrt{\frac{|Y|}{|X'|}} \ . \end{aligned}$$

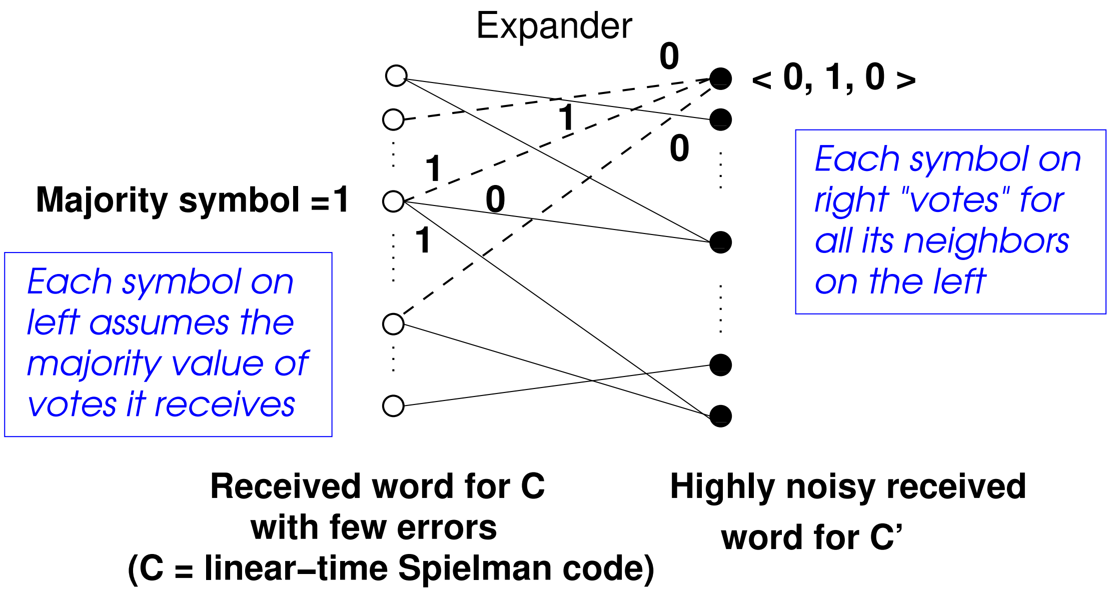

Figure 11-4: The majority voting based decoding algorithm

Together with $E(X' : Y) \leq \Delta|X'|/2$, this implies that

$$|X'| \leq \frac{\lambda^2 |Y|}{\varepsilon^2} = O\Big(\frac{n}{\Delta\varepsilon^2}\Big) \leq \gamma n \ ,$$

for $\Delta = \Omega(\frac{1}{\gamma\varepsilon^2})$. Hence we conclude that the graph G required in (2) above exists with degree $\Delta = O(\frac{1}{\gamma\varepsilon^2}) = O(1/\varepsilon^2)$, since γ is an absolute constant.

Given the code C and graph G, our final code, call it C', is constructed in the same way as the code from Theorem 11.6. Namely, to encode a message x according to C', we first encode it into $C(x)$, and then push symbols of $C(x)$ along the edges of G. The i'th symbol of the codeword $C'(x)$, for $1 \leq i \leq n$, comprises of the collection of the symbols received at the i'th node of the right side B of G. This is the same as the construction illustrated in Figure 11-3, with the left code being fixed to the linear-time codes due to Spielman [Spi96].

Since C has constant rate, clearly C' has rate $\Omega(1/\Delta) = \Omega(\varepsilon^2)$. Since C is uniquely decodable up to a fraction γ of errors, its relative distance must be at least 2γ, and this together with the expansion property (a) of G clearly implies that C' has relative distance at least $(1 - \varepsilon)$.

The encoding time for C' is the same as for C (i.e., linear), plus $O(n\Delta) = O(n/\varepsilon^2)$. In order to decode a received word z which differs from a codeword $C'(x)$ in at most a fraction $(1/2 - \varepsilon)$ of positions, we first perform the following key voting step, which is illustrated in Figure 11-4: *Each node v in A recovers the bit which is the majority of the neighbors of v in B (ties broken arbitrarily).*

Since z and $C'(x)$ agree on at least $(1/2 + \varepsilon)n$ positions, appealing to the property (b) of the graph G, we conclude that at most γn nodes in A recover incorrect bits of $C(x)$ in the above voting procedure. Then, by the property of the code C, we can decode x in linear time. The total decoding time is again equal to $O(n/\varepsilon^2)$ for the first stage and then a further $O(n)$ time for the decoding of C. Hence the total decoding time is $O(n/\varepsilon^2)$, as claimed. $\square$

11.5.2 Binary codes with rate $\Omega(\varepsilon^4)$ decodable up to a fraction $(1/4-\varepsilon)$ of errors

In this section we show how to augment the linear-time codes from the previous section in order to obtain binary codes with linear-time encoding, and linear-time decoding up to a fraction $(1/4-\varepsilon)$ of errors.

Theorem 11.10 *For every $\varepsilon > 0$ there is a binary linear code family of rate $\Omega(\varepsilon^4)$ and relative distance at least $(1/2-O(\varepsilon))$, such that a code of blocklength N from the family can be uniquely decoded from up to a fraction $(1/4-\varepsilon)$ of errors in $O(N/\varepsilon^2 + 2^{O(1/\varepsilon^4)})$ time, and can be encoded in $O(N + 2^{O(1/\varepsilon^2)})$ time. The code can be constructed in probabilistic $O(1/\varepsilon^4)$ or deterministic $2^{O(1/\varepsilon^4)}$ time.*

Proof: The code is constructed by concatenating the code from Theorem 11.9 with a suitable binary code. Details follow. Let C' be the code from the Theorem 11.9.[9] The alphabet size of C' is $Q = 2^{O(1/\varepsilon^2)}$. Let C_3 be any $[O(\lg Q/\varepsilon^2), \lg Q]_2$ linear code with relative distance at least $(1/2-\varepsilon)$. Such a code can be constructed by a picking random linear code from a "Wozencraft ensemble" in probabilistic $O(1/\varepsilon^4)$ time or by a brute-force search in such an ensemble in $2^{O(1/\varepsilon^4)}$ time, cf. Proposition 8.10. We concatenate C' with C_3 obtaining a binary linear code , say C^*, of blocklength $N = O(n/\varepsilon^4)$, rate $\Omega(\varepsilon^4)$ and relative designed distance at least $\delta \stackrel{\text{def}}{=} (1-\varepsilon)(1/2-\varepsilon) = (1/2 - O(\varepsilon))$.[10] Since C' can be encoded in $O(n/\varepsilon^2)$ time, the encoding of C^* can be performed in $O(n/\varepsilon^4)$ time (since each encoding by C_3 can be done in $1/\varepsilon^4$ time using a look-up table building which takes a one-time cost of $2^{O(1/\varepsilon^2)}$ time and space). As the overall blocklength of C^* equals $N = O(n/\varepsilon^4)$, the claimed encoding time holds.

We will now show how to unique decode C^* from a fraction $\delta/2$ of errors in linear-time. Since $\delta = (1-\varepsilon)(1/2-\varepsilon)$ and $\varepsilon > 0$ is arbitrary, this will imply the claimed result. To this end, we use the following general result on decoding concatenated codes. This result appears to be folklore and has been observed and used by several authors, including Justesen [Jus72] who gave an algorithm to decode his asymptotically good concatenated code construction up to the product bound (i.e., half the product of the distances of the outer and inner codes). The basic idea and inspiration behind this method comes from the work of Forney [For66b] on Generalized Minimum Distance (GMD) decoding.

Proposition 11.11 (GMD decoding of concatenated codes) *Let C_{out} be an $(N,K)_Q$ code where $Q = q^k$ and let C_{in} be an $(n,k)_q$ code with minimum distance at least d. Let $\mathbf{C}$ be the $(Nn, Kk)_q$ code obtained by concatenating C_{out} with C_{in}. Assume that there exists an algorithm running in time T_{in} to uniquely decode C_{in} up to less than $d/2$ errors. Assume also the existence of an algorithm running in time T_{out} that uniquely decodes C_{out} from S*

[9] Actually, we will need to make slight changes in the assumptions about the components used in the construction of C' in Theorem 11.9, namely in the assumptions about the expander graph G. But the construction of C' itself (given the left code C and the expander G), as well all its properties claimed in Theorem 11.9, remain unaltered — we will only pose some stronger requirements on G and the decodability of C'. We will discuss these and justify how they can be achieved without any loss in rate later in the proof.

[10] The code C^* will be linear since C_3 is linear and it is easy to check that the construction from Theorem 11.9 gives an additive code C'.

erasures and E errors as long as $2E + S < \tilde{D}$ for some $\tilde{D} \le \mathrm{dist}(C_{\mathrm{out}})$. Then there exists an algorithm $\mathcal{A}$ running in $O(NT_{\mathrm{in}} + dT_{\mathrm{out}})$ time that uniquely decodes $\mathbf{C}$ from any pattern of less than $\frac{d\tilde{D}}{2}$ errors.

The number of times the decoding algorithm for C_{out} is invoked by the final algorithm $\mathcal{A}$ is at most d (and hence the dT_{out} term in the runtime), and *independent* of the outer blocklength N. This will be crucial for us. The fact that GMD decoding allows for such a few invocations of the outer decoding algorithm has been observed before (for example, by Justesen [Jus72] who makes a one line comment on it). For completeness sake, we provide a proof of the above proposition in Appendix A.

Since the relative distance of the inner code C_3 in our concatenated construction C^* is at least $(1/2-\varepsilon)$, it can be uniquely decoded up to a fraction $(1/2-\varepsilon)/2$ of errors. Since the blocklength of C_3 is $O(1/\varepsilon^4)$, this can be accomplished in $O(1/\varepsilon^4)$ time using table look-up where building up the table takes a one-time cost of $2^{O(1/\varepsilon^4)}$ time. Below we will show that outer code C' can be decoded in $O(n/\varepsilon^2)$ time from any pattern of fraction s of erasures and fraction e of errors as long as $2e+s \le (1-\varepsilon)$. Appealing to the above lemma, therefore, we will have an $O(n/\varepsilon^4 + 2^{O(1/\varepsilon^4)} + \varepsilon^{-4} \cdot n/\varepsilon^2)$ time algorithm to decode C^* up to a fraction $(1/2-\varepsilon)(1-\varepsilon)/2 = \delta/2$ of errors (the distance d of the inner code C_3 equals $O(1/\varepsilon^4)$). Hence the total decoding time is $O(n/\varepsilon^6 + 2^{O(1/\varepsilon^4)}) = O(N/\varepsilon^2 + 2^{O(1/\varepsilon^4)})$ (since $N = O(n/\varepsilon^4)$).

It remains to show that the outer code C' can be decoded, in linear time, from a fraction s of erasures and a fraction e of errors for any e, s that satisfy $2e+s \le (1-\varepsilon)$. To this end, recall the construction of C' from Theorem 11.9 using a binary asymptotically good code C and the expander G. For this proof, we assume some small changes in the parameters of the expander $G = (A, B, E)$. Specifically, we assume that G satisfies a property (b'), defined below, which is stronger than the property (b) from Equation (11.2) that was used in Theorem 11.9.[11]

(b') For every subset $X \subset A$ with $|X| \ge \gamma n/2$ and every $Y \subseteq B$, we have

$$\left| \frac{|E(X:Y)|}{|X|\Delta} - \frac{|Y|}{|B|} \right| \le \varepsilon/3 \ . \tag{11.3}$$

Note the above strongly resembles the isoperimetric property of Lemma 11.4, and in fact is implied by it provided $\lambda = O(\varepsilon\sqrt{\gamma})$. Therefore, by using Ramanujan graphs which achieve $\lambda = O(1/\sqrt{\Delta})$, we can achieve the above property with $\Delta = O(\frac{1}{\gamma\varepsilon^2})$. Thus, we can achieve the stronger property (b') above with essentially no asymptotic increase in the degree Δ. Hence the rate of C' is still $\Omega(\varepsilon^2)$.

We now proceed to describe the claimed $O(n/\varepsilon^2)$ time error-erasure decoding algorithm for C' that can successfully decode from a fraction s of erasures and a fraction e of errors for any s, e satisfying $2e+s \le (1-\varepsilon)$. Let S be the set of erasures in the received word z, and let F be the set of errors. The decoding algorithm proceeds by the same *majority voting* scheme as before. That is, each node on the left side A of the bipartite graph G assumes the bit which is majority of the values of the neighbors of that node on the right side B.

[11] Of course we could have defined and used this stronger property in the proof of Theorem 11.9 itself. But, we chose to avoid this since we wanted to state only the minimum required property on the expander to make the construction work. This has the advantage that a better construction of the combinatorial object might be possible which in turn might lead to improvements in the code construction.

The difference now is that the nodes corresponding to the set S of erased symbols do not participate in the voting. In order to prove the correctness of the decoding algorithm, we would like to prove that all but a fraction γ of the positions on the left (that of the code C) receive the correct bit as the majority vote by this procedure. Then we can complete the decoding using the linear-time decoder for C (which by assumption can correct up to a fraction γ of errors). Note that the decoding is the time to decode C, i.e. $O(n)$, plus the time to perform the majority votings, which is $O(n/\varepsilon^2)$, for a total of $O(n/\varepsilon^2)$ time, as desired.

To prove that at most a fraction γ of the nodes in A receive the incorrect bit as the majority vote by the above voting procedure, define $X \subseteq A$ to be the set of nodes which have at most a fraction $(1 - s - \varepsilon/3)$ of neighbors in the set $B \setminus S$. Also, define $X' \subseteq A$ to be the set of nodes which have at least a fraction $(e + \varepsilon/3)$ of neighbors in F. It is easy to infer using the property (b') of Equation (11.3) that $|X|, |X'| \leq \gamma n/2$.

Now consider any node from $A \setminus (X \cup X')$. It has less than a fraction $(e+\varepsilon/3)$ of neighbors in F. Moreover, it has more than a fraction $(1 - s - \varepsilon/3) - (e + \varepsilon/3) = (1 - s - e - 2\varepsilon/3)$ of neighbors in $B \setminus (S \cup F)$ (i.e., in the set of correct positions). Since $2e + s \leq (1 - \varepsilon)$, we have $(e + \varepsilon/3) \leq (1 - s - e - 2\varepsilon/3)$. Therefore, every node in $A \setminus (X \cup X')$ has more neighbors among the correct positions (i.e., those in $B \setminus (S \cup F)$) than among the incorrect positions (i.e. those in F). It follows that the the voting procedure works correctly for all nodes in $A \setminus (X \cup X')$, and hence for all but a fraction γ of the codeword positions of C (since $|X|, |X'| \leq \gamma n/2$). This is exactly what we desired to show, and the proof is complete. □

11.6 Linear-time codes with improved "optimal" rate

In this section, we present our linear-time codes (3a,3b) which enable unique decoding up to the maximum radius of $(1/2 - \varepsilon)$ (resp. $(1/4 - \varepsilon)$) for codes over a large alphabet (resp. binary alphabet), and which have rates $\Omega(\varepsilon)$ (resp. $\Omega(\varepsilon^3)$). The former rate is in fact *optimal* (up to constant factors), while the latter $\Omega(\varepsilon^3)$ rate for binary codes is the best known for constructive binary codes with relative distance $(1/2 - O(\varepsilon))$. Thus in some sense our results get the optimal error-correction performance with the best rate one can (currently) hope for, together with linear time encoding and decoding algorithms.

The construction follows an approach similar to (2a,2b) with some differences that enable us to improve the rate by a factor of ε factor. Our presentation in this section will sometimes refer to ideas from the previous section, so familiarity with the constructions (2a,2b) might be useful in reading this section.

As usual, we first present the construction over a large alphabet. The binary construction will then follow using an appropriate concatenation together with GMD decoding as in Theorem 11.10.

11.6.1 Codes of rate $\Omega(\varepsilon)$ uniquely decodable up to a fraction $(1/2 - \varepsilon)$ of errors

Below we state a more general result that handles both errors and erasures. This will help us deduce the result for binary codes in the next section very easily, since the GMD algorithm (as in the proof of Theorem 11.10) requires an errors-and-erasures decoding algorithm.

Theorem 11.12 *For any $\varepsilon > 0$ there is an* explicitly *specified code family with rate $\Omega(\varepsilon)$, relative distance at least $(1 - \varepsilon/2)$ and alphabet size $2^{O(\varepsilon^{-2} \log(1/\varepsilon))}$, such that a code of block-*

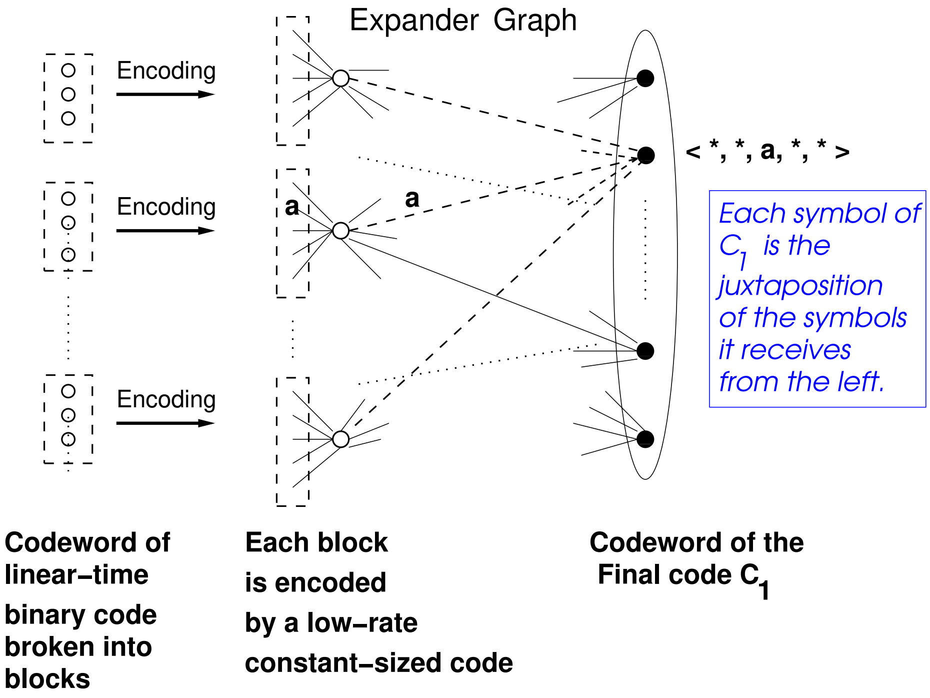

Figure 11-5: Basic structure of the construction from Theorem 11.12 that achieves the optimal $\Omega(\varepsilon)$ rate for unique decoding up to a fraction $(1/2 - \varepsilon)$ of errors. The binary code is first broken into blocks and each block encoded by a constant-sized code. Note that the *second* symbol **a** of the encoded block is sent to the *second* neighbor of the corresponding node of the expander. This is in general how symbols are redistributed from the left to the right using the expander. On the right side, the symbol at each position is the juxtaposition of the symbols received from the neighbors on the left. For example, in the figure the second position receives **a** from its third neighbor on the left, and therefore has **a** at the third position of the 5-tuple of symbols that it receives.

length N from the family can be (a) encoded in $O(N \log^{O(1)}(1/\varepsilon)/\varepsilon^2)$ time, and (b) uniquely decoded from a fraction e of errors and a fraction s of erasures in $O(N \log^{O(1)}(1/\varepsilon)/\varepsilon^2)$ time as long as e, s satisfy $2e + s \leq (1 - \varepsilon)$.

Proof: The construction is roughly similar to that of Theorem 11.9 (where we achieved a rate of $\Omega(\varepsilon^2)$). The main difference is that the symbols of the codeword of the linear-time binary left code C are not directly fed into the expander graph G (as in Figure 11-3), but rather the codeword of C undergoes another level of encoding before this happens. The intuition behind the construction is the following. The construction of Theorem 11.9 can be viewed as encoding each symbol of the left code by a repetition code (of length equal to the degree of the expander), and then sending these repeated symbols along the edges of the expander. The construction here attempts to use a better choice for this code (compared to the repetition code). By picking an appropriate code for this step, we are able to improve the rate of the whole construction to the optimal bound of $\Omega(\varepsilon)$.

The specific details are as follows. Each codeword of C is broken up into blocks each with b bits, for some large enough constant b (jumping ahead, we will pick $b = O(\varepsilon^{-1} \log(1/\varepsilon))$).

Each block is then encoded using a constant-sized q-ary code for some $q = O(1/\varepsilon^2)$, call it $\hat{C}$. The code $\hat{C}$ will have rate $\Omega(\varepsilon)$ and relative distance at least $(1-\varepsilon/4)$. This gives us $n' = n/b$ blocks of, say, $\Delta = O(\frac{b}{\varepsilon \lg q})$ symbols over $\mathbb{F}_q$. The bits in each block are then redistributed using the edges of a suitable Δ-regular bipartite expander with n' vertices on each side in the following way: the j'th bit of the i'th block, for $1 \le j \le \Delta$ and $1 \le i \le n'$, is sent to the j'th neighbor on the right of the the i'th vertex on the left. See Figure 11-5 for an illustration of the basic construction scheme. Let us denote by C_1 the final code obtained after redistribution of the symbols by the expander in the above manner.

We now highlight the construction of each of the three combinatorial objects $C, G, \hat{C}$ used in the above construction.

- The properties we need from C are the same as the one in Theorem 11.9, namely that it be an explicit $[n,k]_2$ code of constant rate encodable and uniquely decodable from a fraction γ of errors in linear time, for an absolute constant $\gamma > 0$. An explicit construction of such a code is known [Spi96].

- For the bipartite expander $G = (A, B, E)$, we will require a Δ-regular expander with $|A| = |B| = n'$ which has the following property (this is the same as Property 11.3 with $\varepsilon/3$ replaced by $\varepsilon/4$):

  (b') For every subset $X \subset A$ with $|X| \ge \gamma n'/2$ and every $Y \subseteq B$, we have
  $$\left| \frac{|E(X:Y)|}{|X|\Delta} - \frac{|Y|}{|B|} \right| \le \varepsilon/4 \ . \tag{11.4}$$

  Once again, as in the proof of Theorem 11.10, Ramanujan graphs of degree $\Delta = O(\frac{1}{\gamma\varepsilon^2})$ give explicit constructions of bipartite graphs with Property (11.4).

- For the code $\hat{C}$ we can take a Reed-Solomon code of rate, say $\varepsilon/4$, and blocklength Δ over a field of size q, where $q = O(\Delta) = O(1/\varepsilon^2)$ is a power of two. It will map blocks of $b = O(\varepsilon^{-1}\log q) = O(\varepsilon^{-1}\log(1/\varepsilon))$ bits, or equivalently $O(1/\varepsilon)$ symbols over $\mathbb{F}_q$, into Δ symbols over $\mathbb{F}_q$. The relative distance of $\hat{C}$ is at least $(1-\varepsilon/4)$, as desired.

Rate, alphabet size, and distance of C_1. Let us estimate the parameters of the final code construction C_1. The blocklength of C_1 is $n' = n/b = O(\varepsilon n/\log(1/\varepsilon))$. Now each position of C_1 receives symbols belonging to $\mathbb{F}_q$ from Δ positions on the left. Therefore the alphabet size of C_1 is $Q = q^\Delta = 2^{O(\varepsilon^{-2}\log(1/\varepsilon))}$ (since $q, \Delta = O(1/\varepsilon^2)$). The dimension of C_1, as a Q-ary code, equals $k' = k/\lg Q = \Omega(\varepsilon^2 k/\log(1/\varepsilon))$. The rate of C_1 is
$$\frac{k'}{n'} = \Omega\Big(\frac{\varepsilon^2 k/\log(1/\varepsilon)}{\varepsilon n/\log(1/\varepsilon)}\Big) = \Omega\Big(\frac{\varepsilon k}{n}\Big) = \Omega(\varepsilon)$$
(since the linear-time binary code C we start from has constant rate).

Let us compute the relative distance of C_1. The relative distance of C is at least 2γ, since it can correct a fraction γ of errors. Let c be a non-zero codeword of C — it must have 1's in at least $2\gamma n$ positions. Hence a fraction 2γ of the blocks of c must be non-zero, and these will be mapped to non-zero strings of Hamming weight at least $(1-\varepsilon/4)\Delta$ by $\hat{C}$ (by the distance property of $\hat{C}$). Let $X_1 \subseteq A$ be the vertices corresponding to such blocks, we have $|X_1| \ge 2\gamma n'$. Let $Y_1 \subseteq B$ be the set of non-zero positions of the final codeword of C_1 that corresponds to c. Each vertex in X_1 contributes at least $(1-\varepsilon/4)\Delta$ non-zero symbols, and hence must have at least $(1 - \varepsilon/4)\Delta$ edges into Y_1. Therefore $\frac{|E(X_1:Y_1)|}{|X_1|\Delta} \geq 1 - \varepsilon/4$. Since $|X_1| \geq 2\gamma n' > \gamma n'/2$, Condition (11.4) implies that $|Y_1| \geq (1 - \varepsilon/2)n'$. Hence any non-zero codeword of C_1 has Hamming weight at least $(1 - \varepsilon/2)n'$, implying that C_1 has relative distance at least $(1 - \varepsilon/2)$.

ENCODING COMPLEXITY. The time to encode C_1 is time to encode C, which is $O(n) = O(n'b)$, plus the time to encode each of the n' Reed-Solomon codes $\hat{C}$ of blocklength Δ, which takes a total of $O(n'\Delta \log^{O(1)} \Delta)$ time, plus the time to redistribute the symbols using the expander, which takes time $O(n'\Delta)$. Since $\Delta = O(1/\varepsilon^2)$, the total encoding time is $O(n' \log^{O(1)}(1/\varepsilon)/\varepsilon^2)$, as claimed.

DECODING COMPLEXITY. It remains to describe the claimed linear time error-erasure decoding algorithm for C_1. Let z be a received word for C_1 with a fraction s of erasures and a fraction e of errors, where $2e + s \leq (1 - \varepsilon)$. Since the relative distance of C_1 is greater than $(1 - \varepsilon)$, there is a unique message x that is solution to the decoding problem. Let S be the set of erasures in the received word z, and let F be the set of errors (i.e., the positions where $C_1(x)$ and z differ). We have $|S| = sn'$ and $|F| = en'$.

Given the received word z, the decoding algorithm proceeds as follows. In the first step, the word z is used to compute certain "received words" z_i, $1 \leq i \leq n'$, for the n' encodings by $\hat{C}$ (corresponding to the n' blocks into which a codeword of C is broken into). This is done as follows. For each i, j, $1 \leq i \leq n'$ and $1 \leq j \leq \Delta$, if the j'th neighbor of the i'th node of A has an unerased symbol, say $\zeta \in \mathrm{GF}(q^{\Delta})$, then the j'th symbol of z_i is set to the symbol in the appropriate coordinate of ζ (namely, the coordinate which received that symbol through the expander). If the j'th neighbor of the i'th node of A has an erased symbol, then we declare an erasure at the j'th position of z_i.

For each i, $1 \leq i \leq n'$, let z_i be the received word thus obtained for the encoding of i'th block. Let s_i be the fraction of positions in z_i which are erased, and let e_i be the fraction of positions in z_i which are set to a wrong symbol. With the z_i's computed, the algorithm continues as follows. For each i, we run a unique error-erasure decoding algorithm for the Reed-Solomon code $\hat{C}$ with received word z_i. If it succeeds in decoding, we let y_i be the message it outputs (note that $y_i \in \{0,1\}^b$), otherwise we let y_i be an arbitrary string in $\{0,1\}^b$. Finally, the decoding is completed by running the linear time unique decoding algorithm for C on the received word $y = \langle y_1, y_2, \ldots, y_{n'} \rangle$, and outputting whatever message x it outputs.

It is easily checked that the runtime of the decoding algorithm is $O(n' \log^{O(1)}(1/\varepsilon)/\varepsilon^2)$ (the significant component is the n' Reed-Solomon decodings, each of which can be implemented in $O(\Delta \log^{O(1)} \Delta)$ time). We now prove the correctness of this procedure. We claim that it suffices to prove that the received words z_i (obtained from the first stage of the decoding that uses the expander) satisfy $2e_i + s_i < (1 - \varepsilon/4)$ for at least $(1 - \gamma)n'$ values of i. Indeed, for any such i, the Reed-Solomon decoder will succeed in finding the correct block y_i (as the relative distance of each Reed-Solomon code is at least $(1 - \varepsilon/4)$). Hence the received word y passed to the decoding algorithm for C will agree with $C(x)$ entirely on a fraction $(1 - \gamma)$ of the blocks, or in other words y and $C(x)$ will differ in at most γn positions. Since the assumed decoding algorithm for C can correct up to a fraction γ of errors, we will correctly find and output the message x.

It remains to prove that $2e_i + s_i < (1 - \varepsilon/4)$ for all but $\gamma n'$ values of i. This part closely parallels the final portions of the proof of Theorem 11.10. Define $X' \subset A$ to be the set of nodes which have at least a fraction $(s + \varepsilon/4)$ of neighbors in the set S (the set of erasures in the received word z). Also define $X'' \subset A$ to the nodes which have at least a fraction

$(e+\varepsilon/4)$ of neighbors in F (the set of erroneous positions in z). It easily follows from the isoperimetric property (11.4) of the expander that $|X'|, |X''| \le \gamma n'/2$.

Now consider any node $i \in A \setminus (X' \cup X'')$. It has less than a fraction $(e+\varepsilon/4)$ of neighbors in F. These correspond to the errors in the received word z_i, and hence we have

$$e_i < e + \varepsilon/4 \quad \text{for every } i \in A \setminus (X' \cup X'') \ . \tag{11.5}$$

A node $i \in A \setminus (X' \cup X'')$ also has less than a fraction $(s+\varepsilon/4)$ of neighbors in S. These correspond to the erasures in the received word z_i, and hence we have

$$s_i < s + \varepsilon/4 \quad \text{for every } i \in A \setminus (X' \cup X'') \ . \tag{11.6}$$

Since $2e + s \le (1-\varepsilon)$ by hypothesis, we have, combining (11.5) and (11.6) that $2e_i + s_i < (1-\varepsilon/4)$, for each $i \in A \setminus (X' \cup X'')$. Since $|X'|, |X''| \le \gamma n'/2$, we have proved that the condition $2e_i + s_i < (1-\varepsilon/4)$ holds for all but a fraction γ of i's in the range $1 \le i \le n'$. This completes the proof of correctness of the decoding algorithm. □

11.6.2 Linear-time binary codes of rate $\Omega(\varepsilon^3)$ decodable up to a fraction $(1/4-\varepsilon)$ of errors

Theorem 11.13 *For every $\varepsilon > 0$ there is a binary linear code family of rate $\Omega(\varepsilon^3)$ and relative distance at least $(1/2-\varepsilon)$, such that a code of blocklength N from the family can be uniquely decoded from up to a fraction $(1/4-\varepsilon)$ of errors in $O(N \log^{O(1)}(1/\varepsilon)/\varepsilon^2 + 2^{O(\varepsilon^{-4}\log(1/\varepsilon))})$ time, and can be encoded in $O(N + 2^{O(\varepsilon^{-2}\log(1/\varepsilon))})$ time. The code can be constructed in probabilistic $O(\varepsilon^{-4}\log(1/\varepsilon))$ or deterministic $2^{O(\varepsilon^{-4}\log(1/\varepsilon))}$ time.*

Proof: The construction is obtained by concatenating the codes C_1 from Theorem 11.12 with a binary linear code, say $\tilde{C}$, of dimension $O(\varepsilon^{-2}\log(1/\varepsilon))$, rate $\Omega(\varepsilon^2)$ and relative distance at least $(1-\varepsilon)/2$. The concatenated code, say C^*, will have rate $\Omega(\varepsilon^3)$ and relative distance at least $(1/2-\varepsilon)$. By Theorem 11.12, C_1 has a linear time error-erasure decoding algorithm to decode a fraction e of errors and a fraction s of erasures as long as $2e + s \le (1-\varepsilon)$. Together with Proposition 11.11 on GMD decoding of concatenated codes, this implies a linear time decoding algorithm for the concatenated code C^* up to a fraction $\frac{(1-\varepsilon)(1-\varepsilon)}{4} \ge (1/4-\varepsilon)$ of errors. The claimed runtimes of the encoding and decoding algorithm follows from a straightforward calculation similar to that from the proof of Theorem 11.10. □

Note that the above result uses only a constant amount of non-constructivity, even which can be eliminated by giving up an ε factor in the rate (one would then use an explicit rate $\Omega(\varepsilon^3)$ code of relative distance $(1/2-O(\varepsilon))$ as the inner code in the concatenation scheme, instead of random codes of rate $\Omega(\varepsilon^2)$).

Chapter 12

Sample Applications outside Coding Theory

An ounce of application is worth a ton of abstraction.

- Booker's Law

We now move on to provide a sample of some of the applications which both combinatorial and algorithmic aspects of list decoding have found in contexts outside of coding theory. As it turns out, by now there are numerous such applications to complexity theory and cryptography. Hopefully, by providing a peek into some of these applications, this chapter will not only highlight the importance of list decoding to these areas, but also give a flavor of why the notion of list decoding perfectly fits the ball in several of these situations. Except for a few of the applications where we will present formal theorem statements and/or proofs, the nature of this chapter is more survey-like, and we will only provide a brief high-level discussion of the applications. But in cases where we only sketch an application, we will provide the relevant pointers where the interested reader can find more details.[1]

We actually begin this chapter with an application of list decoding to an algorithmic question, before moving on to the complexity-theoretic and cryptographic applications. The algorithmic problem, called "Guessing Secrets", is discussed in detail in Section 12.1 (the results of this section appear in a recent paper [AGKS01]). This will be followed by a survey of the several complexity-theoretic applications in Section 12.2. Finally, we will discuss a few cryptographic applications of list decoding in Section 12.3.

12.1 An algorithmic application: Guessing Secrets

Under the familiar "20 questions" game a player, say **B**, tries to discover the identity of some unknown secret drawn by a second player, say **A**, from a large space of N secrets, by asking binary (Yes/No) questions about the secret to **A** (cf. [I52]). The assumption is that **A** answers each question truthfully according to the secret he picked. The goal of **B** is of course to recover the secret by asking as few questions as possible. If the N secrets

[1]The criterion for inclusion of proofs and/or formal statement of results for only a small subset the results is simple — these applications, or at least portions of them, were discovered jointly by the author, and this thesis will be the first published form where the results are discussed. For all other applications there are published articles or preprints where further details can be found, and accordingly we only sketch the details and point the reader to the original articles.

are associated with $\lceil \lg N \rceil$-bit strings, then clearly $\lceil \lg N \rceil$ questions are both necessary and sufficient to discover the secret.

Now, consider the following variant of the above game. Under this variant, the player **A** picks not one, but a set of k secrets, for some $k \geq 2$. For each question asked by **B**, **A** gets to adversarially choose which one of the k secrets to use in supplying the answer, but having made the choice must answer truthfully according to the chosen secret. This variant was introduced by Chung, Graham and Leighton in [CGL01], and they called the problem "Guessing Secrets". In this situation, what is the best strategy for **B**, and how many questions does it take in the worst-case for **B** to "find" the secrets? In addition to being an interesting "puzzle", secret guessing problems of this type have apparently arisen recently in connection with certain Internet traffic routing applications (cf. [CGL01]). Moreover, problems of a related nature have been studied in the literature under the label of separating systems (see [Seg94, CES01] and references therein), and have been applied in different areas of computer science such as technical diagnosis, constructions of hash functions, and authenticating ownership claims. The focus of much of this line of work has been combinatorial, and our work appears to be the first to present non-trivial algorithms to deal with (certain kinds of) separating systems. Specifically, in this section we present an algorithmic solution using list decoding to a problem left open by the work of [CGL01] for the case of $k = 2$ secrets. The details will be made clear shortly.

12.1.1 Formal problem description

We first restrict ourselves to the case $k = 2$ when there are only two secrets. This is already a non-trivial case, and as we shall see one where a very satisfactory solution exists to the guessing secrets problem. In this case, **A** has a set $X = \{x_1, x_2\}$ of two secrets, chosen from a universe U of N possible secrets. We now proceed to precisely formulate the algorithmic problem that **B** wishes to (and can hope to) solve (the reader familiar with the paper [CGL01] probably already knows the formal definition, and can skip the next few paragraphs).

Note that **A** can always choose to answer according to the secret x_1, and thus **B** can never hope to learn with certainty more than one of **A**'s secrets. Moreover, a moment's thought reveals that **B** cannot even hope to pin down with certainty one secret and claim that it must be one of **A**'s secrets. This is because **A** could pick there secrets $\{x_1, x_2, x_3\}$ and answer each question asked by **B** according to the majority vote of the answers corresponding to x_1, x_2, x_3. For such a strategy, irrespective of the number of questions **B** asks, **A** can always "justify" any subset of two of these secrets as the set X he picked.

In light of these, it turns out that the best that **B** can hope for is the following: For every set of two *disjoint* pairs of secrets $X = \{x_1, x_2\}$ and $Y = \{x_3, x_4\}$ where the x_i's are all distinct, rule out one of X or Y as the set which **A** picked. An instructive way to visualize this requirement is in terms of graphs. Let K_N denote the complete graph on the universe U of N secrets. View a pair of secrets $X = \{x_1, x_2\}$ as an edge (x_1, x_2) of K_N. A question is simply a function $F : U \to \{0, 1\}$, and the answer to it naturally induces a partition $U = F^{-1}(0) \cup F^{-1}(1)$. If **A** answers question F with a bit $b \in \{0, 1\}$, then we know that the set X which A picked must satisfy $X \cap F^{-1}(1 - b) = \emptyset$, and hence **B** can "eliminate" all edges within the subgraph of K_N spanned by $F^{-1}(1 - b)$. Stated in this language, the goal of **B** is to ask a series of questions by which he can eliminate all edges except those in a set W that contains *no pair of disjoint edges*.

Now, there are only two possibilities for such a surviving set W. Either W must be a "star", i.e., a set of edges all sharing a common x_0, or W must be a "triangle", i.e., the set of three edges amongst a set $\{x_1, x_2, x_3\}$ of three secrets. In the former case, **B** can assert that x_0 must be one of **A**'s secrets. In the latter case, **B** can assert that the secret pair of **A** is one of (x_1, x_2), (x_2, x_3), or (x_3, x_1). In the sequel, when we use the phrase "find the secrets" we implicitly assume that we mean finding the underlying star or triangle as the case may be. We also use the phrase "solve the 2-secrets problem" to refer to the task of finding the underlying star or triangle.

Oblivious vs. Adaptive Strategies: There are two possible strategies that one can consider for **B**: *adaptive* and *oblivious* (also called *non-adaptive*). For adaptive strategies each question of **B** can depend on **A**'s answers to the previous questions. For oblivious strategies **B** must fix the set of questions to be asked right at the outset and be able to infer the secrets just based on **A**'s answers to those fixed questions.

Definitely, adaptive strategies seem more natural for a "20 questions" kind of set-up. However, oblivious strategies have the merit of being easy to play (and being more democratic in terms of different players' abilities), since one just has to read out a fixed predetermined set of questions. Moreover, as we shall see, it is possible to do surprisingly well using just oblivious strategies. In fact, it turns out that there exist oblivious strategies that find the secrets using just $O(\log N)$ questions, which is only a constant-factor off the obvious lower bound of $\log N$ on the number of necessary questions. Moreover, the quest for oblivious strategies runs into some intriguing combinatorial questions, and leads us, quite surprisingly, to list decodable codes! We focus exclusively on oblivious strategies here. (See the work of [CGL01] for some lower and upper bounds on the number of questions required by adaptive strategies.)

A probabilistic construction shows that $O(\log N)$ questions are sufficient to solve the 2-secrets problem [CGL01]. But this only proves the existence of good strategies and the questions are not explicitly specified. In the next section, we discuss how certain binary codes give explicit oblivious strategies.

12.1.2 An explicit strategy with $O(\log N)$ questions

A characterization of oblivious strategies using separating codes

An oblivious strategy for **B** is simply a sequence $\mathcal{F}$ of n Boolean functions (questions) $f_i : [N] \to \{0,1\}$, $1 \le i \le n$. We say a strategy solves the 2-secrets guessing problem if the answers to the questions f_i bring down the possible pairs of secrets to a star or a triangle.

For each secret $x \in [N]$, we denote the sequence of answers to the questions f_i on x by $C(x) = \langle f_1(x), f_2(x), \ldots, f_n(x) \rangle$. We suggestively call the mapping $C : [N] \to \{0,1\}^n$ thus defined as the *code* used by the strategy. There is clearly a one-one correspondence between oblivious strategies $\mathcal{F}$ and such codes C (defined by $f_i(x) = C(x)_i$, where $C(x)_i$ is the i'th bit of $C(x)$). Hence we will from now on refer to a strategy $\mathcal{F}$ using its associated code C.

We say that a code C is $(2,2)$*-separating* (or simply, *separating*) if for every 4-tuple of distinct secrets $a, b, c, d \in [N]$, there exists at least one value of i, $1 \le i \le n$, called the *discriminating index*, for which $C(a)_i = C(b)_i \ne C(c)_i = C(d)_i$. Note that if **B** asks questions according to a separating code C, then for every two disjoint pairs of edges (a, b) and (c, d), **B** can rule out one of them based on the answer which **A** gives on the i'th question, where i is a discriminating index for the 4-tuple (a, b, c, d). In fact it is easy to see that the $(2,2)$-separating property of C is also necessary for the corresponding strategy to solve the 2-secrets guessing game.

This implies the following characterization for the existence of oblivious strategies for the 2-secrets guessing game.

Lemma 12.1 *There exists a* $(2,2)$*-separating code* $C : [N] \to \{0,1\}^n$ *if and only if there exists an oblivious strategy for* $\mathbf{B}$ *using* n *questions that solves the* 2*-secrets guessing problem for a universe size of* N*.*

Hence the problem of finding a small set of questions to solve the 2-secrets problem reduces to the task of finding a good $(2,2)$-separating code. There is a reason why we called these objects "codes" since the following result states that any error-correcting code with a certain property is also a $(2,2)$-separating code. We will assume without loss of generality that $N = 2^m$ so that we can conveniently view each secret as an m-bit binary string. The separating code C then encodes an m-bit string into an n-bit string.

Lemma 12.2 *Let* C *be an* $[n,m]_2$ *binary linear code with minimum distance* d *and maximum distance (i.e., the maximum number of coordinates where two distinct codewords differ) equal to* m_1*. Assume further that* d, m_1 *satisfy the condition* $d > \frac{3m_1}{4}$*. Then,* C *is a* $(2,2)$*-separating code. If the constraint of linearity is removed, then an* $(n,m)_2$ *binary code* C *is* $(2,2)$*-separating if* $d > \frac{m_1}{2} + \frac{n}{4}$*.*

The above lemma is proved in the work of Cohen, Encheva, and Schaathun [CES01]. The result for linear codes had been previously proved by Segalovich [Seg94]. The result for non-linear codes can be strengthened and in fact a code C is $(2,2)$-separating provided $d > n/2$ (cf. [AGKS01]).

There is a big advantage in using linear codes C for $\mathbf{B}$'s strategy, since then each question simply asks for the inner product over GF(2) of the secret with a fixed m-bit string. Thus all questions have a succinct description, which is not the case for general non-linear codes. Hence, we focus exclusively on strategies based on linear separating codes from now on.

Construction of good linear separating codes

Definition 12.1 (ε-biased codes) *A binary linear code of blocklength* n *is defined to be* ε-biased *if every non-zero codeword in* C *has Hamming weight between* $(1/2 - \varepsilon)n$ *and* $(1/2 + \varepsilon)n$*.*

Now Lemma 12.2 implies the following separation property of ε-biased codes.

Corollary 12.3 *If a binary linear code* C *is* ε*-biased for some* $\varepsilon < 1/14$*, then* C *is a* $(2,2)$*-separating code.*

Thus, in order to get explicit $(2,2)$-separating codes (and hence, an explicit strategy for the 2-secrets guessing game), it suffices to explicitly construct an ε-biased code for some $\varepsilon < 1/14$.

A simple explicit construction of ε-biased codes can be obtained by concatenating an outer Reed-Solomon code with relative distance $(1 - 2\varepsilon)$ with an inner binary Hadamard code. It is easy to see that all non-zero codewords have relative Hamming weight between $(1/2 - \varepsilon)$ and $1/2$, and thus this gives an ε-biased space. This construction encodes m bits into $O(m^2/\varepsilon^2)$ bits. Other explicit constructions of ε-biased codes of dimension m and blocklength $O(m^2/\varepsilon^2)$ are also known (cf. [AGHP92]). In fact, the explicit construction of a secret guessing strategy with $O(\log^2 N)$ questions in [CGL01] is based on one of the ε-biased codes from [AGHP92]. All these constructions suffer from the drawback of needing $\Omega(\log^2 N)$ questions, and this means they provide a strategy with $O(\log^2 N)$ questions, while we would like to achieve the optimal $O(\log N)$ questions.

But, there are also known ways to achieve ε-biased codes with blocklength $O(m/\varepsilon^{O(1)})$. For example, one can use a concatenated scheme with outer code any explicitly specified code with relative distance $(1 - O(\varepsilon))$ over a constant alphabet size (that depends on ε), and inner code itself being a Reed-Solomon concatenated with Hadamard code. Specifically, one can use for the outer code the construction from [ABN$^+$92] that achieves rate $\Omega(\varepsilon)$ and alphabet size $2^{O(1/\varepsilon)}$. It is easy to check that this gives an explicit $[O(m/\varepsilon^4), m]_2$ ε-biased code. A better choice of inner code can be used to bring down the blocklength to $O(m/\varepsilon^3)$ [ABN$^+$92], but this is not very important to us since this will only improve the number of questions by a constant factor.

We therefore have:

Lemma 12.4 ([ABN$^+$92]) *For any $\varepsilon > 0$, there exists an* explicitly specified *family of constant rate binary linear ε-biased codes.*

Applying the above with any $\varepsilon < 1/14$, and using the connection to separating codes from Corollary 12.3 and the result of Lemma 12.1, we get the following:

Theorem 12.5 *There is an explicit oblivious strategy for the 2-secrets guessing game that uses $O(\log N)$ questions where N is the size of the universe from which the secrets are drawn.*

12.1.3 An efficient algorithm to recover the secrets

The construction of an explicit strategy using $O(\log N)$ questions is not difficult, and follows rather easily once one realizes the connection to ε-biased spaces. However, a fairly basic and important point has been ignored so far in our description. We have only focused on strategies that "combinatorially" limit the possible pairs of secrets to a star or a triangle. But how can **B** figure out the star or triangle as the case may be, once he receives the answers to all the questions? One obvious method is to simply go over all pairs of secrets and check each one for consistency with the answers. By the combinatorial property of the strategy, we will be left with only a star or a triangle. The disadvantage of this approach, however, is that it requires $O(N^2)$ time. We would ideally like to have a strategy to recover the secrets that runs in $\text{poly}(\log N)$ time, since we would like to have a runtime which is polynomial in the number of bits in the secret. Strategies with such an efficient secret recovery algorithm are called *invertible strategies* in [CGL01]. In [CGL01], the authors mention an invertible strategy for the 2-secrets guessing game, attributed to Lincoln Lu, which uses $O(\log^3 N)$ questions to find the star/triangle in $O(\log^4 N)$ time. Note, however, the number of questions is much larger than $O(\log N)$. The problem of finding an invertible strategy that uses only $O(\log N)$ questions was left unanswered in [CGL01]. Independent of our work, [MS01] answered this question by presenting an *adaptive* invertible strategy using only $O(\log N)$ questions, together with an $O(\log^2 N)$ time algorithm to recover the secrets.

In this section, we present a connection between list decoding and the 2-secrets guessing game. Using this connection, we are able to give an invertible strategy that uses only

$O(\log N)$ questions. The time to recover the secrets (i.e., the triangle or a succinct representation of the star) is $O(\log^3 N)$. We stress that, unlike the result of [MS01], our strategy is *oblivious*, and is therefore incomparable to their result (it is not strictly better because the constants in front of the $\log N$ in the number of questions and the time needed to find the secrets are slightly worse in our construction). Details on the connection to list decoding and our construction follow.

Connection to list decoding

Lemma 12.6 *Suppose that C is a $[cm, m]_2$ binary linear code which is ε-biased for some constant $\varepsilon < 1/14$. Suppose further that there exists a list decoding algorithm for C that corrects up to a fraction $(1/4 + \varepsilon/2)$ of errors in time $O(T(m))$. Then, C is a $(2,2)$-separating code which gives a strategy to solve the 2-secrets guessing game for a universe size $N = 2^m$ in $O(T(\lg N) + \lg^3 N)$ time using $c \lg N$ questions.*[2]

Proof: Let C be a code as in the statement of the lemma and assume that **B** is using C for its strategy. Let $X = \{x_1, x_2\}$ be the set which **A** claims he picked after giving all the answers. Let the set of answers be $\mathbf{a} = (a_1, a_2, \ldots, a_n)$. Then for each i, we must have either $C(x_1)_i = a_i$ or $C(x_2)_i = a_i$ since **A** is supposed to answer each question according to one of x_1 or x_2. Now by the property of C, we have $C(x_1)_i = C(x_2)_i$ for all $i \in A$ for some set $A \subseteq [n]$ of size at least $(1/2 - \varepsilon)n$. For each $i \in A$ we have $C(x_1)_i = C(x_2)_i = a_i$, and for each $i \notin A$, exactly one of $C(x_1)_i$ and $C(x_2)_i$ equals a_i. It follows that either $C(x_1)$ or $C(x_2)$ is within Hamming distance $(n - |A|)/2$ of $\mathbf{a}$; assume without loss of generality that it is $C(x_1)$. Then

$$\Delta(\mathbf{a}, C(x_1)) \leq \frac{n - |A|}{2} \leq (1/2 + \varepsilon)\frac{n}{2} = \Big(\frac{1}{4} + \frac{\varepsilon}{2}\Big)n \ .$$

The algorithm for **B** to recover the secrets (i.e., the triangle or the star) after receiving the answer vector $\mathbf{a}$ is as follows.

1. Perform list decoding of the code C using the assumed algorithm to find the set, say S, of all $x \in \{0,1\}^m$ that satisfy $\Delta(\mathbf{a}, C(x)) \leq (\frac{1}{4} + \frac{\varepsilon}{2})n$.
2. For each $x \in S$ returned by the list decoding algorithm in the previous step, do the following. Compute $A = \{i : C(x)_i = a_i\}$, and perform an *erasure list decoding* of the received word $\mathbf{a}$ when all of its symbols in positions in A are erased. In other words find (some representation of) the set S_x of all x' for which $C(x')_i = a_i$ for each $i \in [n] \setminus A$. If S_x is empty, then remove x from S.
3. Return the set of unordered pairs $\{(x, x') : x \in S, x' \in S_x\}$ as the final set of all possible feasible pairs.

We now argue the correctness of the algorithm. First note that any pair returned by the algorithm is a proper solution to the guessing secrets. This is because the set S_x consists of precisely those secrets that could form the other secret in a pair with x so that the resulting pair will be "consistent" with the answers $\mathbf{a}$. We next prove that any pair (x, x') which

[2]The $\lg^3 N$ component in the runtime comes from the time it takes to solve an $O(n) \times O(n)$ linear system, where $n = \lg N$. We can therefore replace the runtime by $O(T(\log N) + M(\lg N))$ where $M(n)$ is the time taken to multiply two $n \times n$ matrices over GF(2) (which is asymptotically the same as the time required to solve an $O(n) \times O(n)$ linear system over GF(2)). To keep things simple, we simply use an $O(\lg^3 N)$ time bound instead of $O(M(\lg N))$ for this task.

is a consistent solution to the 2-secrets problem for the answers $\mathbf{a}$, will be found by the algorithm. Appealing to $(2,2)$-separation property of C (which is implied by Corollary 12.3 since C is ε-biased for some $\varepsilon < 1/14$), the above two facts imply that the final set of pairs *will* either be a triangle or a star.

If a pair (x, x') is consistent with $\mathbf{a}$, then we know by the initial arguments in this proof that $\min\{\Delta(\mathbf{a}, C(x)), \Delta(\mathbf{a}, C(x'))\} \le (1/4 + \varepsilon/2)n$. Assume without loss of generality that $\Delta(\mathbf{a}, C(x)) \le (1/4 + \varepsilon/2)n$. Then, x will be found as part of the set S is the first list decoding step of the above algorithm. Now for each i such that $C(x)_i \neq a_i$, we *must* have $C(x')_i = a_i$, or otherwise (x, x') would not be a consistent pair for the answers $\mathbf{a}$. Hence x' will be a solution to the erasure decoding performed in the second step. It follows that $x' \in S_x$ and that (x, x') will be output by the algorithm, as desired.

Now we move on to the runtime analysis of the algorithm. By the hypothesis of the lemma, the first list decoding step can be performed in $O(T(m))$ time. Moreover, the size of the list S returned will be bounded by an absolute constant. This follows from the Johnson bound (cf. Theorem 3.1, Chapter 3), which for binary codes states that list decoding to a fraction $\alpha/2$ of errors in a code of relative distance $\delta/2$ when $\alpha < 1 - \sqrt{1-\delta}$, requires lists of size at most $(\alpha^2 - 2\alpha + \delta)^{-1}$. Applying this with $\alpha = 1/2 + \varepsilon$ and $\delta = 1 - 2\varepsilon$, gives that the list size will be at most $(\varepsilon^2 - 3\varepsilon + 1/4)^{-1}$, which is at most 24.5 for $\varepsilon < 1/14$. Hence we will have $|S| \le 24$ and therefore the second erasure decoding step will only be performed for $O(1)$ choices of x.

For the second step we critically use the fact that C is a linear code, and hence erasure list decoding amounts to finding all solutions to a linear system. The set S_x, therefore, is either empty or the coset of a linear subspace, say W_x, of $\mathbb{F}_2^m$, and in the latter case can be represented by one solution together with a basis for W_x. Hence an $O(m^2)$ size representation of each non-empty S_x can be computed in the time needed to solve a linear system, which is certainly $O(m^3)$.

Hence the above algorithm finds either the triangle or the star of all pairs of secrets consistent with the answer vector $\mathbf{a}$ in $O(T(m) + m^3)$ time. Note that in the case when it outputs a star, the number of pairs could be quite large (as high as $(N-1)$ in case the answer vector $\mathbf{a}$ exactly matches $C(x)$ for some secret x). The algorithm exploits the fact that the non-hub vertices the star, being the set of solutions to a linear system, can be described succinctly as the coset of a linear space. $\square$

Remark: We stress here that the use of list decoding in the above application is critical, and the result cannot be obtained using unique decoding. This is because for any pair (x, x') which is consistent with $\mathbf{a}$, we are only guaranteed that one of x or x' is within Hamming distance $(1/4 + \varepsilon/2)n$ from $\mathbf{a}$. Thus, we need to perform decoding to a relative radius of $(1/4 + \varepsilon/2)$. Therefore, if we were to perform unique decoding, we would need a relative distance of $(1/2 + \varepsilon)$, which is of course impossible for binary codes (unless they just have a constant number of codewords which is not very useful). Also, note that after the list decoding algorithm finds the set S of codewords close to $\mathbf{a}$, the application gives a *natural post-processing routine to prune the list* and actually zero down the possibilities to the actual solutions. This will also be a characteristic of several of the applications of list decoding discussed in this chapter.

The final result using specific list decodable codes

We now prove that explicit codes with the property needed in Lemma 12.6 exist, and thus conclude our main algorithmic result about the 2-secrets guessing game. The following result is quite standard and can be proved easily using techniques from Chapter 8 on concatenated codes. The only new element is the requirement of an ε-biased code, but as we shall see this necessitates no significant change in the proof technique.

Lemma 12.7 *For every positive constant $\alpha < 1/2$, the following holds. For all small enough $\varepsilon > 0$, there exists an explicit asymptotically good family of binary linear ε-biased codes of which can be list decoded up to a fraction α of errors in $O(n^2(\frac{\log n}{\varepsilon})^{O(1)})$ time.*

Proof: (Sketch) We only sketch the proof since it is by now quite routine. Given $\alpha < 1/2$, we pick $\varepsilon = O((1/2-\alpha)^2)$. The code construction will be the concatenation of an outer Reed-Solomon code C_{RS} of rate smaller than ε with inner code C_{in} being an explicitly specified $\varepsilon/2$-biased binary linear code (such a code exists by Lemma 12.4). It is clear that the resulting concatenated code, say C, has relative distance at least $(1-\varepsilon)(1/2-\varepsilon/2) > 1/2-\varepsilon$, and maximum relative distance at most $1 \cdot (1/2+\varepsilon/2) < 1/2+\varepsilon$. Hence C is definitely an ε-biased code.

Assume that the Reed-Solomon code be defined over $\mathrm{GF}(2^\ell)$ and has blocklength $n_0 = 2^\ell$. Let the blocklength of C_{in} be n_1. The blocklength of C is then $N = n_0 n_1$. To list decode a received word $\mathbf{r} \in \mathbb{F}_2^n$, we first divide $\mathbf{r}$ into n_0 blocks $r_1, r_2, \ldots, r_{n_0}$ corresponding to the n_0 inner encodings, where each $r_i \in \mathbb{F}_2^{n_1}$. Each of the r_i's is decoded by brute-force to produce a list L_i of all $\zeta \in \mathrm{GF}(2^\ell)$ for which $\Delta(C_{\mathrm{in}}(\zeta), r_i) \le \beta n_1$, for some β where $\alpha < \beta < 1/2$. Since $\delta(C_{\mathrm{in}}) \ge 1/2 - \varepsilon$ and $\alpha = 1/2 - \Omega(\sqrt{\varepsilon})$ (by our choice of ε), it follows using Johnson bounds for list decoding from Chapter 3, for example using Theorem 3.1, that for each i, $|L_i| = O(1/\varepsilon)$. Now if x is such that $\Delta(C(x), \mathbf{r}) \le \alpha N$, then by an averaging argument for at least a fraction α/β of i's in the range $1 \le i \le n_0$, we must have $C_{\mathrm{RS}}(x)_i \in L_i$. Therefore, to finish the list decoding, it suffices to *list recover* the outer Reed-Solomon code to find all x for which $C_{\mathrm{RS}}(x)$ has an element from L_i at the i'th position for at least $\alpha n_0/\beta$ values of i. If the rate of C_{RS} is at most $O(\alpha^2\varepsilon/\beta^2)$, this can be accomplished in $O((n_0/\varepsilon)^2 \log^3 n_0)$ time using the Reed-Solomon list decoding algorithms from Chapter 6 (for example, the version stated in Theorem 6.17). This completes the proof of the lemma. □

Applying the above result with $\alpha = 1/4 + 1/28 = 2/7$ and any $\varepsilon < 1/14$, gives an explicit construction of the codes which were needed in Lemma 12.6, with a quadratic list decoding algorithm. The $O(\lg^3 N)$ time required to perform the "simple, clean-up" erasure decodings in Lemma 12.6 therefore dominates the overall time to recover the triangle or star of secrets. This gives our main result of this section, which achieves the optimal (up to constant factors) number of questions together with a $\mathrm{poly}(\log N)$ algorithm to recover the secrets.

Theorem 12.8 (Main Result on Guessing Secrets [AGKS01]) *For the guessing secrets game between players* **B** *and* **A** *with 2 secrets picked out of a universe of size N, there exists an explicit oblivious strategy for* **B** *to discover the underlying star or triangle of possible pairs of secrets, that requires $O(\log^3 N)$ time and uses $O(\log N)$ questions.*

12.1.4 The case of more than two secrets

One can also consider the situation when the player **A** has $k > 2$ secrets. In this case, stated in the same graph-theoretic language that we used to describe the 2-secrets problem, the goal of **B** would be to find a k-uniform hypergraph H with vertex set being the N secrets with the property that every two hyperedges of H intersect. Let us call such a hypergraph an *intersecting hypergraph.*

12.1.5 Explicit oblivious strategies with $O(\log N)$ questions

Unlike the case of graphs, where there were only two classes of such graphs, namely a triangle or a star, the situation even for 3-regular hypergraphs is much more complicated. Nevertheless, there exist explicit strategies which will allow **B** to "combinatorially" reduce the possibilities to an intersecting hypergraph, even though we do not know any method for **B** to actually find some representation of this hypergraph short of trying out all k element subsets of secrets and pruning out all "inconsistent" ones. This follows from a connection of the k-secrets guessing problem to the study of $2k$-universal families of binary strings. The latter problem concerns finding a subset $S \subset \{0,1\}^N$ of as small size as possible with the property that for every subset of $2k$ indices $i_1, i_2, \ldots, i_{2k}$ and every $(a_1, a_2, \ldots, a_{2k}) \in \{0,1\}^{2k}$, there exists a string $x \in S$ such that $x_{i_j} = a_j$ for each $j = 1, 2, \ldots, 2k$. Explicit constructions of such universal families of very small size, namely at most $c_k \log N$, are known [NN93, ABN$^+$92], where c_k is a constant that depends exponentially on k.

We claim this implies the existence of explicit oblivious strategies using $c_k \log N$ questions for the k-secrets guessing game. Indeed let $\{y_1, y_2, \ldots, y_n\}$ be a $2k$-universal family of N-bit strings for some $n \leq c_k \log N$. For $1 \leq i \leq n$, define the function $f_i : [N] \to \{0,1\}$ as follows: for each $x \in [N]$, $f_i(x)$ is simply the x'th bit of the string y_i. That is, the string y_i gives the truth table of the function f_i. Clearly if the y_i's are explicitly specified then so are the functions f_i. We claim the sequence of questions $f_1, f_2, \ldots, f_n$ is a valid oblivious strategy for the k-secrets guessing game. This is because, for every pair of disjoint sets of k secrets each, say $S_1 = \{i_1, i_2, \ldots, i_k\} \subset [N]$ and $S_2 = \{i_{k+1}, \ldots, i_{2k}\} \subset [N]$, by the $2k$-universality property there exists some i for which $f_i(x) = 0$ for each $x \in S_1$ and $f_i(z) = 1$ for each $z \in S_2$. This implies that the answer to question number i rules out one of the sets S_1 or S_2 as being a possible set of k secrets consistent with all answers to the questions $f_1, f_2, \ldots, f_n$. This is exactly what we wanted to show, and the k-sets of secrets consistent with any answer to the questions $f_1, f_2, \ldots, f_n$ therefore form an intersecting k-uniform hypergraph.

The best known explicit construction of $2k$-universal families, due to [ABN$^+$92], achieves $c_k = ck2^{6k}$ where c is an absolute constant. We therefore have:

Theorem 12.9 *For the k-secrets guessing game over a universe of size N, there exists an explicit oblivious strategy for* **B** *that uses at most $ck2^{6k} \log N$ questions, where c is an absolute constant independent of k.*

12.1.6 An efficient "partial solution" for the k-secrets game

For the case of $k > 2$ secrets, the question of whether there exists a strategy together with an efficient algorithm to actually find a representation of the intersecting hypergraph is wide open, and appears to be rather difficult to solve. We instead aim for the weaker goal of finding a *small* "core" of secrets such that any k-set which **A** might pick must intersect the core in at least one secret. This at least gives useful partial information about the set of secrets which **A** could have picked.

This version of the problem is quite easily solved if we could ask not binary questions, but questions over a larger alphabet $[q]$. That is, each of the n questions that **B** asks is now a function $F_i : [N] \to [q]$, $1 \leq i \leq n$. For any set of k secrets which **A** might pick, the sequence of answers $\mathbf{a} \in [q]^n$ must agree with the correct answers to one of the secrets for at least n/k values of i. If q is a prime power bigger than k, the results of Chapter 8 give explicit constructions of q-ary linear codes, say C, with N codewords and blocklength

$n = O(\log N)$ which are efficiently list decodable from a fraction $(1 - 1/k)$ of errors. Basing the questions F_i on the n positions of the code (as in the earlier binary case), the answer vector $\mathbf{a}$ of $\mathbf{A}$ will differ from at least one secret in $\mathbf{A}$'s set in at most a fraction $(1 - 1/k)$ of positions. The list decoding algorithm, when run on input $\mathbf{a}$, will output a small list that includes that secret.

When $\mathbf{B}$ is only allowed binary questions, we can still give such a "core finding" strategy as follows. Pick q to be a prime power larger than k^2, and C to be an explicit q-ary linear code that is list decodable from a fraction $(1 - 1/k^2)$ of errors (such an explicit code follows from the concatenated code constructions of Chapter 8). As above, we first encode each secret by C to get a string of length $n = O(\log N)$ over $[q]$. We then encode each element of $[q]$ further using $2k$-universal family $\mathcal{F}$ of strings in $\{0,1\}^q$. That is, we encode $i \in [q]$, by the string comprising of the i'th entry from the set of strings in $\mathcal{F}$. In other words, we *concatenate* C with the $2k$-universal family $\mathcal{F}$ to get a binary code C'. Player $\mathbf{B}$ now asks $\mathbf{A}$ for the bits of the encoding of the secret as per the concatenated code C'.

Using the $2k$-universal property of $\mathcal{F}$, $\mathbf{B}$ can recover an intersecting k-hypergraph H_i on $[q]$ for the value of the i'th symbol of the encoding of the k secrets by C. $\mathbf{B}$ can do this by a brute-force search over all k-element subsets of q, and since q, k are constants, this only takes constant time for each i. $\mathbf{B}$ then picks one of the hyperedges E_i from H_i arbitrarily, and then picks an element $a_i \in [q]$ from it at random.

Let $S = \{x_1, x_2, \ldots, x_k\}$ be the set of k-secrets that $\mathbf{A}$ picked. Note that each hyperedge E_i in H_i must intersect the set, say $S_i = \{C(x_1)_i, \ldots, C(x_k)_i\}$, that consists of the i'th symbols of the encoding of the secrets in S by C. Therefore, we will have $a_i \in S_i$ for an expected fraction $1/k$ of i's. An averaging argument that implies that there must exist at least one $x_j \in S$, for some $1 \le j \le k$, for which $a_i = C(x_j)_i$ holds for at least a fraction $1/k^2$ of i's. Therefore, the assumed list decoding algorithm for C on input $\mathbf{a} = \langle a_1, a_2, \ldots, a_n \rangle$ will find a small list that includes the secret x_j. This gives us the following result.

Theorem 12.10 *For the k-secrets guessing game with a universe of N secrets, there exists an explicit oblivious strategy for $\mathbf{B}$ that uses $O(\log N)$ questions. Moreover, there is an efficient $\mathrm{poly}(\log N)$ time algorithm for $\mathbf{B}$ to find a small core of $\mathrm{poly}(k)$ secrets such that the k-set picked by $\mathbf{A}$ must contain at least one secret from the core.*

12.2 Applications to Complexity Theory

The interplay of coding theory and computational complexity theory has had a long and sustained history, and has been a rich source of both problems and results. In particular there are numerous examples of results in complexity theory that make use of, or are inspired by, results in coding theory. Examples include the early work on computation in the presence of noise, and more recent successes like the theory of program testing and correcting, and new characterizations of traditional complexity classed like PSPACE, NEXP and NP in terms in interactive or probabilistically checkable proofs. The survey article of Feigenbaum [Fei95] gives a detailed account of many such uses of results from coding theory in complexity theory.

Here we present applications of error-correcting codes to complexity theory which differ from the above ones in that they crucially rely on the strength of the decoding algorithms, and in particular on the ability to *list decode* from a very large fraction of errors. A related survey article by Sudan [Sud00b] also deals with connections between list decoding and complexity theory. In comparison, our survey is a little more detailed in nature.

12.2.1 Hardcore predicates from one-way permutations

The first work that (implicitly) exploited list decoding for the purpose of a complexity-theoretic application seems to be the seminal work of Goldreich and Levin [GL89], who gave a generic construction of hardcore predicates from an arbitrary one-way function. Implicitly, their work gives a *highly efficient* list decoding algorithm to decode Hadamard codes from up to a fraction $(1/2-\varepsilon)$ of errors.[3] We now discuss how any good list decodable binary code immediately gives such a hardcore predicate. Moreover, viewing the question in the general terms of list decodable codes gives some quantitative improvements in the construction, as was first noticed by Impagliazzo in an unpublished work. Details of this generic connection to list decoding have since appeared in the survey article by Sudan [Sud00b].

The hardcore predicate construction problem lies at the very foundations of cryptography. Formally, the problem is the following. Given an arbitrary one-way permutation $f : \{0,1\}^k \to \{0,1\}^k$ which is easy to compute everywhere but hard to invert by size s circuits even on a small fraction of the range, construct a Boolean predicate $P : \{0,1\}^k \to \{0,1\}$ such that $P(x)$ is very hard to predict given $f(x)$, even for a $(1/2+\varepsilon)$ fraction of the x's, by size s' circuits (for $s' \simeq s$). Intuitively, the existence of such a predicate means that any one-way permutation hides at least one bit, since the knowledge of $f(x)$ does not help at all in predicting the value of $P(x)$.

The above task, as stated, is actually impossible to achieve, since for any predicate P, it is possible to construct one-way permutations f such that $f(x)$ immediately gives away $P(x)$. However, this arises due to the deterministic nature of P, and by allowing P to be a function of x and a random string r, such predicates can indeed be built. In fact any list decodable code with a certain property gives such a construction, which is sketched below.

Let C be an $(n,k)_2$ binary code with $n = \text{poly}(k/\varepsilon)$ that has an efficient encoding algorithm and an efficient (poly(n) time) list decoding algorithm to decode up to a fraction $(1/2-\varepsilon)$ of errors. Then $P(x,r) = C(x)_r$, i.e. the r'th bit of the encoding of x, gives us a predicate with the desired hardness. Indeed, if for some x, a small circuit $\hat{C}$ computes $P(x,r)$ correctly for a $(1/2+\varepsilon)$ fraction of r's, then one can use the assumed list decoding algorithm to decode $\hat{C}$'s output to find a small list of candidates that includes x. Further, the knowledge of $f(x)$ tells us how to find out which element of the list is x. This gives a small circuit to invert $f(x)$ for any x for which $\hat{C}$ predicts $P(x,r)$ correctly for a $(1/2+\varepsilon)$ fraction of r's. The assumed hardness of inverting f now implies that the fraction of such x's must be tiny, and hence $\hat{C}$ does not predict P much better than random guessing.

Finally, we note that several constructions of the code C with the required properties are now known. All of them are based on code concatenation and indeed the results of Chapter 8 imply that such codes exist for $n = O(k/\varepsilon^4)$. If one seeks explicit constructions of the predicate, then one can achieve $n = O(k^2/\varepsilon^4)$ or $n = O(k/\varepsilon^8)$. The exact dependence of n on k, ε does not matter for this application, as long it is polynomial in k and $1/\varepsilon$.

We would like to stress two aspects of the above application. First, the power of list decoding was necessary since we want to decode binary codes up to a fraction $(1/2-\varepsilon)$ of errors. Second, the application gave a natural tie-breaking scheme, namely the value of $f(x)$, to pick the "correct" codeword from the list of candidates output by the decoding

[3] Since a Hadamard code of blocklength n has only n codewords, it is trivial to list decode the Hadamard code up to any radius under the models we have been considering so far. The result of [GL89] works under an "implicit model" where one only has oracle access to the codeword, and gives a poly$(\log n)$ time algorithm to decode the Hadamard code under such a model. We will return to the implicit version of list decoding in further detail in Section 12.2.2.

algorithm. These should provide some indication that list decoding exactly fits the ball for this application.

It is also possible to get hardcore functions that output more than one bit. For a constant number of bits this just involves using q-ary list decodable codes for larger q as opposed to the binary codes used above. Recent work by Ta-Shma and Zuckerman [TZ01] uses extractors to give constructions of codes over very large alphabets which can be used to extract as many as $O(\log^2 k)$ hardcore bits.

In a different construction, Håstad and Näslund [HN98] prove that all bits of $ax+b \bmod p$ give hardcore predicates for any one-way function. Specifically, they prove that if f is any one-way function mapping n bits into n bits, then for each i, $1 \le i \le m$, the predicate $P_i(x; p, a, b)$ defined to be the i'th bit of $(ax+b) \bmod p$ where p is a random m bit prime (for $m = \Omega(\log n)$) and a, b are random numbers modulo p, is a hardcore predicate for f. This result also uses list decoding; specifically it uses an efficient list decoding algorithm for Chinese Remainder codes (similar to those discussed in Chapter 7).

12.2.2 Hardness amplification of Boolean functions

An important area of complexity theory that has benefited greatly from elegant connections to coding theory is the hardness amplification of Boolean functions. The basic question here is the following. Given a Boolean function $f : \{0,1\}^\ell \to \{0,1\}$ with high *worst-case* circuit complexity, i.e., no small circuit can compute f correctly on *every* input, the goal is to "amplify" its hardness and transform it into a Boolean function $f' : \{0,1\}^{\ell'} \to \{0,1\}$ which has very high *average-case* circuit complexity, i.e. no small circuit can compute f' on even an α fraction of inputs, for some α, $1/2 < \alpha < 1$. Ideally, we would like to obtain extreme average-case hardness without blowing up the input length too much. Quantitatively, this means we would like to have $\ell' = O(\ell)$ and α very close to $1/2$ (say, $\alpha = 1/2 + 2^{-\Omega(\ell)}$ — we call such hardness "extreme average-case hardness" in the sequel).

The transformation of worst-case hardness into extreme average-case hardness can be achieved via a two-step process. The first step obtains a predicate with mild average-case hardness (i.e., $\alpha = 1 - \ell^{-O(1)}$) from a worst-case assumption [BFNW93], and is itself inspired by ideas from coding theory. In the second step, the hardness is amplified by using the celebrated Yao's XOR Lemma, which states that a mild average-case hardness can be amplified to extreme average-case hardness by taking the XOR of several independent instances together (cf. [GNW95]). The problem with this approach is that the length of the input of the function f' blows up; specifically we will have $\ell' = \Omega(\ell^2)$.

The big motivation for reducing the input length ℓ' to $O(\ell)$ is the application to derandomization of BPP. Specifically, using a result of Nisan and Wigderson [NW94], such a hardness amplification implies BPP = P under a worst-case hardness assumption (namely that E = DTIME$(2^{O(n)})$ does not have circuits of size $2^{o(n)}$). Such a hardness amplification (with $\ell' = O(\ell)$ and $\alpha = 1/2 + 2^{-O(\ell)}$) was achieved by a striking result due to Impagliazzo and Wigderson [IW97]. The transformation involved three steps: the worst-case to mild hardness transformation due to [BFNW93], a first derandomized XOR Lemma due to Impagliazzo [Imp95], and a second derandomized XOR Lemma due to [IW97] (the derandomized XOR lemmas amplify the hardness while blowing up the input length only by a constant factor).

In one of the most striking applications of list decoding to complexity theory, Sudan, Trevisan, and Vadhan [STV99] show that, using certain very efficiently list decodable codes, one can achieve the above hardness amplification in a *single* step, without any need for complicated XOR lemmas. They require codes $C_{k,\varepsilon}$ which encode k bits into $n = \text{poly}(k, 1/\varepsilon)$ symbols (assume n is a power of two), which can be encoded in $\text{poly}(n)$ time, and which can be list decoded in $\text{poly}(\log k, 1/\varepsilon)$ time from up to a fraction $(1/2 - \varepsilon)$ of errors. This is similar to the codes we required for the application to hardcore predicate construction, except that we now allow the list decoder only time which is polynomial in $\log k, 1/\varepsilon$ (and not the "more reasonable" $\text{poly}(k, 1/\varepsilon)$ time).

To obtain the desired hardness amplification using such a code, we treat the truth table of f as a $k = 2^\ell$ bit string, and treat its encoding it by $C_{k,\varepsilon}$, which is of length $n = \text{poly}(k/\varepsilon) = 2^{O(\ell+\log(1/\varepsilon))}$, as a function $f' : \{0,1\}^{\ell'} \to \{0,1\}$ where $\ell' = O(\ell+\log(1/\varepsilon))$. Thus we can pick $\varepsilon = 2^{\Omega(\ell')}$ and still have $\ell' = O(\ell)$. Using the list decoding properties of $C_{k,\varepsilon}$, it is easy to prove that a circuit for f' that computes it correctly on a fraction $(1/2+\varepsilon)$ of inputs implies a slightly larger circuit that computes f correctly on every input. The assumed worst-case hardness of f then immediately implies the extreme average-case hardness of f'.

Of course the requirement on our code $C_{k,\varepsilon}$ is fairly severe, since we want a list decoding algorithm that runs in time which is sub-linear in k. In particular, this implies that there is not even enough time to scan the entire received word before decoding it, or to output the entire message after decoding is complete! This seems impossible to achieve. But by both specifying the input received word "implicitly" and allowing the output message to also be specified "implicitly", it becomes possible to decode in time sub-linear in the blocklength. The implicit representation of the input just means that there exists an oracle which when queried with index i responds with the i'th bit of the received word. The implicit representation of the output message is more tricky to define, but loosely speaking, we require the decoding to output a program, which on input j will output the j'th bit of the message. The exact definition allows these programs to be randomized and is a little more subtle, and we point the reader to [STV99] for further details.

Finally, the authors of [STV99] are also able to construct such a code $C_{k,\varepsilon}$ with a $\text{poly}(\log k, 1/\varepsilon)$ list decoding algorithm in the above implicit model. Their construction is based the concatenation of a Reed-Muller code with the Hadamard code, and a highly efficient list decoding algorithm for Reed-Muller codes in the implicit model. The first such list decoding algorithm for Reed-Muller codes was due to Arora and Sudan [AS97], where they used the list-decoding algorithm/algebra to reduce the "low-degree testing question", which is an important one in complexity theory, to an analysis of a version of Hilbert's test for irreducibility of multivariate polynomials. While the algorithm from [AS97] would have sufficed for the result from [STV99] stated below, [STV99] also gives a simpler list decoding algorithm for Reed-Muller codes that corrects more errors. After concatenation with an Hadamard code, they are able to prove:

Theorem 12.11 ([STV99]) *For every q, ε, k, if $n \geq \text{poly}(k, q, 1/\varepsilon)$, then there exists an explicitly specified $[n, k]_q$ linear code with a polynomial time list decoding algorithm for up to a fraction $(1 - 1/q - \varepsilon)$ of errors. Furthermore, the algorithm runs in* $\text{poly}(\log k, q, 1/\varepsilon)$ *time if both the input and the output are specified implicitly.*

(The case $q = 2$ is described explicitly in [STV99]; the case of larger q is stated explicitly in [Sud00b] and it can also be inferred from the proof in [STV99].)

Using the above code construction and the above-mentioned connection to hardness amplification, [STV99] obtain an elegant *one-step* hardness amplification of Boolean functions, which is strong enough to imply BPP = P under a worst-case hardness assumption.

12.2.3 Average-case hardness of Permanent

One of the first results to establish the average-case hardness of a problem that is believed to be very hard on the worst-case was Lipton's striking discovery of the random self-reducibility of the permanent modulo a large enough prime. Specifically, Lipton [Lip91] showed the following result: if it is easy to compute the permanent of an $n \times n$ matrix modulo a prime $p > n$ with probability $1 - O(1/n)$, then it is also easy to compute the permanent of *every* $n \times n$ matrix modulo p. This important result laid down the seed for a number of interesting results in complexity theory, including IP = PSPACE and the PCP characterizations of NP.

Subsequent results attempted to obtain the same (or similar) consequence as Lipton's result but assuming algorithms which only worked for a much smaller fraction of random matrices. For example, the work of Feige and Lund [FL92] showed that the existence of an efficient procedure to compute the permanent of $n \times n$ matrices over $\mathbb{F}_p$ on even an exponentially small fraction of matrices would imply that $\mathrm{P}^{\#\mathrm{P}} \subseteq \mathrm{AM}$, and in particular that the polynomial hierarchy collapses. Therefore, such an algorithm is unlikely to exist. Here, we would like to point out that their proof implicitly proves and uses a certain combinatorial list decodability property of Reed-Solomon codes.

Cai, Pavan and Sivakumar [CPS99] strengthened the result of Lipton by showing that it suffices to assume an algorithm that works for an inverse polynomial fraction of matrices, provided the prime p is sufficiently large. Their result uses the efficient list decodability of Reed-Solomon codes from a large fraction of errors (and builds upon ideas from an earlier paper by Gemmell and Sudan [GS92], who obtained a weaker result using unique decoding of Reed-Solomon codes). In [GRS99], it was shown that if there exists an algorithm for computing the permanent with an inverse polynomial success probability when both the matrix *and* the prime are picked at random, then $\mathrm{P}^{\#\mathrm{P}} \subseteq \mathrm{BPP}$. This result uses list decoding algorithms for both the Reed-Solomon code and the Chinese Remainder code. Cai *et al.* [CPS99] later observed that it is possible to obtain the same results as [GRS99] using their techniques, thus avoiding the use of Chinese Remainder decoding. But the use of efficient Reed-Solomon list decoding remains crucial to these applications.

12.2.4 Extractors and Pseudorandom Generators

Explicit constructions of combinatorial objects which exhibit strong pseudorandom properties are desirable for numerous applications in computer science. Examples of such objects include expanders, dispersers, extractors, and pseudorandom generators. Recently various non-trivial interconnections have been found between some of these seemingly disparate objects. One of the common threads underlying these results has been the crucial role played by list decoding and constructions of good list decodable codes. We now review some of these inter-relations between extractors, pseudorandom generators, and list decodable codes. We first give brief, informal definitions of extractors and pseudorandom generators.

Extractors are functions which take two inputs, say x and y, where x comes from a weak source of randomness (which has a certain "min-entropy"), and y is a much smaller string of truly random bits. When fed with inputs from such distributions, the output of the extractor should be statistically close to uniform. Intuitively, an extractor is a function that "extracts" almost truly random bits from the output of weak random source using a small number of additional truly random bits as a catalyst. We refer the reader to the survey by Nisan [Nis96] for further details on extractors.

Pseudorandom generators are deterministic functions which take as input a small "seed" and expand it into a much longer string. The crucial property of pseudorandom generators is that when the input seed is completely random, the output of the pseudorandom generator is *computationally indistinguishable* from a truly random string of the same length. Intuitively, a pseudorandom generator stretches a small random string into a much longer one which appears random to and "fools" any computationally-bounded adversary. Such a generator which creates randomness out of nowhere provably cannot exist in the information-theoretic setting, but exists in the computational setting (assuming the existence of one-way functions, for example). We refer the reader to [Gol99, Chap. 3] for further definitions and background on pseudorandom generators.

Extractors from codes

Explicit constructions of extractors are of great interest and have a wide variety of applications. Initial constructions of extractors all relied on properties offered by various families of hash functions. In a departure from these approaches, Trevisan [Tre99] gave a breakthrough construction of extractors by combining a pseudorandom generator of Nisan and Wigderson [NW94] together with binary codes with good *combinatorial list decoding* properties. The intriguing aspect of this result was the use of pseudorandom generators, originally intended to work only in a computational setting, to derive a purely information-theoretic result. The use of a list decodable code as in Trevisan's construction is also part of the numerous improvements to Trevisan's extractor that have since been obtained.

Very recently, a different, more direct, algebraic approach to constructing extractors was found by Ta-Shma, Zuckerman, and Safra [TZS01]. Their results were later improved by Shaltiel and Umans [SU01]. In this construction, the string sampled from the weak random source is viewed as a multivariate polynomial over a finite field and the seed is viewed as a random evaluation point. In coding-theoretic terms, the construction can be viewed as encoding the input from the weak source by a code obtained by concatenating a Reed-Muller code with a "suitable" binary code, and then selecting a (carefully chosen) subset of bits of the encoding as the output (the actual subset chosen is governed by the random seed). The binary code used is again one with good *combinatorial list decoding* properties. In addition to the use of list decodable codes in the construction itself, the proof of the extractor property also critically makes use of the combinatorial list decoding property of Reed-Solomon codes. In fact, codes are more inherent to and more deeply exploited in these constructions than that of Trevisan, where the use of pseudorandom generators was the most crucial component.

Explicit constructions of codes with *efficient* (as opposed to just combinatorial) list decodability is crucial to the work of Mossel and Umans [MU01], who use such codes to construct "extractor-like" objects, namely zero-error dispersers for certain generalized bit-fixing sources. (Such "dispersers" are used in [MU01] to prove the Σ_3-hardness of approximating the VC-dimension to a factor even slightly better than 2.)

Codes from extractors

The above applications exploit codes and list decoding to construct extractors (or extractor-like objects). Recently, Ta-Shma and Zuckerman [TZ01] showed a result in the opposite direction, namely that extractors give codes over large alphabets with good combinatorial list decoding properties. However, in general it is not clear how to match the combinatorial list decoding potential of the code with an efficient list decoding algorithm. For the specific case of Trevisan's extractor, [TZ01] were able to obtain an efficient list decoding algorithm. While the parameters of such a code are not very interesting from a coding point of view (the alphabet size is huge and the rate is extremely small), they applied such extractor codes to constructions of hardcore functions that output many bits. In order to give an efficient algorithm, [TZ01] needed the underlying binary code used by Trevisan's construction itself to have an *efficient list decoding* algorithm to correct close to a fraction $1/2$ of errors. Therefore, this application, even though its final output is a "new" list decodable code, needs an *efficiently* list decodable binary code to start with.

Pseudorandom generators from codes

In their paper, Shaltiel and Umans [SU01] also modify their extractor construction based on Reed-Muller codes to construct a new pseudorandom generator, directly based on a worst-case hardness assumption for functions in $\mathrm{E} = \mathrm{DTIME}(2^{O(n)})$. Their construction matches the parameters of the construction of [IW97, STV99], and in particular is strong enough to prove that BPP = P under a worst-case hardness assumption for some function in E. It is also the first such construction that does not use the Nisan-Wigderson generator.

We point out here that the new pseudorandom generator construction of [SU01] requires *efficiently* list decodable binary codes, as well as efficient list decodability of Reed-Solomon codes. Recall that pseudorandom generators were crucial to Trevisan's extractor, which in turn were used in [TZ01] to construct good list decodable codes (at least over very large alphabets). Therefore, it is interesting to note that this application of list decodable codes to a pseudorandom generator construction "returns the favor" to pseudorandom generators, and completes a full circle!

12.2.5 Membership comparable sets

List decoding also has applications to "membership comparability" of NP-complete languages. A language A is said to be $k(n)$ membership comparable if there is a polynomial time computable function that, given $k(n)$ instances of A of length at most n, excludes one of the $2^{k(n)}$ possibilities for memberships of the given strings in A.

The motivation for studying membership comparability is two-fold. First, membership comparability investigates whether even the least amount of information about the k-wise direct product of a hard function can be computed. To elaborate with an example, if SAT is the characteristic function of the NP-complete language of satisfiable CNF formulae, then the k-fold direct product SAT^k, which is a function that takes a vector of k formulae $\vec{\phi} = \langle \phi_1, \phi_2, \ldots, \phi_k \rangle$ and outputs the k-bit vector $\langle SAT(\phi_1), \ldots, SAT(\phi_k) \rangle$, is presumably much harder to compute. Membership comparability asks if even the least amount of information about SAT^k can be computed efficiently. That is, given an instance vector $\vec{\phi}$, can one rule out at least one of the 2^k k-bit vectors as being *not equal* to $SAT^k(\vec{\phi})$?

Second, it generalizes the notion of *p-selectivity*, which in turn has found many applications in complexity theory. A language A is said to be p-selective if there is a polynomial time computable function f such that for all x, y, $f(x, y) \in \{x, y\}$ and $f(x, y) \in A$ whenever at least one of x, y is in A. Informally, the selector function tells which of its two input strings is "more likely" to be a string that belongs to the language A. It is easy to see that p-selective sets are a special case of $k(n)$ membership comparable sets when $k(n) = 2$, where given instances x, y of A, the membership comparator always excludes one of the possibilities $(0, 1)$ or $(1, 0)$ for the sequence $(\chi_A(x), \chi_A(y))$.

We refer the reader to the paper of Sivakumar [Siv99] for an excellent discussion of membership comparable sets, their motivation and role in complexity theory, and further pointers. For us, we just would like to mention the one connection to p-selective sets that motivates the problem solved here. It is known that if a language A is polynomial time truth table reducible to a p-selective set, then A is $O(\log n)$ membership comparable. Sivakumar [Siv99] proves the nice hardness result that if SAT is $O(\log n)$ membership comparable, then NP = RP. Through the connection to p-selective sets, this implies an alternative proof of the fact that if SAT is reducible to a p-selective set by polynomial truth-table reductions, then NP = RP. This proof was, however, more complicated than the original proofs of [Tod91, Bei88]. Below we give a somewhat easier proof of Sivakumar's result which works by simply appealing to the existence of certain list decodable codes.

Theorem 12.12 ([Siv99]) *If* SAT *is $O(\log n)$ membership comparable, then* NP = RP.

Proof: Suppose there exists a constant d such that SAT is $d \log n$ membership comparable. We will prove that UniqueSAT $\in P$ under this assumption. Together with the randomized reduction from SAT to UniqueSAT [VV86], this implies the claim of the theorem.

Let ϕ be an instance of UniqueSAT on n boolean variables. Define $p = d \log n$ and $q = 2^p = n^d$. Let $\mathbf{C}$ be an $[N, n]_q$ linear code of dimension n and blocklength N which can be list decoded in poly(n) time from a fraction $\left(1 - \frac{1}{q-1}\right)$ of errors. An explicit construction of such a code with $N = \text{poly}(n)$ can be obtained using a suitable concatenated code (for example one can apply Theorem 12.11 with the settings $k = n$, $q = n^d$ and $\varepsilon = n^{-2d}$).

For each i, j, where $1 \le i \le N$ and $1 \le j \le p$, construct a collection of $p = d \log n$ SAT formulae $\phi_{i,j}$ over n variables as follows: For each $a \in \{0,1\}^n$,

$$\phi_{i,j}(a) \stackrel{\text{def}}{=} (\phi(a) \wedge \text{The } j\text{th bit of } \mathbf{C}(a)_i \text{ equals } 1) \ .$$

(Here $\mathbf{C}(a)_i$ stands for the i'th symbol in the encoding of a as per the code $\mathbf{C}$, and we view elements of GF(2^p) as p-bit vectors using some fixed basis GF(2^p) over GF(2).)

Suppose ϕ were satisfiable (in case it is not, we will never find a witness, so we only worry about the satisfiable case), and let a be the *unique* satisfying assignment to ϕ. We use the polynomial membership comparator function f guaranteed by the hypothesis, to get, for $1 \le i \le N$, vectors $b_i = f(\phi_{i,1}, \ldots, \phi_{i,p}) \in \{0,1\}^p$ such that $b_i \ne (\chi_{SAT}(\phi_{i,1}), \ldots, \chi_{SAT}(\phi_{i,p}))$. By the definition of ϕ_{ij} and the fact that a is the unique satisfying assignment to ϕ, we can conclude, for $1 \le i \le N$, that b_i when viewed as an element of GF(q), is not equal to $\mathbf{C}(a)_i$. Thus we have a word $(b_1, b_2, \ldots, b_N) \in \text{GF}(q)^N$ with **all** symbols in disagreement with the codeword $\mathbf{C}(a)$.

Now, for each i, $1 \le i \le N$, if we pick r_i at random from GF$(q) \setminus \{b_i\}$, then $\mathbf{r} = \langle r_1, r_2, \ldots, r_N \rangle$ will have expected Hamming distance at most $(1 - \frac{1}{q-1})N$ from $\mathbf{C}(a)$. This procedure can be easily derandomized to find $\mathbf{r}$ which is guaranteed to be within Hamming distance $(1 - \frac{1}{q-1})N$ from $\mathbf{C}(a)$. Now we can run the list decoding algorithm for $\mathbf{C}$ on input $\mathbf{r}$, and the list $\mathcal{L}$ that is output will contain the string a. One can then go over all strings in $\mathcal{L}$ to see if any of them satisfy the formula ϕ. If we find such a string, we accept ϕ, otherwise we reject ϕ. This would give a polynomial time procedure to decide membership in UniqueSAT, as desired. $\square$

Once again note that the application above provided a nice tie-breaking criterion, namely the satisfiability of the formula ϕ, to pick the "correct" element from the list output by the list decoding algorithm.

12.2.6 Inapproximability of NP witnesses

For an NP-complete language, say L, given a string $x \in L$, we know that it is unlikely that a polynomial time algorithm can find a witness for the membership of x in L. But can a polynomial time algorithm compute a non-trivial "approximation" to the witness?

There are various ways in which one can formalize the notion of "approximation". For example, one can compute a small portion of some true witness. Or, one can compute a string which agrees with the witness is a non-trivial fraction of positions. These notions were first studied by [GHLP99] and [KS99]. In this section, we use list decoding to prove hardness results for even approximately computing NP witnesses.

The models

The work of Gál, Halevi, Lipton, and Petrank [GHLP99] considered the first model where the goal is to correctly compute a small portion of the witness. For several important NP-complete problems, the authors of [GHLP99] proved that computing even a small fraction of the witness is as hard as finding the full witness. For example, by using codes uniquely decodable up to a fraction $(1/2-\varepsilon)$ of erasures, they proved that computing a $(1/2+\varepsilon)$ fraction of any satisfying assignment of a CNF formula is NP-hard. By an *implicit* use of codes with very good *list decodability* from erasures, they also proved that, for any $\gamma > 0$, given a SAT instance on N variables, computing an assignment to $N^{1/2+\gamma}$ variables which can be extended to a full satisfying assignment to all N variables, is NP-hard *under randomized reductions*.

The realization that the application in [GHLP99] really calls for codes *list* decodable from a large number of erasures was made in [KS99]. There is, however, one subtle and important difference between the models used by [GHLP99] and [KS99], which we highlight below. Every language L in NP comes with a polynomial-time decidable *witness predicate* R_L such that $L = \{x : \exists w,\ (|w| = |x|^c) \wedge R_L(x,w)\}$. Moreover, there is often a "natural" choice for the witness predicate R_L. For example, for SAT, the witness is just a satisfying assignment and the witness predicate, on input (ϕ, w), simply checks if the assignment w satisfies the formula ϕ. Gál *et al.* [GHLP99] wanted to map an instance x of L into another instance y of L, and then argue that partial computation of some witness for membership of y in L for the predicate R_L, enables the computation of an entire witness for membership of x in L, *for the same witness predicate* R_L. In [KS99], the authors allowed a different, not so natural, witness predicate R'_L, and related the partial computation of a witness for R'_L with the computation of a full witness for the "natural" predicate R_L. This implies that there is a formulation of every NP language via *some* (unnatural) witness predicate R'_L for which computing a partial witness or approximating a correct witness is NP-hard. Naturally, results under the model of [KS99] are easier to obtain. In fact, as observed by [KS99], there a simple, general transformation of predicates using list decodable codes that does the job in the latter model. This is outlined next.

The connection to list decoding

Let R_L be a witness predicate for a language $L \in$ NP. Let $\mathcal{C}$ be a family of good list decodable binary codes, with a code C_i of dimension i for every $i \geq 1$. Suppose that each code C_i is list decodable from a fraction $(1-\varepsilon)$ of erasures in poly(i) time. We use

$\mathcal{C} = \{C_i\}_{i \geq 1}$ to define a predicate R'_L from R_L as follows:

$$R'_L(x,z) = \Big[(|z| = |x|^d) \wedge (\exists y)\ [R_L(x,y) \wedge (|y| = |x|^c) \wedge (z = C_{|y|}(y))]\Big] .$$

Now, suppose $x \in L$ with $|x| \in \{0,1\}^n$, and that there is a polynomial time procedure to compute a string $\hat{z}$ comprising an ε fraction of the symbols in a witness z for which $R'_L(x,z)$ holds. Let $N = n^c$ and let $y \in \{0,1\}^N$ be such that $C_N(y) = z$. Then, one can run the polynomial time erasure list decoding algorithm for C_N on input $\hat{z}$, to compute, in $\text{poly}(N) = \text{poly}(n)$ time, a polynomial-sized list $\{y_1, y_2, \ldots, y_\ell\} \subset \{0,1\}^N$ which includes the witness y. We can then run through the elements in this list and for each y_i check if $R_L(x, y_i)$ holds, and if so, output it as the witness that $x \in L$ (for the witness predicate R_L). This gives a polynomial time algorithm to compute an entire witness for the predicate R_L, given only an ε fraction of some witness for the predicate R'_L.

By using a code family $\mathcal{C}$ which is list decodable from a large fraction of *errors* (as opposed to erasures), we can similarly deduce the hardness of computing an "approximate witness", i.e., computing a string with non-trivial agreement with some witness.

Exploiting this connection, Kumar and Sivakumar [KS99] presented various results in this vein. But the codes they used were not the best possible, and later the author and Sudan [GS00] constructed better list decodable codes. Using these better codes, they were able to improve the results of [KS99]. The results they obtain are formally stated below; the proofs may be found in [GS00].

Theorem 12.13 ([GS00]) *For every $\gamma > 0$ the following holds. For every language L in NP, there exists a polynomial time decidable witness predicate R'_L such that for every $x \in L$, given any $|z|^{1/2+\gamma}$ bits of an unknown witness z that satisfies $R'_L(x,z)$, one can, in* $\text{poly}(|x|)$ *time, compute a witness z' that satisfies $R'_L(x,z')$.*

Theorem 12.14 ([GS00]) *For every $\gamma > 0$ the following holds. For every language L in NP, there exists a polynomial time decidable witness predicate R'_L such that for every $x \in L$, given an arbitrary string of length N which agrees with some (unknown) N-bit witness z that satisfies $R'_L(x,z)$ in at least $N/2 + N^{3/4+\gamma}$ positions, one can, in* $\text{poly}(|x|)$ *time, compute a witness z' that satisfies $R'_L(x,z')$.*

Hardness of approximating "natural" NP witnesses

While the results of Theorems 12.13 and 12.14 provide strong hardness results for approximate witness computation for NP languages, they have one shortcoming. They assume we have enough freedom in how to encode witnesses. Specifically, the predicates R'_L for which the stated results hold may not be the "natural" ones for the language. For example, for SAT, it would be desirable to get a result similar to the above results when the witness is a satisfying assignment of the CNF formula and R'_L is the natural predicate that simply checks if the witness assignment satisfies the input CNF formula. This model was the original focus of [GHLP99]. But the notion of a "natural witness" cannot be formalized in any generality, and hence results in the vein of [GHLP99] have to be on a problem-to-problem basis. Nevertheless, it is a worthwhile question to study at least for the most fundamental NP-complete problems. Such a study was recently undertaken by Feige, Langberg and Nissim [FLN00], who built upon and improved the results in [GHLP99].

Using good binary codes *efficiently* list decodable from up to a fraction $(1/2 - \varepsilon)$ of errors (for example any of the concatenated codes we discussed in Chapter 8), they proved the following result on the witness inapproximability of SAT.

Theorem 12.15 ([FLN00]) *For every $\varepsilon > 0$, the following holds. Given a satisfiable* 3SAT *instance ϕ on n variables, the problem of finding a string of length n that agrees with some satisfying assignment of ϕ on at least $(1/2+\varepsilon)n$ variables, is NP-hard.*

Actually one can have the above result with $\varepsilon = n^{-\gamma}$ for some $\gamma > 0$. In addition to being interesting in its own right, as noted in [FLN00], the above result is also important due to a recent randomized algorithm for finding a satisfying assignment for 3SAT formulas due to Schöning [Sch99]. This algorithm has a runtime of $O((4/3)^n)$, which is the best currently known for solving 3SAT. The runtime of Schöning's algorithm can be further improved if there were a polynomial time algorithm to compute an initial assignment that agrees with some satisfying assignment on a $(1/2+\varepsilon)$ fraction of the variables for some constant $\varepsilon > 0$. The above result pretty much rules out the prospect of improving the algorithm in this manner.

A similar result for the hardness of finding partial satisfying assignments, stated below, was proved earlier by [GS00], using techniques from [GHLP99] together with good erasure list decodable codes.

Theorem 12.16 ([GS00]) *For every $\gamma > 0$, the following holds. Given a satisfiable* 3SAT *instance ϕ on n variables, the problem of finding an assignment to some subset of $n^{3/4+\gamma}$ variables such that the partial assignment can be extended to a satisfying assignment, is NP-hard.*

The authors of [GHLP99] had obtained the above result even for an assignment to $n^{1/2+\gamma}$ variables, but they only proved NP-hardness under randomized reductions (this was because they used probabilistic constructions of certain erasure list decodable codes, while Theorem 12.16 uses explicit constructions of such codes with slightly worse parameters).

Similar to 3SAT, [FLN00] proves hardness results for approximating witness for several canonical NP-complete problems to within a factor even slightly better than what random guessing would achieve. As an example, we state their result for 3-COLORING below. This result uses the existence of ternary codes that are efficiently list decodable from up to the "maximum possible" fraction $(2/3-\varepsilon)$ of errors.

Theorem 12.17 *For every $\varepsilon > 0$, the following holds. Given a 3-colorable graph G on n vertices, the problem of finding a 3-coloring of its vertices which agrees with some proper 3-coloring of G on at least $(1/3+\varepsilon)n$ vertices, is NP-hard.*

12.3 Applications to Cryptography

The hardcore predicate construction problem that we discussed in Section 12.2.1 as a complexity-theoretic application of list decoding is also at the very foundations of modern cryptography. Below we discuss some other cryptographic applications of list decoding.

12.3.1 Cryptanalysis of certain block ciphers

Block ciphers are constructs used to securely permute a block of bits (of certain convenient size). Mathematically, a block cipher is a collection of permutations mapping plaintexts into ciphertexts, each of which is determined by a "key". Knowledge of the key enables efficient computation of both the permutation and its inverse. The ideal security condition of a block cipher demands that without knowledge of the key a polynomial-time bounded adversary with oracle access to both directions of the permutation is unable to distinguish the cipher from a truly random permutation on the same message space.

A number of block ciphers used in practice encode the plaintext in several rounds. At each round a certain round function (possibly using its own "round key") is applied, and the output of one round function is the input to the next round function. The security of the overall block cipher generally improves with the number of rounds, but a larger number of rounds also means larger keys and less efficient enciphering algorithms. For some block ciphers, the round function can be described by a low-degree polynomial for a non-negligible fraction of its input values. Some simple round functions of this nature in fact provide very good security against the common forms of attacks involving differential and linear cryptanalysis, even for relatively few rounds.

Nevertheless, intuitively such ciphers do appear to be weak, since the existence of a non-trivial algebraic relation between the plaintext and the ciphertext would make the round function appear quite non-random, and it is not clear this will be alleviated by taking several rounds of such functions. But there was no formal attack which corroborated this intuition. Recently, Jakobsen [Jak98] exposed the weakness of such block ciphers using techniques from coding theory, and in particular list decoding. Specifically, he used the Reed-Solomon list decoding algorithms to break several rounds of block ciphers whose round functions have even a small agreement with low-degree polynomials. In particular, he was able to break up to 10 rounds of a construction by Nyberg and Knudsen that was provably secure against differential and linear cryptanalysis. This represents an interesting use of list decoding in cryptanalysis, and on the flip side provides useful new design criteria for block ciphers. In particular, it indicates that good properties against differential or linear attacks alone is not enough, and that one must avoid using round functions which are algebraically very simple. For further details on the details of the cryptanalysis, we refer the reader to the original article [Jak98].

12.3.2 Finding smooth integers

An integer N is said to be s-smooth if all its prime factors are smaller than s. The problem of finding smooth integers in a given interval is important since common factoring algorithms such as the quadratic sieve work by searching for smooth integers. Actually these algorithms search for integers $x \in [-B, B]$ for which $f(x)$ is s-smooth, where f is some low-degree polynomial over the integers, and B, s are suitable parameters used by the algorithm. For example, the quadratic sieve algorithm, on input N, uses polynomials of the form $f_a(x) = (x + \lfloor\sqrt{aN}\rfloor)^2 - aN$ for some small values of a. It uses a technique called "sieving" to find integers $x \in [-B, B]$ such that $f_a(x)$ is s-smooth. Further details about the quadratic sieve algorithm may be found, for example, in [LL90].

By generalizing the list decoding algorithm for Chinese Remainder codes, Boneh [Bon00] showed how to solve the above problem in polynomial time for certain settings of parameters. As a side consequence, the upper bound on list size also implies interesting upper bounds on the number of integers that can lie in intervals of certain size and have a large smooth factor. However, the current bounds on CRT decoding turn out to be inadequate for improving the basic quadratic sieve. The main problem is that random intervals of the length for which CRT decoding applies are too small to contain sufficiently many smooth integers. In order for this line of research to have hopes of improving the best known factoring algorithms, a significant improvement to CRT decoding would be required. Specifically one would need a version of CRT decoding which works for *random intervals* of much larger length by exploiting the fact that such an interval, unlike a possibly *worst-case interval* of similar size, will not contain too many smooth integers.

12.3.3 Efficient traitor tracing

Consider the situation where a set of authorized users/subscribers have access to a paid service via their own "keys" (which can be bought from the subscription provider). A good example is the distribution of digital information over a broadcast channel (like encrypted pay-TV programs), where each subscriber has access to a "decoding box" that contains a secret decryption key. In this situation, nothing prevents an authorized subscriber from simply distributing his/her secret key to other users, thus making the "paid" service freely available also to non-subscribers. One possible approach to combat this, which motivates and underlies the rationale of *traitor tracing*, is to distribute a set consisting of several keys to each subscriber in such a manner that the set of keys not only enables the subscriber to decrypt, but also *identifies* the subscriber. The threat of an illegitimately distributed key being *traced back* to the owner of that key would then serve as a deterrent to the subscribers from involving in piracy.

In general, traitor tracing tries to do even better and deal with the situation where a set of c subscribers collude together and combine portions of their respective sets of keys in a clever way in order to create a new set of keys, which can still decrypt, but which hopefully (from their perspective!) cannot be traced back to any one of them. The phrase traitor tracing itself was first coined in [CFN94], and since then there has been a lot of research on the design of good traitor tracing schemes.

The design of a traitor tracing scheme is a two-fold process. The first task is combinatorial and the goal here is to choose the set of keys that will be distributed to each authorized subscriber. The second task is algorithmic and deals with the actual tracing of traitors involved in a collusion that produces a fake set of keys. Here, the usual assumption is that the number of users involved in the collusion is fairly small, and tracing schemes are designed to deal with collusions of up to some size c. The ideal goal would be that the tracing algorithm finds and implicates *all* members of the collusion and also *only* members involved in the collusion. But one can also relax this requirement to that of finding at least one member of the collusion (as that itself serves as a reasonable deterrent), and possibly allow for a small probability of accusing an innocent subscriber.

We now come to the connection to list decoding. We will only sketch this connection at a high level, and point the reader to the relevant articles for further details. By assuming that the set of keys distributed to each subscriber is ordered, one can construct tracing schemes with the necessary combinatorial property by deriving the keys from error-correcting codes with large minimum distance. Thus, roughly, each subscriber has some codeword for her set of keys. Then, given a pirated set of keys, viewing that as a noisy received word and performing list decoding, gives a list of codewords which includes at least one of the codewords corresponding to colluding subscribers, and hopefully no spurious codewords corresponding to innocent subscribers.

Silverberg, Staddon and Walker [SSW00] observed that the above connection, when applied to constructions based on Reed-Solomon, algebraic-geometric, and concatenated codes (with outer Reed-Solomon or AG-codes and inner Hadamard code), gives very good traitor tracing schemes. Their result crucially exploits the list decoding algorithms discussed in this thesis for these codes. In fact, they need list decoding up to *exactly the radius* to which our algorithms from Chapters 6 and 8 can decode these codes! Namely, they need the ability to decode up to (close to) the Johnson radius. In particular, the improvement in number of errors corrected that we obtained over the earlier works [Sud97a, SW99] is essential for the application in [SSW00]. [4]

The work of [SSW00] gives highly efficient traitor tracing algorithms. For example, the scheme based on Reed-Solomon codes can run in poly$(\log N)$ time where N is the number of subscribed users, and protect against collusions of size $\log^{\Omega(1)} N$. The drawback is that the scheme is guaranteed to find only one traitor in the worst-case, though it never accuses innocent subscribers.

Codes have also been used in less direct ways for traitor tracing. For example, the scheme of Boneh and Franklin [BF99] uses Reed-Solomon codes in the construction, albeit together with other components. Their tracing algorithm requires only unique decoding of Reed-Solomon codes; however, they do mention that using the list decoding algorithms will increase the size of the collusion for which the algorithm performs meaningful tracing (see [BF99] for further details). The nice feature of their scheme is that it traces *all* traitors in coalitions of up to a certain size, and once again never accuses innocent subscribers.

[4] This makes this application somewhat unique. For example, the improvement in Reed-Solomon decoding we obtained in this thesis over the earlier work of [Sud97a] is definitely important for its own sake and also because the technique generalized to soft decoding which in turn is crucial in applications to decoding concatenated codes and elsewhere. But for all the applications to complexity theory that we have outlined, the earlier result of [Sud97a] would have sufficed. This is not the case for this application to traitor tracing.

Chapter 13

Concluding Remarks

The outcome of any serious research can only be to make two questions grow where only one grew before.

Thorstein Veblen

13.1 Summary of contributions

In this thesis, we have addressed several fundamental questions concerning list decoding. We began in the first part of the thesis with the study of certain combinatorial aspects of list decoding, and established lower and upper bounds on the number of errors correctable via list decoding, as a function of the rate and minimum distance of the code. In particular, the "Johnson bounds" highlighted the radius up to which list decoding with small lists is possible for any code of certain minimum distance, thereby posing algorithmic challenges to design efficient algorithms to decode important codes up to their "list decoding potential" (i.e., their respective Johnson radii).

We then met this challenge for several important families of codes. In particular, we presented such a list decoding algorithm for Reed-Solomon codes, and also obtained a version of it that could handle soft information. We also presented a unified (soft) list decoding algorithm for a general family of codes called ideal-based codes that includes Reed-Solomon, algebraic-geometric, and Chinese Remainder codes as special cases.

Using our soft decoding algorithm for Reed-Solomon and algebraic-geometric codes at the core, we then presented algorithms to list decode several interesting families of concatenated codes to close to their "list decoding potential" (at least for low rates, and sometimes for all rates). This enabled us to construct binary codes of good rate which were efficiently list decodable from the "maximum" possible fraction of errors, i.e., a fraction $(1/2 - \varepsilon)$ of errors for a constant $\varepsilon > 0$ as small as we seek. The best construction obtained a rate of $\Omega(\varepsilon^4)$, which comes close to $\Theta(\varepsilon^2)$, the best possible rate (achieved by random codes and exponential time brute-force decoding). We also studied the analogous question of list decoding from erasures, established combinatorial bounds for it, and obtained constructions of codes with good, and sometimes almost optimal, rate along with list decoding algorithms that worked even when almost all (i.e., a fraction $(1 - \varepsilon)$) of the symbols are erased.

In the quest for either improving the rate, or achieving a similar rate with simpler constructions, we then presented several novel constructions of codes that shared the common thread of using expander graphs as a component to redistribute symbols. This yielded codes of rate $\Omega(\varepsilon^2)$ which were efficiently list decodable using small lists even when most (i.e., a fraction $(1-\varepsilon)$) of the symbols are in error. This construction was much simpler and the decoding algorithms much faster than those for AG-codes, which also enjoy a similar list decodability property. We were also to obtain the *optimal* $\Omega(\varepsilon)$ rate for such codes, but the list decoding algorithm had sub-exponential runtime and the construction was probabilistic. En route obtaining these results, we also introduced two important constructs: *pseudolinear codes* and *juxtaposed codes*, which we believe will find applications in future constructions as well.

Using the ideas from our expander-based list decodable code constructions, we were also able to obtain a construction of asymptotically good *linear time encodable and decodable* codes that can be *unique* (not list) decoded from the maximum possible fraction of errors; i.e., codes over a large alphabet that can correct a fraction $(1/2-\varepsilon)$ of errors, and binary codes that can correct a fraction $(1/4-\varepsilon)$ of errors. The large alphabet codes simultaneously achieve optimal (up to constant factors) rate, encoding time, and decoding time! The binary codes also achieve optimal encoding and decoding times, and their rate is the best known for constructive codes of relative distance $(1/2-O(\varepsilon))$ (which is an a priori requirement for being able to unique decode a fraction $(1/4-\varepsilon)$ of errors). Hence, in some sense our linear-time binary code constructions are also "optimal" in all relevant dimensions.

13.2 Directions for future work

Although we managed to answer some of the basic algorithmic questions concerning list decoding, a number of questions and directions remain open for future work. We have already described many specific open questions in the relevant chapters, but there a few which we highlight below, followed by the mention of a broad, long-term goal where much work remains to be done.

13.2.1 Some specific open questions

The following lists a couple of central combinatorial questions concerning list decoding that are still unanswered (Open Questions 4.1 and 6.1):

- Is the Johnson radius $J(\delta) = (1-\sqrt{1-2\delta})/2$ the largest (relative) radius for which a binary *linear* code of relative distance δ is guaranteed to have a polynomial number of codewords? We answered this question in the affirmative in Chapter 4 assuming the GRH (Theorem 4.6), and the result is known to hold unconditionally for general, *non-linear* codes (Proposition 4.1).

- Is $(1-\sqrt{r})$ the largest fraction of errors that can be list decoded with polynomial-sized lists for a Reed-Solomon code of rate r?

We next list some questions concerning algorithmic list decodability and explicit constructions of list decodable codes.

- Is there a near-linear time algorithm to list decode Reed-Solomon codes of rate r up to a fraction $(1-\varepsilon)(1-\sqrt{r})$ of errors (for $\varepsilon > 0$ a fixed, but arbitrarily small constant)? (Question 6.2)

- Is there a polynomial time construction of codes over a large but constant-sized alphabet which are list decodable up to a fraction $(1-\varepsilon)$ of errors in polynomial time and which have rate asymptotically better than $\Omega(\varepsilon^2)$ (here $\varepsilon > 0$ is an arbitrarily small constant) ?

- The same question as above for *binary* codes, with decoding radius and rate replaced by $(1/2 - \varepsilon)$ and $\Omega(\varepsilon^4)$ respectively. (Question 8.4)

- Is there a *constructive* family of binary codes list decodable in polynomial time from a fraction $(1 - \varepsilon)$ of erasures and which have rate close to the optimal bound of $\Omega(\varepsilon)$? (Question 10.2)

- Is there a polynomial time algorithm to list decode concatenated codes (with outer code being, say, a Reed-Solomon code) beyond the product bound for *every choice* of outer and inner distances? (Question 8.3)

13.2.2 Construction of "capacity-approaching" list decodable codes

The reader might recall the discussion in Chapter 5 (namely the one following Theorem 5.1) about the fact that list decoding permits one to approach the capacity of, say, the binary symmetric channel, even when the errors are adversarially, and not randomly, effected. Specifically, for binary linear codes, Theorem 5.5 shows that one can get within ε of the Shannon capacity $1-H(p)$ of the binary symmetric channel BSC_p with cross-over probability p, $0 < p < 1/2$, using list decoding with lists of size $1/\varepsilon$, even if the p fraction of errors are adversarially chosen by the channel.

This result is highly non-constructive, and it is not clear how to construct binary codes with rate close to the capacity $1 - H(p)$ and which are list decodable from a fraction p of errors using lists of a constant, or even polynomial, size. The requirement of an efficient list decoding algorithm for up to a fraction p of errors makes the question even harder.

Towards the latter portions of this thesis, we studied this question focusing on the high-noise regime, i.e., when the fraction p of errors equals $(1/2 - \varepsilon)$ for some small constant $\varepsilon > 0$. (We also considered the case $p = (1 - \varepsilon)$ for codes over a large alphabet, but let us now focus on the binary case.) In this case, the "capacity result" says that there exist binary linear codes of rate $\sigma\varepsilon^2$ list decodable from a fraction $(1/2 - \varepsilon)$ of errors (using lists of size $O(1/\varepsilon^2)$), for an absolute constant $\sigma > 0$. For this question, even matching the quadratic dependence on ε, let alone getting close to the exact rate $\sigma\varepsilon^2$, is a highly non-trivial task. Our best constructive results achieve a rate of $\Omega(\varepsilon^4)$, and we can improve it to $\Omega(\varepsilon^3)$ with a sub-exponential time list decoding algorithm. There is much room for improvement; however, getting anywhere close to the optimal $\Omega(\varepsilon^2)$ rate seems extremely challenging, and such a result would definitely be a major breakthrough.

All this applied only to the high-noise (and consequently, low-rate) regime. In general, an interesting question is how close to the capacity $(1 - H(p))$ one can get for codes list decodable from a fraction p of errors, for other, smaller, values of p. We are still a long way off from answering this question. A constructive family of binary codes of rate $(1 - H(p) - f(L))$ and that is efficiently list decodable from a fraction p of errors using lists of size L, for *any* function f (let alone the function $f(L) = 1/L$ as guaranteed by the existential results), would be a remarkable result. Such a result would represent a constructive version of the "capacity theorem" for list decoding. We are no where close to such a result yet.

A good goal to attack first is to construct such binary codes with a rate better than $1 - H(2p)$ for every value of p in the range $0 < p < 1/2$ (for $1/4 < p < 1/2$, we already know such constructions, so the interesting case is when $0 < p < 1/4$). The motivation for this question is the following. In order to be able to unique decode a code up to a fraction p of errors, its relative distance has to be at least $2p$. The best rate known for such codes, even non-constructively, is $1 - H(2p)$, as given by the Gilbert-Varshamov bound. Therefore, beating a rate of $(1 - H(2p))$ would imply a constructive version of list decoding that surpasses the best performance achievable using unique decoding, even when it is allowed non-constructive codes and exponential time decoding! There is some hope that a concatenated code based on outer AG-code and a carefully chosen constant-sized inner code, together with a decoding algorithm that uses the soft list decoder for AG-codes with a careful choice of weights, can achieve this goal. But, further work needs to be done to verify this intuition and formally prove such a result.

In conclusion, we want to point out that the quest for better and better codes that approach Shannon capacity in the probabilistic noise model has led to a lot of ground-breaking research in coding theory and has seen significant successes. List decoding offers the potential of achieving rates close to capacity even under adversarial noise models. As this thesis provides ample evidence that list decoding can be efficiently performed for a wide variety of codes, this now raises the hope that the analogous pursuit of constructive capacity-approaching codes for list decoding as discussed above, might, after all, be a tractable one, and one which will eventually meet with substantial successes. The end result of such a pursuit, if it is successful, will of course be dramatic, but in addition we believe that there are several novel code constructions and algorithmic techniques to be discovered en route.

Appendix A

GMD Decoding of Concatenated Codes

We present a proof of Proposition 11.11, restated below, which was used in the construction of linear-time binary codes from Chapter 11. The result in particular implies an efficient algorithm to decode concatenated codes up to the product bound provided there exists an efficient errors-and-erasures decoding algorithm for the outer code, and the decoding of the inner codes can also be performed efficiently (which is usually easy since the dimension of the inner code is typically small).

Proposition A.1 *Let C_{out} be an $(N,K)_Q$ code where $Q = q^k$ and let C_{in} be an $(n,k)_q$ code with minimum distance at least d. Let $\mathbf{C}$ be the $(Nn, Kk)_q$ code obtained by concatenating C_{out} with C_{in}. Assume that there exists an algorithm running in time T_{in} to uniquely decode C_{in} up to less than $d/2$ errors. Assume also the existence of an algorithm running in time T_{out} that uniquely decodes C_{out} from S erasures and E errors as long as $2E + S < \tilde{D}$ for some $\tilde{D} \le \text{dist}(C_{\text{out}})$. Then there exists an algorithm $\mathcal{A}$ running in $O(NT_{\text{in}} + dT_{\text{out}})$ time that uniquely decodes $\mathbf{C}$ from any pattern of less than $\frac{d\tilde{D}}{2}$ errors.*

The proof is based on the same approach as the GMD decoding algorithm due to Forney [For66b, For66a] and its use by Justesen [Jus72] to decode his explicit asymptotically good code constructions. The exact style of presentation is inspired by that of [Sud00a]. One technical aspect in the proof is that we show that GMD decoding works with as many rounds of decoding of the outer code as there are distinct weights passed by the inner stage. In particular this implies that one has to invoke the outer errors-and-erasures decoding algorithm at most $\lfloor d/2 \rfloor + 1$ times.

A.1 Proof

Let $\mathbf{r} \in [q]^{Nn}$ be a received word which is at a Hamming distance less than $d\tilde{D}/2$ from a codeword $\mathbf{z}$ of the concatenated code $\mathbf{C}$. We divide $\mathbf{r}$ and $\mathbf{z}$ into N blocks of n symbols each corresponding to the n encodings by the inner codes. Denote by r_i (resp. z_i) the i'th block of $\mathbf{r}$ (resp. $\mathbf{z}$), for $1 \le i \le N$. Let y_i be the unique codeword of C_{in} with $\Delta(y_i, r_i) < d/2$, if one exists. The inner decoder can find such an y_i if it exists in time T_{in}. If the inner decoder fails to find any codeword within distance $d/2$ of r_i, we set y_i to be an arbitrary codeword of C_{in}. For each y_i, we compute a weight $w_i = \min\{\Delta(r_i, y_i), \lfloor d/2 \rfloor\}$. The inner codewords $y_1, y_2, \ldots, y_N$ together with the weights $w_1, \ldots, w_N$ can all be found in NT_{in} time.

Assume without loss of generality that $w_1 \le w_2 \le \cdots \le w_N$. Let s be the number of distinct weights among $w_1, w_2, \ldots, w_N$. By the definition of the weights, we clearly have $s \le \lfloor \frac{d}{2} \rfloor + 1$. Let S_j be the block of (contiguous) indices with the same value of w_i for $1 \le j \le s$, and denote by $\tilde{w}_j$ this common weight. Let $n_j = |S_j|$.

The decoding of $\mathbf{r}$ is now finished as follows. For each p, $1 \le p \le s$, we run the assumed errors-and-erasures decoding algorithm for C_{out} on the received word $\langle y_1, y_2, \ldots, y_N \rangle$, by declaring the y_i's in the last p blocks S_j, $s - p + 1 \le j \le s$, as erasures. If any of these decodings finds a message x such that $\Delta(\mathbf{r}, \mathbf{C}(x)) < d\tilde{D}/2$, we output the codeword $\mathbf{C}(x)$ and terminate the algorithm, otherwise we report that there exists no codeword of $\mathbf{C}$ at a Hamming distance less than $d\tilde{D}/2$ from $\mathbf{r}$.

Since the algorithm runs the outer decoding algorithm s times, the total time of the decoding algorithm is $O(NT_{\text{in}} + dT_{\text{out}})$, as claimed. We next proceed to prove the correctness of the algorithm. That is, if there exists $\mathbf{z} \in \mathbf{C}$ with $\Delta(\mathbf{r}, \mathbf{z}) < d\tilde{D}/2$, then the above algorithm will find and output $\mathbf{z}$.

Let $\ell_i = \Delta(r_i, z_i)$ — then by our definition of w_i, we have $\ell_i \ge w_i$. Also, if $y_i \ne z_i$ (i.e., the inner decoder makes a mistake in position i), then clearly $\ell_i \ge d - w_i$ (by triangle inequality). So if we denote by a_i the indicator variable for $y_i \ne z_i$, we have $\ell_i \ge a_i(d - w_i)$. Together with $\ell_i \ge w_i$, this gives

$$\ell_i \ge (1 - a_i)w_i + a_i(d - w_i) = w_i + a_i(d - 2w_i) \ . \tag{A.1}$$

We would like to prove that if the decoding failed to find the codeword $\mathbf{z}$, then we must have $\Delta(\mathbf{r}, \mathbf{z}) \ge d\tilde{D}/2$ errors. In our notation this means we want to prove

$$\sum_{i=1}^{N} \ell_i \ge \frac{\tilde{D}d}{2} \ . \tag{A.2}$$

Define the quantities $A_j = \sum_{i \in S_j} a_i$ and $L_j = \sum_{i \in S_j} \ell_i$. We have by (A.1), for $1 \le j \le s$,

$$L_j \ge n_j \tilde{w}_j + A_j(d - 2\tilde{w}_j) \ . \tag{A.3}$$

Rewriting (A.2), recall that our goal is to prove that

$$\frac{1}{d} \sum_{j=1}^{s} L_j \ge \frac{\tilde{D}}{2} \ . \tag{A.4}$$

Define $x_j = (1 - 2\tilde{w}_j/d)$. Clearly $1 \ge x_1 > x_2 > \cdots > x_s \ge 0$. We have from (A.3) that

$$\frac{L_j}{d} \ge n_j \frac{(1 - x_j)}{2} + A_j x_j \ . \tag{A.5}$$

Define $\Delta_j = \frac{n_j}{2} - A_j$. Using (A.5) above and the fact that $\sum_{j=1}^{s} n_j = N$, we get that in order to prove (A.4), it suffices to prove that

$$\sum_{j=1}^{s} \Delta_j x_j \le \frac{N - \tilde{D}}{2} \ . \tag{A.6}$$

Now if each of the s errors-and-erasures outer decodings fail to find the codeword $\mathbf{z}$, then in each run the $E + S/2 < \tilde{D}/2$ condition must fail. In such a case we must have, for each p, $1 \le p \le s$,

$$\sum_{j=1}^{p} A_j + \frac{1}{2} \cdot \sum_{j=p+1}^{s} n_j \ge \frac{\tilde{D}}{2} \ , \tag{A.7}$$

which is the same as

$$\sum_{j=1}^{p} \Delta_j \leq \frac{N - \tilde{D}}{2} \ . \tag{A.8}$$

Define $x_{s+1} = 0$. Multiplying the p'th equation above with the non-negative quantity $(x_p - x_{p+1})$ for $1 \leq p \leq s$, and adding up the resulting inequalities, we get

$$\sum_{j=1}^{s} \Delta_j x_j \leq \frac{N - \tilde{D}}{2} \cdot x_1 \leq \frac{N - \tilde{D}}{2} \ , \tag{A.9}$$

which is exactly Equation (A.6) that we had to prove. □

Bibliography

[AVZ00] Erik Agrell, Alexander Vardy, and Kenneth Zeger. Upper bounds for constant-weight codes. *IEEE Transactions on Information Theory*, 46:2373–2395, 2000.

[Ahl73] Rudolf Ahlswede. Channel capacities for list codes. *Journal of Applied Probability*, 10:824–836, 1973.

[Alo99] Noga Alon, October 1999. Personal Communication.

[ABN$^+$92] Noga Alon, Jehoshua Bruck, Joseph Naor, Moni Naor, and Ronny Roth. Construction of asymptotically good low-rate error-correcting codes through pseudo-random graphs. *IEEE Transactions on Information Theory*, 38:509–516, 1992.

[AGHP92] Noga Alon, Oded Goldreich, Johan Håstad, and Réne Peralta. Simple constructions of almost k-wise independent random variables. *Random Structures and Algorithms*, 3:289–304, 1992.

[AGKS01] Noga Alon, Venkatesan Guruswami, Tali Kaufman, and Madhu Sudan. Guessing secrets efficiently via list decoding. *Manuscript*, July 2001.

[AS92] Noga Alon and Joel Spencer. *The Probabilistic Method.* John Wiley and Sons, Inc., 1992.

[ALRS99] Sigal Ar, Richard Lipton, Ronitt Rubinfeld, and Madhu Sudan. Reconstructing algebraic functions from mixed data. *SIAM Journal on Computing*, 28(2):488–511, 1999.

[AS97] Sanjeev Arora and Madhu Sudan. Improved low-degree testing and its applications. *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 485–495, 1997.

[Art65] Emil Artin. *Collected Papers.* ed. S. Lang and J. T. Tate, Springer-Verlag, 1965. pp. viii-ix.

[Art91] Michael Artin. *Algebra.* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1991.

[ABL00] Alexei Ashikhmin, Alexander Barg, and Simon Litsyn. A new upper bound on codes decodable into size-2 lists. In Ingo Althofer *et al.*, editor, *Numbers, Information and Complexity*, pages 239–244. Boston: Kluwer Publishers, 2000.

[AB83] C. Asmuth and J. Bloom. A modular approach to key safeguarding. *IEEE Transactions on Information Theory*, 29:208–210, March 1983.

[AP00] Daniel Augot and Lancelot Pecquet. A Hensel lifting to replace factorization in list decoding of algebraic-geometric and Reed-Solomon codes. *IEEE Transactions on Information Theory*, 46:2605–2613, November 2000.

[BFNW93] László Babai, Lance Fortnow, Noam Nisan, and Avi Wigderson. BPP has subexponential time simulations unless EXPTIME has publishable proofs. *Computational Complexity*, 3(4):307–318, 1993.

[BZ01] Alexander Barg and Gillés Zémor. Error exponents of expander codes. *IEEE Transactions on Information Theory*, to appear, 2001.

[Bei88] Richard Beigel. NP-hard sets are p-superterse unless R = NP. *Technical Report TR 4, Department of Computer Science, Johns Hopkins University*, 1988.

[BGS98] Mihir Bellare, Oded Goldreich, and Madhu Sudan. Free bits, PCP's and non-approximability — towards tight results. *SIAM Journal on Computing*, 27(3):804–915, 1998.

[Ber68] Elwyn Berlekamp. *Algebraic Coding Theory.* McGraw Hill, New York, 1968.

[Ber70] Elwyn Berlekamp. Factoring polynomials over large finite fields. *Mathematics of Computations*, 24:713–735, 1970.

[Ber96] Elwyn Berlekamp. Bounded distance +1 soft-decision Reed-Solomon decoding. *IEEE Transactions on Information Theory*, 42(3):704–720, 1996.

[Bla83] Richard E. Blahut. *Theory and Practice of Error Control Codes.* Addison-Wesley, Reading, Massachusetts, 1983.

[Bli86] Volodia M. Blinovsky. Bounds for codes in the case of list decoding of finite volume. *Problems of Information Transmission*, 22(1):7–19, 1986.

[Bli97] Volodia M. Blinovsky. *Asymptotic Combinatorial Coding Theory.* Kluwer Academic Publishers, Boston, 1997.

[Bli00] Volodia M. Blinovsky. Lower bound for the linear multiple packing of the binary hamming space. *Journal of Combinatorial Theory, Series A*, 92(1):95–101, 2000.

[Bol86] Bela Bollobás. *Combinatorics.* Cambridge University Press, Cambridge, U.K, 1986.

[Bon00] Dan Boneh. Finding smooth integers in short intervals using CRT decoding. *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing*, pages 265–272, 2000.

[BF99] Dan Boneh and Matthew Franklin. An efficient public-key traitor tracing scheme. In *Proc. Advances in Cryptography – Crypto '99*, pages 338–353. Lecture Notes in Computer Science 1666, Springer-Verlag, 1999.

[BR60] R. C. Bose and D. K. Ray-Chaudhuri. On a class of error correcting binary group codes. *Information and Control*, 3:68–79, 1960.

[Bro98] Andries E. Brouwer. *Bounds on the size of linear codes.* Chapter 4 in *Handbook of Coding Theory*, V. S. Pless and W. C. Huffman (eds), Elsevier, pp. 295-461, 1998.

[Bus52] K. A. Bush. Orthogonal arrays of index unity. *Ann. Math. Stat.*, 23:426–434, 1952.

[CPS99] Jin-Yi Cai, A. Pavan, and D. Sivakumar. On the hardness of the permanent. *Proceedings of the 16th International Symposium on Theoretical Aspects of Computer Science*, March 1999.

[CFN94] Benny Chor, Amos Fiat, and Moni Naor. Tracing traitors. In *Proc. Advances in Cryptography – Crypto '94*, pages 257–270. Lecture Notes in Computer Science 839, Springer-Verlag, 1994.

[CGL01] Fan Chung, Ron Graham, and Tom Leighton. Guessing secrets. *The Electronic Journal of Combinatorics*, 8(1):R13, 2001.

[CES01] Gérard D. Cohen, Sylvia B. Encheva, and Hans G. Schaathun. On separating codes. *Technical report, Ecole Nationale Supérieure des Télécommunications, Paris*, 2001.

[Coh93] Henri Cohen. *A Course in Computational Algebraic Number Theory.* Graduate Texts in Mathematics **138**, Springer Verlag, Berlin, 1993.

[Den83] D. E. R. Denning. *Cryptography and Data Security.* Reading, MA: Addison-Wesley, 1983.

[DV83] V. G. Drinfeld and Serge G. Vlăduţ. Number of points of an algebraic curve. *Func. Anal.*, 17:53–54, 1983.

[Dum89] Ilya I. Dumer. Two algorithms for the decoding of linear codes. *Problems of Information Transmission*, 25(1):24–32, 1989.

[Dum98] Ilya I. Dumer. Concatenated codes and their multilevel generalizations. In V. S. Pless and W. C. Huffman, editors, *Handbook of Coding Theory*, volume 2, pages 1911–1988. North Holland, 1998.

[Eli55] Peter Elias. Coding for two noisy channels. *Information Theory, Third London Symposium*, pages 61–76, September 1955.

[Eli57] Peter Elias. List decoding for noisy channels. *Technical Report 335, Research Laboratory of Electronics, MIT*, 1957.

[Eli88] Peter Elias. Zero error capacity under list decoding. *IEEE Transactions on Information Theory*, 34(5):1070–1074, September 1988. Originally appeared as *Quarterly Progress Report*, vol. 48, pp. 88-90, Research Laboratory of Electronics, MIT, January 1958.

[Eli91] Peter Elias. Error-correcting codes for list decoding. *IEEE Transactions on Information Theory*, 37:5–12, 1991.

[Elk97] Noam D. Elkies. Explicit modular towers. *Proceedings of the 35th Annual Allerton Conference on Communication, Control and Computing*, pages 23–32, 1997.

[Elk01] Noam D. Elkies. Excellent non-linear codes from Modular curves. *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, pages 200–208, July 2001.

[EZ95] Thomas Ericson and Victor Zinoviev. Spherical codes generated by binary partitions of symmetric pointsets. *IEEE Transactions on Information Theory*, 41:107–129, 1995.

[FLN00] Uriel Feige, Michael Langberg, and Kobbi Nissim. On the hardness of approximating NP witnesses. *Proceedings of the 3rd International Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX)*, pages 120–131, September 2000.

[FL92] Uriel Feige and Carsten Lund. On the hardness of computing the permanent of random matrices. *Proceedings of the 24th ACM Symposium on Theory of Computing*, pages 643–654, 1992.

[Fei95] Joan Feigenbaum. The use of coding theory in computational complexity. In R. Calderbank, editor, *Proceedings of Symposia in Applied Mathematics*, pages 203–229. American Mathematics Society, Providence, 1995.

[FR93] Gui-Liang Feng and Thammavarapu R. N. Rao. Decoding algebraic geometric codes up to the designed minimum distance. *IEEE Transactions on Information Theory*, 39(1):37–45, 1993.

[For66a] G. David Forney. *Concatenated Codes*. MIT Press, Cambridge, MA, 1966.

[For66b] G. David Forney. Generalized Minimum Distance decoding. *IEEE Transactions on Information Theory*, 12:125–131, 1966.

[For68] G. David Forney. Exponential error bounds for erasure, list, and decision feedback schemes. *IEEE Transactions on Information Theory*, 14(2):206–220, March 1968.

[GHLP99] Anna Gál, Shai Halevi, Richard J. Lipton, and Erez Petrank. Computing from partial solutions. *Proceedings of the 14th Annual IEEE Conference on Computation Complexity*, pages 34–45, 1999.

[GS00] Shuhong Gao and M. Amin Shokrollahi. Computing roots of polynomials over function fields of curves. *Coding Theory and Cryptography: From Enigma and Geheimschreiber to Quantum Theory (D. Joyner, Ed.), Springer*, pages 214–228, 2000.

[GS95a] Arnaldo Garcia and Henning Stichtenoth. Algebraic function fields over finite fields with many rational places. *IEEE Transactions on Information Theory*, 41:1548–1563, 1995.

[GS95b] Arnaldo Garcia and Henning Stichtenoth. A tower of Artin-Schreier extensions of function fields attaining the Drinfeld-Vlădut bound. *Inventiones Mathematicae*, 121:211–222, 1995.

[GS96] Arnaldo Garcia and Henning Stichtenoth. On the asymptotic behavior of some towers of function fields over finite fields. *Journal of Number Theory*, 61(2):248–273, December 1996.

[GS92] Peter Gemmell and Madhu Sudan. Highly resilient correctors for multivariate polynomials. *Information Processing Letters*, 43(4):169–174, 1992.

[Gol99] Oded Goldreich. *Modern Cryptography, Probabilistic Proofs, and Pseudorandomness.* Number 17 in Algorithms and Combinatorics. Springer-Verlag, 1999.

[GL89] Oded Goldreich and Leonid Levin. A hard-core predicate for all one-way functions. *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*, pages 25–32, May 1989.

[GNW95] Oded Goldreich, Noam Nisan, and Avi Wigderson. On Yao's XOR Lemma. *Technical Report TR95-050, Electronic Colloquium on Computational Complexity*, March 1995. `http://www.eccc.uni-trier.de/eccc.`

[GRS99] Oded Goldreich, Dana Ron, and Madhu Sudan. Chinese remaindering with errors. *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, pages 225–234, 1999.

[GRS00] Oded Goldreich, Dana Ron, and Madhu Sudan. Chinese remaindering with errors. *IEEE Transactions on Information Theory*, 46(5):1330–1338, July 2000.

[GRS95] Oded Goldreich, Ronitt Rubinfeld, and Madhu Sudan. Learning polynomials with queries: The highly noisy case. *Proceedings of the 36th Annual IEEE Symposium on Foundations of Computer Science*, pages 294–303, 1995.

[Gop81] V. D. Goppa. Codes on algebraic curves. *Soviet Math. Doklady*, 24:170–172, 1981.

[Gri84] Dima Grigoriev. Factorization of polynomials over a finite field and the solution of systems of algebraic equations. *Translated from Zapiski Nauchnykh Seminarov Lenningradskogo Otdeleniya Matematicheskogo Instituta im. V. A. Steklova AN SSSR*, 137:20–79, 1984.

[Gur01a] Venkatesan Guruswami. Limits to list decodability of linear codes, May 2001. Unpublished manuscript.

[Gur01b] Venkatesan Guruswami. List decoding from erasures: Bounds and code constructions. *Proceedings of the 21st Foundations of Software Technology and Theoretical Computer Science*, to appear, December 2001.

[Gur01c] Venkatesan Guruswami. Constructions of codes from number fields. *Proceedings of the 14th Symposium on Applied Algebra, Algebraic Algorithms and Error-Correcting Codes (AAECC)*, to appear, November 2001.

[GHSZ00] Venkatesan Guruswami, Johan Håstad, Madhu Sudan, and David Zuckerman. Combinatorial bounds for list decoding. *Proceedings of the 38th Annual Allerton Conference on Communication, Control and Computing*, pages 603–612, October 2000.

[GI01a] Venkatesan Guruswami and Piotr Indyk, August 2001. Untitled manuscript.

[GI01b] Venkatesan Guruswami and Piotr Indyk. Expander-based constructions of efficiently decodable codes. *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science*, to appear, October 2001.

[GSS00] Venkatesan Guruswami, Amit Sahai, and Madhu Sudan. Soft-decision decoding of Chinese Remainder codes. *Proceedings of the 41st IEEE Symposium on Foundations of Computer Science*, pages 159–168, 2000.

[GS99] Venkatesan Guruswami and Madhu Sudan. Improved decoding of Reed-Solomon and algebraic-geometric codes. *IEEE Transactions on Information Theory*, 45:1757–1767, 1999.

[GS00] Venkatesan Guruswami and Madhu Sudan. List decoding algorithms for certain concatenated codes. *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing*, pages 181–190, 2000.

[GS01a] Venkatesan Guruswami and Madhu Sudan. Extensions to the Johnson bound. *Manuscript*, February 2001.

[GS01b] Venkatesan Guruswami and Madhu Sudan. On representations of algebraic-geometric codes. *IEEE Transactions on Information Theory*, 47(4):1610–1613, May 2001.

[Ham50] Richard W. Hamming. Error Detecting and Error Correcting Codes. *Bell System Technical Journal*, 29:147–160, April 1950.

[Har69] Frank Harary. *Graph Theory*. Addison-Wesley, Reading, MA, 1969.

[HLP52] G. H. Hardy, J. E. Littlewood, and G. Pólya. *Inequalities*. Cambridge University Press, 2nd edition, 1952.

[Hel74] Hermann J. Helgert. Alternant codes. *Information and Control*, 26:369–380, 1974.

[Hoc59] A. Hocquenghem. Codes correcteurs d'erreurs. *Chiffres (Paris)*, 2:147–156, 1959.

[Hoo67] Christopher Hooley. On Artin's conjecture. *J. Reine Angew. Math.*, 225:209–220, 1967.

[HN98] Johan Håstad and Mats Näslund. The security of all RSA and Discrete Log bits. *Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science*, pages 510–519, 1998.

[I52] *I've Got a Secret*. A classic '50's and '60's television gameshow. See `http://www.timvp.com/ivegotse.html`.

[Iha81] Y. Ihara. Some remarks in the number of rational points of algebraic curves over finite fields. *J. Fac. Sci. Tokyo*, 28:721–724, 1981.

[Imp95] Russell Impagliazzo. Hard-core distributions from somewhat hard problems. *Proceedings of the 36th Annual IEEE Symposium on Foundations of Computer Science*, pages 538–545, October 1995.

[IW97] Russell Impagliazzo and Avi Wigderson. P = BPP if E requires exponential circuits: Derandomizing the XOR Lemma. *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, pages 220–229, May 1997.

[IR90] Kenneth Ireland and Michael Rosen. *A Classical Introduction to Modern Number Theory.* Springer-Verlag, 2 edition, 1990.

[Jak98] Thomas Jakobsen. Cryptanalysis of block ciphers with probabilistic non-linear relations of low degree. In Hugo Krawczyk, editor, *Proc. Advances in Cryptography – Crypto '98.* Lecture Notes in Computer Science 1462, Springer-Verlag, 1998.

[Jof74] A. Joffe. On a set of almost deterministic k-independent random variables. *Annals of Probability*, 2(1):161–162, 1974.

[Joh62] Selmer M. Johnson. A new upper bound for error-correcting codes. *IEEE Transactions on Information Theory*, 8:203–207, 1962.

[Joh63] Selmer M. Johnson. Improved asymptotic bounds for error-correcting codes. *IEEE Transactions on Information Theory*, 9:198–205, 1963.

[Jus72] Jϕrn Justesen. A class of constructive asymptotically good algebraic codes. *IEEE Transactions on Information Theory*, 18:652–656, 1972.

[Jus76] Jϕrn Justesen. On the complexity of decoding Reed-Solomon codes (corresp.). *IEEE Transactions on Information Theory*, 22(2):237–238, March 1976.

[Jus01] Jϕrn Justesen. On bounds for list decoding. *Manuscript*, March 2001.

[JH01] Jϕrn Justesen and Tom Hϕholdt. Bounds on list decoding of MDS codes. *IEEE Transactions on Information Theory*, 47(4):1604–1609, May 2001.

[JLJ$^+$89] Jϕrn Justesen, Knud J. Larsen, Helge E. Jensen, Allan Havemose, and Tom Hϕholdt. Construction and decoding of a class of algebraic geometry codes. *IEEE Transactions on Information Theory*, 35:811–821, July 1989.

[JLJH92] Jϕrn Justesen, Knud J. Larsen, Helge E. Jensen, and Tom Hϕholdt. Fast decoding of codes from algebraic plane curves. *IEEE Transactions on Information Theory*, 38:111–119, 1992.

[Kal85] Erich Kaltofen. Polynomial-time reductions from multivariate to bi- and univariate integral polynomial factorization. *SIAM Journal on Computing*, 14(2):469–489, 1985.

[Kal92] Erich Kaltofen. Polynomial factorization 1987–1991. *Proceedings of LATIN '92, I. Simon (Ed.), Springer LNCS*, 583:294–313, 1992.

[KR81] Richard Karp and Michael Rabin. Efficient randomized pattern-matching algorithms. *Technical report TR-31-81, Aiken Computation Laboratory, Harvard University*, 1981.

[KTV84] G. L. Katsman, Michael A. Tsfasman, and Serge G. Vlădut. Modular curves and codes with a polynomial construction. *IEEE Transactions on Information Theory*, 30:353–355, 1984.

[Kiw97] Marcos Kiwi. Testing and weight distributions of dual codes. *ECCC Technical Report TR-97-010*, 1997.

[KV00] Ralf Koetter and Alexander Vardy. Algebraic soft-decision decoding of Reed-Solomon codes. *Proceedings of the 38th Annual Allerton Conference on Communication, Control and Computing*, pages 625–635, October 2000.

[KKLS94] H. Krishna, B. Krishna, K. Y. Lin, and J. D. Sun. *Computational Number Theory and Digital Signal Processing: Fast algorithms and error control techniques.* Boca Raton, FL: CRC, 1994.

[KS99] S. Ravi Kumar and D. Sivakumar. Proofs, codes, and polynomial-time reducibilities. *Proceedings of the 14th Annual IEEE Conference on Computation Complexity*, 1999.

[Len85] Arjen K. Lenstra. Factoring multivariate polynomials over finite fields. *Journal of Computer and System Sciences*, 30(2):235–248, April 1985.

[LL90] Arjen K. Lenstra and Hendrik W. Lenstra. Algorithms in number theory. *Handbook of Theoretical Computer Science (Volume A: Algorithms and Complexity), Chap. 12*, pages 673–715, 1990.

[LLL82] Arjen K. Lenstra, Hendrik W. Lenstra, and Lászlo Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261:515–534, 1982.

[Len86] Hendrik W. Lenstra. Codes from algebraic number fields. In L.G.L.T. Meertens M. Hazewinkel, J.K. Lenstra, editor, *Mathematics and computer science II, Fundamental contributions in the Netherlands since 1945*, pages 95–104. North-Holland, Amsterdam, 1986.

[Lev98] V. I. Levenshtein. Universal bounds for codes and designs. *Chapter 6 in Handbook of Coding Theory, V. S. Pless and W. C. Huffman (Eds.)*, pages 499–648, 1998.

[Lip91] Richard Lipton. New directions in testing. In *Distributed Computing and Cryptography*, volume 2 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 191–202. AMS, 1991.

[L9́8] B. López. Codes on Drinfeld modular curves. In J. Buchmann *et al*, editor, *Coding Theory, Cryptography and Related Areas*, pages 175–183. Springer, Heidelberg, 1998.

[LPS88] Alex Lubotzky, R. Phillips, and Peter Sarnak. Ramanujan graphs. *Combinatorica*, 8(3):261–277, 1988.

[MS81] F. J. MacWilliams and Neil J. A. Sloane. *The Theory of Error-Correcting Codes*. Elsevier/North-Holland, Amsterdam, 1981.

[Man76] David M. Mandelbaum. On a class of arithmetic codes and a decoding algorithm. *IEEE Transactions on Information Theory*, 21:85–88, 1976.

[Man78] David M. Mandelbaum. Further results on decoding arithmetic residue codes. *IEEE Transactions on Information Theory*, 24:643–644, 1978.

[MV85] Y. I. Manin and Serge G. Vlădut. Linear codes and modular curves. *J. Soviet. Math.*, 30:2611–2643, 1985.

[Mar88] G. A. Margulis. Explicit group-theoretical constructions of combinatorial schemes and their applications to the design of expanders and superconcentrators. *Problems of Information Transmission*, 24:39–46, 1988.

[Mas69] James L. Massey. Shift-register synthesis and BCH decoding. *IEEE Transactions on Information Theory*, 15:122–127, January 1969.

[Mat99] R. Matsumoto. On the second step in the Guruswami-Sudan list decoding algorithm for AG-codes. *Technical Report of the Institute of Electronics, Information and Communication Engineers (IEICE)*, pages 65–70, 1999.

[MRRW77] Robert J. McEliece, Eugene R. Rodemich, Howard Rumsey Jr., and Lloyd R. Welch. New upper bounds on the rate of a code via the Delsarte-Macwilliams inequalities. *IEEE Transactions on Information Theory*, 23:157–166, 1977.

[Mic99] Daniele Micciancio. Lecture notes on Lattices in Cryptography and Cryptanalysis, University of California at San Diego. Available at `http://www-cse.ucsd.edu/~daniele/cse291fa99.html`, Fall 1999.

[MS01] Daniele Micciancio and Nathan Segerlind. Using prefixes to efficiently guess two secrets. *Manuscript*, July 2001.

[MU01] Elchanan Mossel and Christopher Umans. On the complexity of approximating the VC dimension. *Proceedings of the 16th Annual Conference on Computational Complexity*, June 2001.

[NN93] Joseph Naor and Moni Naor. Small-bias probability spaces: Efficient constructions and applications. *SIAM Journal on Computing*, 22(4):838–856, 1993.

[Nie00] Rasmus R. Nielsen. Decoding concatenated codes using Sudan's algorithm. *Manuscript submitted for publication*, May 2000.

[NH99a] Rasmus R. Nielsen and Tom Hϕholdt. Decoding Hermitian codes with Sudan's algorithm. *Proceedings of AAECC-13, LNCS 1719*, pages 260–270, 1999.

[NH99b] Rasmus R. Nielsen and Tom Hϕholdt. Decoding Reed-Solomon codes beyond half the minimum distance. *Coding Theory, Cryptography and Related areas, (eds. Buchmann, Hoeholdt, Stichtenoth and H. tapia-Recillas)*, pages 221–236, 1999.

[Nis96] Noam Nisan. Extracting Randomness: How and Why – A survey. *Proceedings of the 11th Annual IEEE Symposium on Computational Complexity*, pages 44–58, 1996.

[NW94] Noam Nisan and Avi Wigderson. Hardness vs randomness. *Journal of Computer and System Sciences*, 49(2):149–167, October 1994.

[OS99] Vadim Olshevsky and M. Amin Shokrollahi. A displacement structure approach to efficient list decoding of algebraic geometric codes. *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, pages 235–244, 1999.

[Pec01] Lancelot Pecquet. List decoding of algebraic-geometric codes. *Manuscript*, May 2001.

[Pel89] Ruud Pellikaan. On a decoding algorithm for codes on maximal curves. *IEEE Transactions on Information Theory*, 35:1228–1232, 1989.

[Pet60] W. Wesley Peterson. Encoding and error-correction procedures for Bose-Chaudhuri codes. *IEEE Transactions on Information Theory*, 6:459–470, 1960.

[RS60] Irving S. Reed and Gustav Solomon. Polynomial codes over certain finite fields. *J. SIAM*, 8:300–304, 1960.

[RR00] Ronny Roth and Gitit Ruckenstein. Efficient decoding of Reed-Solomon codes beyond half the minimum distance. *IEEE Transactions on Information Theory*, 46(1):246–257, January 2000.

[RR01] Gitit Ruckenstein and Ronny Roth. Upper bounds on the list-decoding radius of Reed-Solomon codes. *Proceedings of the International Symposium on Information Theory (ISIT) 2001*, June 2001.

[Sch99] Uwe Schoning. A probabilistic algorithm for k-SAT and Constraint Satisfaction Problems. *Proceedings of the 40th IEEE Symposium on Foundations of Computer Science*, pages 410–414, 1999.

[Seg94] Yu L. Segalovich. Separating systems. *Problems of Information Transmission*, 30(2):105–123, 1994.

[SU01] Ronen Shaltiel and Christopher Umans. Simple extractors for all min-entropies and a new pseudo-random generator. *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science*, to appear, October 2001.

[Sha48] Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 623–656, 1948.

[Sha56] Claude E. Shannon. The zero error capacity of a noisy channel. *IEEE Transactions on Information Theory*, 2(3):8–19, 1956.

[SGB67] Claude E. Shannon, Robert G. Gallager, and Elwyn R. Berlekamp. Lower bounds to error probability for coding on discrete memoryless channels. *Information and Control*, 10:65–103 (Part I), 522–552 (Part II), 1967.

[She93] Ba-Zhong Shen. A Justesen construction of binary concatenated codes that asymptotically meet the Zyablov bound for low rate. *IEEE Transactions on Information Theory*, 39:239–242, 1993.

[SW99] M. Amin Shokrollahi and Hal Wasserman. List decoding of algebraic-geometric codes. *IEEE Transactions on Information Theory*, 45(2):432–437, 1999.

[Shu00] Kenneth Shum. *A Low-Complexity Algorithm for Constructing Algebraic-Geometric Codes Better than the Gilbert-Varshamov Bound.* PhD thesis, University of Southern California, Los Angeles, 2000.

[SAK$^+$01] Kenneth Shum, Ilia Aleshnikov, P. Vijay Kumar, Henning Stichtenoth, and Vinay Deolalikar. A low-complexity algorithm for the construction of algebraic-geometric codes better than the Gilbert-Varshamov bound. *IEEE Transactions on Information Theory*, 2001.

[Sid94] V. M. Sidelnikov. Decoding Reed-Solomon codes beyond $(d-1)/2$ errors and zeros of multivariate polynomials. *Problems of Information Transmission*, 30(1):44–59, 1994.

[SSW00] Alice Silverberg, Jessica Standon, and Judy Walker. Efficient traitor tracing algorithms using list decoding. *Manuscript*, 2000.

[SS96] Michael Sipser and Daniel Spielman. Expander codes. *IEEE Transactions on Information Theory*, 42(6):1710–1722, 1996.

[Siv99] D. Sivakumar. On membership comparable sets. *Journal of Computer and System Sciences*, 59(2):270–280, October 1999.

[SV90] Alexei N. Skorobogatov and Serge G. Vlădut. On decoding of algebraic geometric codes. *IEEE Transactions on Information Theory*, 36:1051–1060, 1990.

[SJJT86] M. A. Soderstrand, W. K. Jenkins, G. A. Jullien, and F. J. Taylor. *Residue Number System Arithmetic: Modern Applications in Digital Signal Processing.* New York: IEEE Press, 1986.

[Spi95] Daniel Spielman. *Computationally Efficient Error-Correcting Codes and Holographic Proofs.* PhD thesis, Massachusetts Institute of Technology, June 1995.

[Spi96] Daniel Spielman. Linear-time encodable and decodable error-correcting codes. *IEEE Transactions on Information Theory*, 42(6):1723–1732, 1996.

[Sti93] Henning Stichtenoth. *Algebraic Function Fields and Codes.* Universitext, Springer-Verlag, Berlin, 1993.

[Sud97a] Madhu Sudan. Decoding of Reed-Solomon codes beyond the error-correction bound. *Journal of Complexity*, 13(1):180–193, 1997.

[Sud97b] Madhu Sudan. Decoding of Reed-Solomon codes beyond the error-correction diameter. *Proceedings of the 35th Annual Allerton Conference on Communication, Control and Computing*, 1997.

[Sud00a] Madhu Sudan. *A Crash Course in Coding Theory*, Lecture no. 5. Slides available from `http://theory.lcs.mit.edu/~madhu`, November 2000.

[Sud00b] Madhu Sudan. List decoding: Algorithms and applications. *SIGACT News*, 31:16–27, 2000.

[STV99] Madhu Sudan, Luca Trevisan, and Salil Vadhan. Pseudorandom generators without the XOR lemma. *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, pages 537–546, 1999.

[SKHN75] Yasuo Sugiyama, Masao Kasahara, Shigeichi Hirasawa, and Toshihiko Namekawa. A method for solving key equation for decoding Goppa codes. *Information and Control*, 27:87–99, 1975.

[TZ01] Amnon Ta-Shma and David Zuckerman. Extractor Codes. *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, pages 193–199, July 2001.

[TZS01] Amnon Ta-Shma, David Zuckerman, and Shmuel Safra. Extractors from Reed-Muller codes. *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science*, to appear, October 2001.

[Tod91] S. Toda. On polynomial-time truth-table reducibility of intractable sets to p-selective sets. *Math. Systems Theory*, 24:69–82, 1991.

[Tre99] Luca Trevisan. Constructions of near-optimal extractors using pseudorandom generators. *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, pages 141–148, May 1999.

[TVZ82] Michael A. Tsfasman, Serge G. Vlădut, and Thomas Zink. Modular curves, Shimura curves, and codes better than the Varshamov-Gilbert bound. *Math. Nachrichten*, 109:21–28, 1982.

[Val79] Leslie Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201, April 1979.

[VV86] Leslie Valiant and Vijay Vazirani. NP is as easy as detecting unique solutions. *Theoretical Computer Science*, 47:85–93, 1986.

[vL99] J. H. van Lint. *Introduction to Coding Theory.* Graduate Texts in Mathematics **86**, (Third Edition) Springer-Verlag, Berlin, 1999.

[WF94] Victor K. Wei and Gui-Liang Feng. Improved lower bounds on the sizes of error-correcting codes for list decoding. *IEEE Transactions on Information Theory*, 40(2):559–563, 1994.

[WB86] Lloyd R. Welch and Elwyn R. Berlekamp. Error correction of algebraic block codes. *US Patent Number 4,633,470*, December 1986.

[Wel73] Edward J. Weldon, Jr. Justesen's construction — the low-rate case. *IEEE Transactions on Information Theory*, 19:711–713, 1973.

[WB99] Stephen B. Wicker and Vijay K. Bhargava, editors. *Reed-Solomon Codes and Their Applications.* John Wiley and Sons, Inc., September 1999.

[Woz58] John M. Wozencraft. List Decoding. *Quarterly Progress Report, Research Laboratory of Electronics, MIT*, 48:90–95, 1958.

[WS00] Xin-Wen Wu and Paul H. Siegel. Efficient list decoding of algebraic geometric codes beyond the error correction bound. *Proceedings of the International Symposium on Information Theory*, June 2000.

[Z01́] Gillés Zémor. On expander codes. *IEEE Transactions on Information Theory*, 47(2):835–837, 2001.

[ZP82] Victor V. Zyablov and Mark S. Pinsker. List cascade decoding. *Problems of Information Transmission*, 17(4):29–34, 1981 (in Russian); pp. 236-240 (in English), 1982.