2 Preliminaries and Monograph 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 book. Finally, we discuss the structure of this work 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. Note that for general, non-linear codes, the dimension is simply the logarithm to the base q of the number of codewords, and therefore need not be an integer. 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]

[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. Also, in some texts, non-linear codes with M codewords are referred to as $(n, M, d)_q$-codes.

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

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

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

[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 $\mathbb{F}_q^n$.

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.

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 book 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 book 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 $0 \le x \le 1$, and an integer $q \ge 2$, 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 book. Several of these will also be used as building blocks for the new code constructions that we present.

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 $q \ge n$, is defined as follows. Let $\alpha_1, \alpha_2, \ldots, \alpha_n$ be n *distinct* field elements in $\mathbb{F}_q$ (since $q \ge n$, 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 2.5. *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. [193, 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 [198] 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 [132, 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

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

ℓ symbols over $\mathbb{F}_q$ into q^ℓ symbols. But it has very good distance properties — its relative distance equals $(1-1/q)$, 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

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 [59] 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

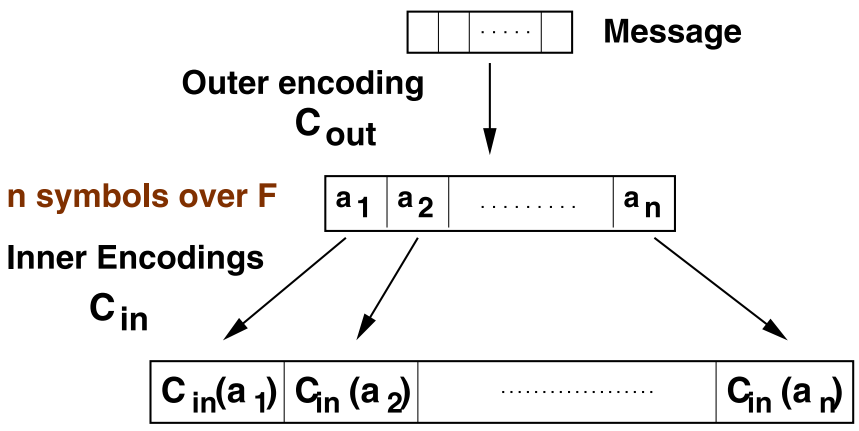

Fig. 2.1. Code concatenation. If the outer code C_{out} is over an alphabet F, and the inner code C_{in} has exactly $|F|$ codewords corresponding to the $|F|$ symbols of F, there is a natural way to combine them by "concatenation".

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 book.

2.2.5 Number-Theoretic Codes

The book 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 analog 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 analogs 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 book. Hence we do not discuss these codes here; the interested reader is pointed to [127, 77] for formal definitions of these codes and details on their properties.

2.3 Detailed Description of Book Chapters

This book presents a 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 in this book 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 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. 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 book.

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 [91].

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 polynomial-sized list is bounded away from the minimum distance of the code.

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 [203, 80, 81].

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 [48, 199], 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 (nontrivial) family of codes that could correct asymptotically more errors than the traditional half the distance bound. Part II of the book 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 [88], and it builds upon the earlier works by Sudan [178] and Ar *et al* [11]. 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 [121] 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 [165]. 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 [86].

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 [89, 80, 90].

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 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 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 [81].

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 [78, 82].

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. Specifically, for every $\varepsilon > 0$ and $0 < r < 1$, we construct codes with rate r that can be encoded in linear time and can be unique decoded from a fraction $(1 - r - \varepsilon)/2$ of errors in linear time. This trade-off between rate and fraction of errors tolerated is optimal since it almost matches the Singleton bound, and in addition we are able to get linear time algorithms. We then concatenate these codes with suitable inner codes to get binary codes that attain the so-called "Zyablov bound" together with linear-time algorithms to perform encoding and decoding up to (almost) half the minimum distance. These linear-time codes significantly improve the fraction of errors corrected by the earlier linear-time codes due to Spielman [176]. Our codes are obtained by using Spielman's codes as a building block and then boosting its error-resilience via suitable expander graphs using techniques from [6, 7]. The results in this chapter are based on joint work with Piotr Indyk [81, 82].

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 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.

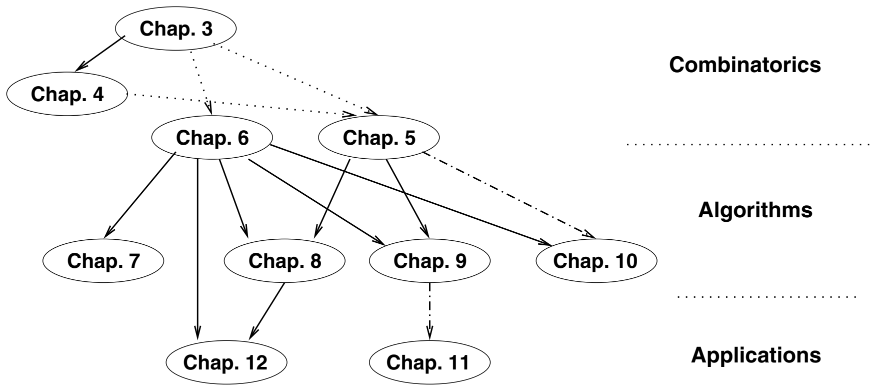

Fig. 2.2. The interrelationship between various chapters. A solid line indicates a dependency (in either techniques or results themselves). A dashed arrow from A to B indicates a "soft" dependency; i.e., reading portions of A prior to B would be helpful, but is not strictly necessary. A dotted line from 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 book 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.