Chapter 2

PRELIMINARIES

In this chapter we will define some basic concepts and notations that will be used throughout this thesis. We will also review some basic results in list decoding that will set the stage for our results. Finally, we will look at some specific families of codes that will crop up frequently in the thesis.

2.1 The Basics

We first fix some notation that will be used frequently in this work, most of which is standard.

For any integer $m \geqslant 1$, we will use $[m]$ to denote the set $\{1, \ldots, m\}$. Given positive integers n and m, we will denote the set of all length n vectors over $[m]$ by $[m]^n$. Unless mentioned otherwise, all vectors in this thesis will be row vectors. $\log x$ will denote the logarithm of x in base 2. $\ln x$ will denote the natural logarithm of x. For bases other than 2 and e, we will specify the base of the logarithm explicitly: for example logarithm of x in base q will be denoted by $\log_q x$.

A finite field with q elements will be denoted by $\mathbb{F}_q$ or $GF(q)$. For any real value x in the range $0 \leqslant x \leqslant 1$, we will use $H_q(x) = x \log_q(q-1) - x \log_q x - (1-x) \log_q(1-x)$ to denote the *q-ary entropy function*. For the special case of $q = 2$, we will simply use $H(x)$ for $H_2(x)$. For more details on the q-ary entropy function, see Section 2.2.2.

For any finite set S, we will use $|S|$ to denote the size of the set.

We now move on to the definitions of the basic notions of error correcting codes.

2.1.1 Basic Definitions for Codes

Let $q \geqslant 2$ be an integer.

Code, Blocklength, Alphabet size :

- An *error correcting code* (or simply a *code*) C is a subset of $[q]^n$ for positive integers q and n. The elements of C are called *codewords*.

- The parameter q is called the *alphabet size* of C. In this case, we will also refer to C as a *q-ary code*. When $q = 2$, we will refer to C as a *binary code*.

- The parameter n is called the *block length* of the code.

Dimension and Rate :

- For a q-ary code C, the quantity $k = \log_q |C|$ is called the *dimension* of the code (this terminology makes more sense for certain classes of codes called linear codes, which we will discuss shortly).

- For a q-ary code C with block length n, its *rate* is defined as the ratio $R = \frac{\log_q |C|}{n}$.

Often it will be useful to use the following alternate way of looking at a code. We will think of a q-ary code C with block length n and $|C| = M$ as a function $[M] \to [q]^n$. Every element x in $[M]$ is called a *message* and $C(x)$ is its *associated codeword*. If M is a power of q, then we will think of the message as length k-vector in $[q]^k$. Viewed this way, C provides a systematic way to add redundancy such that messages of length k over $[q]$ are mapped to n symbols over $[q]$.
(Minimum) Distance and Relative distance : Given any two vectors $\mathbf{v} = \langle v_1, \ldots, v_n \rangle$ and $\mathbf{u} = \langle u_1, \ldots, u_n \rangle$ in $[q]^n$, their *Hamming distance* (or simply distance), denoted by $\Delta(\mathbf{v}, \mathbf{u})$, is the number of positions that they differ in. In other words, $\Delta(\mathbf{v}, \mathbf{u}) = |\{i | u_i \neq v_i\}$.

- The *(minimum) distance* of a code C is the minimum Hamming distance between any two codewords in the code. More 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 of block length n is defined as $\delta = \frac{\mathrm{dist}(C)}{n}$.

2.1.2 Code Families

The focus of this thesis will be on the asymptotic performance of decoding algorithms. For such analysis to make sense, we need to work with an infinite family of codes instead of a single code. In particular, an infinite family of q-ary codes $\mathcal{C}$ is a collection $\{C_i | i \in \mathbb{Z}\}$, where for every i, C_i is a q-ary code of length n_i and $n_i > n_{i-1}$. The rate of the family $\mathcal{C}$ is defined as

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

The relative distance of such a family is defined as

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

From this point on, we will overload notation by referring to an infinite family of codes simply as a code. In particular, from now on, whenever we talk a code C of length n, rate

R and relative distance δ, we will implicitly assume the following. We will think of n as large enough so that its rate R and relative distance δ are (essentially) same as the rate and the relative distance of the corresponding infinite family of codes.

Given this implicit understanding, we can talk about the asymptotics of different algorithms. In particular, we will say that an algorithm that works with a code of block length n is *efficient* if its running time is $O(n^c)$ for some fixed constant c.

2.1.3 Linear Codes

We will now consider an important sub-class of codes called linear codes.

Definition 2.1. *Let q be a prime power. A q-ary code C of block length n is said to be* linear *if it is a linear subspace (over some field $\mathbb{F}_q$) of the vector space $\mathbb{F}_q^n$.*

The size of a q-ary linear code is obviously q^k for some integer k. In fact, it is the dimension of the corresponding subspace in $\mathbb{F}_q^n$. Thus, the dimension of the subspace is same as the dimension of the code. (This is the reason behind the terminology of dimension of a code.)

We will denote a q-ary linear code of dimension k, length n and distance d as an $[n, k, d]_q$ code. (For a general code with the same parameters, we will refer to it as an $(n, k, d)_q$ code.) Most of the time, we will drop the distance part and just refer to the code as an $[n, k]_q$ code. Finally, we will drop the dependence on q if the alphabet size is clear from the context.

We now make some easy observations about q-ary linear codes. First, the zero vector is always a codeword. Second, the minimum distance of a linear code is equal to the minimum *Hamming weight* of the non-zero codewords, where the Hamming weight of a vector is the number of positions with non-zero values.

Any $[n, k]_q$ code C can be defined in the following two ways.

- C can be defined as a set $\{\mathbf{xG}|\mathbf{x} \in \mathbb{F}_q^k\}$, where $\mathbf{G}$ is an $k \times n$ matrix over $\mathbb{F}_q$. $\mathbf{G}$ is called a *generator matrix* of C.

- C can also be characterized by the following subspace $\{\mathbf{c}|\mathbf{c} \in \mathbb{F}_q^n \text{ and } H\mathbf{c}^T = \mathbf{0}\}$, where H is an $(n - k) \times n$ matrix over $\mathbb{F}_q$. H is called the *parity check matrix* of C. The code with H as its generator matrix is called the *dual* of C and is generally denoted by $C^\perp$.

The above two representations imply the following two things for an $[n, k]_q$ code C. First, given the generator matrix $\mathbf{G}$ and a message $\mathbf{x} \in \mathbb{F}_q^n$, one can compute $C(x)$ using $O(nk)$ field operations (by multiplying $\mathbf{x}^T$ with $\mathbf{G}$). Second, given a received word $\mathbf{y} \in \mathbb{F}_q^n$ and the parity check matrix H for C, one can check if $\mathbf{y} \in C$ using $O(n(n-k))$ operations (by computing $H\mathbf{y}$ and checking if it is the all zeroes vector).

Finally, given a q-ary linear code C, we can define the following equivalence relation. $\mathbf{x} \equiv_C \mathbf{y}$ if and only if $\mathbf{x} - \mathbf{y} \in C$. It is easy to check that since C is linear this indeed is an equivalence relation. In particular, $\equiv_C$ partitions $\mathbb{F}_q^n$ into equivalence classes. These are called *cosets* of C (note that one of the cosets is the code C itself). In particular, every coset is of the form $\mathbf{y} + C$, where either $\mathbf{y} = \mathbf{0}$ or $\mathbf{y} \notin C$ and $\mathbf{y} + C$ is shorthand for $\{\mathbf{y} + \mathbf{c} | \mathbf{c} \in C\}$.

We are now ready to talk about definitions and preliminaries for list decoding and property testing of codes.

2.2 Preliminaries and Definitions Related to List Decoding

Recall that list decoding is a relaxation of the decoding problem, where given a received word, the idea is to output all "close-by" codewords. More precisely, given an error bound, we want to output all codewords that lie within the given error bound from the received word. Note that this introduces a new parameter into the mix: the worst case list size. We will shortly define the notion of list decoding that we will be working with in this thesis.

Given integers $q \geqslant 2$, $n \geqslant 1$, $0 \leqslant e \leqslant n$ and a vector $\mathbf{x} \in [q]^n$, we define the *Hamming ball* around $\mathbf{x}$ of *radius* e to be the set of all vectors in $[q]^n$ that are at Hamming distance at most e from $\mathbf{x}$. That is,

$$B_q(\mathbf{x}, e) = \{\mathbf{y} | \mathbf{y} \in [q]^n \textit{ and } \Delta(\mathbf{y}, \mathbf{x}) \leqslant e\}.$$

We will need the following well known result.

Proposition 2.1 ([80]). *Let $q \geqslant 2$ and $e, n \geqslant 1$ be integers such that $e \leqslant (1 - 1/q)n$. Define $\rho = e/n$. Then the following relations are satisfied.*

$$|B_q(\mathbf{0}, e)| = \sum_{i=0}^{e} \binom{n}{i} (q-1)^i \leqslant q^{H_q(e/n)n} = q^{H_q(\rho)n}. \tag{2.1}$$

$$|B_q(\mathbf{0}, e)| \geqslant q^{H_q(\rho)n - o(n)}. \tag{2.2}$$

We will be using the following definition quite frequently.

Definition 2.2 (List-Decodable Codes). *Let C be a q-ary code of block length n. Let $L \geqslant 1$ be an integer and $0 < \rho < 1$ be a real. Then C is called (ρ, L)-list decodable if every Hamming ball of radius ρn has at most L codewords in it. That is, for every $\mathbf{y} \in \mathbb{F}_q^n$, $|B(\mathbf{y}, \rho n) \cap C| \leqslant L$.*

In the definitions above, the parameter L can depend on the block length of the code. In such cases, we will explicitly denote the list size by $L(n)$, where n is the block length.

We will also frequently use the notion of list-decoding radius, which is defined next.

Definition 2.3 (List-Decoding Radius). *Let C be a q-ary code of block length n. Let $0 < \rho < 1$ be a real and define $e = \rho n$. C is said to have a list-decoding radius of ρ (or e) with list size L if ρ (or e) is the* maximum *value for which C is (ρ, L)-list decodable.*

We will frequently use the term *list-decoding radius* without explicitly mentioning the list size in which case the list size is assumed to be at most some fixed polynomial in the block length. Note that one way to show that a code C has a list-decoding radius of at least ρ is to present a polynomial time list-decoding algorithm that can list decode C up to a ρ fraction of errors. Thus, by abuse of notation, given an efficient list-decoding algorithm for a code that can list decode a ρ fraction (or e number) of errors, we will say that the list-decoding algorithm has a list-decoding radius of ρ (or e). In most places, we will be exclusively talking about list-decoding algorithms in which case we will refer to their list-decoding radius as *decoding radius* or just *radius*. In such a case, the code under consideration is said to be list decodable up to the corresponding decoding radius (or just radius). Whenever we are talking about a different notion of decoding (say unique decoding), we will refer to the maximum fraction of errors that a decoder can correct by qualifying the decoding radius with the specific notion of decoding (for example *unique decoding radius*).

We will also use the following generalization of list decoding.

Definition 2.4 (List-Recoverable Codes). *Let C be a q-ary code of block length n. Let $\ell, L \geqslant 1$ be integers and $0 < \rho < 1$ be a real. Then C is called (ρ, ℓ, L)-list recoverable if the following is true. For every sequence of sets $S_1, \ldots, S_n$, where $S_i \subseteq [q]$ and $|S_i| \leqslant \ell$ for every $1 \leqslant i \leqslant n$, there are at most L codewords $\mathbf{c} = \langle c_1, \ldots, c_n \rangle \in C$ such that $c_i \in S_i$ for at least $(1 - \rho)n$ positions i.*

Further, code C is said to (ρ, ℓ)-list recoverable in polynomial time *if it is $(\rho, \ell, L(n))$-list recoverable for some polynomially bounded function $L(\cdot)$, and moreover there is a polynomial time algorithm to find the at most $L(n)$ codewords that are solutions to any $(\rho, \ell, L(n))$-list recovery instance.*

List recovery has been implicitly studied in several works; the name itself was coined in [52]. Note that a $(\rho, 1, L)$-list recoverable code is a (ρ, L)-list decodable code and hence, list recovery is indeed a generalization of list decoding. List recovery is useful in list decoding codes obtained by a certain code composition procedure. The natural list decoder for such a code is a two stage algorithm, where in the first stage the "inner" codes are list decoded to get a sequence of lists, from which one needs to recover codewords from the "outer" code(s). For such an algorithm to be efficient, the outer codes need to be list recoverable.

We next look at the most fundamental tradeoff that we would be interested in for list decoding.

2.2.1 Rate vs. List decodability

In this subsection, we will consider the following question. Given limits $L \geqslant 1$ and $0 < \rho < 1$ on the worst case list size and the fraction of errors that we want to tolerate, what

is the maximum rate that a (ρ, L)-list decodable code can achieve? The following results were implicit in [110] but were formally stated and proved in [35]. We present the proofs for the sake of completeness.

We first start with a positive result.

Theorem 2.1 ([110, 35]). *Let $q \geqslant 2$ be an integer and $0 < \delta \leqslant 1$ be a real. For any integer $L \geqslant 1$ and any real $0 < \rho < 1 - 1/q$, there exists a (ρ, L)-list decodable q-ary code with rate at least $1 - H_q(\rho) - \frac{1}{L+1} - \frac{1}{n^\delta}$.*

Proof. We will prove the result by using the probabilistic method [2]. Choose a code C of block length n and dimension $k = \lceil (1 - H_q(\rho) - \frac{1}{L+1})n - n^{1-\delta} \rceil$ at random. That is, pick each of the q^k codewords in C uniformly (and independently) at random from $[q]^n$. We will show that with high probability, C is (ρ, L)-list decodable.

Let $|C| = M = q^k$. We first fix the received word $\mathbf{y} \in [q]^n$. Consider an $(L+1)$-tuple of codewords $(\mathbf{c}^1, \ldots, \mathbf{c}^{L+1})$ in C. Now if all these codewords fall in a Hamming ball of radius ρn around $\mathbf{y}$, then C is not (ρ, L)-list decodable. In other words, this $(L+1)$-tuple forms a counter-example for C having the required list decodable properties. What is the probability that such an event happens ? For any fixed codeword $\mathbf{c} \in C$, the probability that it lies in $B(\mathbf{y}, \rho n)$ is exactly

$$\frac{|B(\mathbf{y}, \rho n)|}{q^n}.$$

Now since every codeword is picked independently, the probability that the tuple $(\mathbf{c}^1, \ldots, \mathbf{c}^{L+1})$ forms a counter example is

$$\left(\frac{|B(\mathbf{y}, \rho n)|}{q^n} \right)^{L+1} \leqslant q^{-(L+1)n(1-H_q(\rho))},$$

where the inequality follows from Proposition 2.1 (and the fact that the volume of a Hamming ball is translation invariant). Since there are $\binom{M}{L+1} \leqslant M^{L+1}$ different choices of $L+1$ tuples of codewords from C, the probability that there exists at least one $L+1$-tuple that lies in $B(\mathbf{y}, \rho n)$ is at most (by the union bound):

$$M^{L+1} \cdot q^{-(L+1)n(1-H_q(\rho))} = q^{-(L+1)n(1-H_q(\rho)-R)},$$

where $R = k/n$ is the rate of C. Finally, since there are at most q^n choices for $\mathbf{y}$, the probability that there exists some Hamming ball with $L+1$ codewords from C is at most

$$q^n \cdot q^{-(L+1)n(1-H_q(\rho)-R)} = q^{(L+1)n(1-H_q(\rho)-R-1/(L+1)} \leqslant q^{-n^{1-\delta}},$$

where the last inequality follows as $k/n \geqslant 1 - H_q(\rho) - 1/(L+1) - 1/n^\delta$. Thus, with probability $1 - q^{-n^{1-\delta}} > 0$ (for large enough n), C is a (ρ, L)-list decodable code, as desired. □

The following is an immediate consequence of the above theorem.

Corollary 2.2. *Let $q \geqslant 2$ be an integer and $0 < \rho < 1 - 1/q$. For every $\varepsilon > 0$, there exists a q-ary code with rate at least $1 - H_q(\rho) - \varepsilon$ that is $(\rho, O(1/\varepsilon))$-list decodable.*

We now move to an upper bound on the rate of good list decodable codes.

Theorem 2.3 ([110, 35]). *Let $q \geqslant 2$ be an integer and $0 < \rho \leqslant 1 - 1/q$. For every $\varepsilon > 0$, there do not exist any q-ary code with rate $1 - H_q(\rho) + \varepsilon$ that is $(\rho, L(n))$-list decodable for any function $L(n)$ that is polynomially bounded in n.*

Proof. The proof like that of Theorem 2.1 uses the probabilistic method. Let C be any q-ary code of block length n with rate $R = 1 - H_q(\rho) + \varepsilon$. Pick a received word $\mathbf{y}$ uniformly at random from $[q]^n$. Now, the probability that for some fixed $\mathbf{c} \in C$, $\Delta(\mathbf{y}, \mathbf{c}) \leqslant \rho n$ is

$$\frac{|B(\mathbf{0}, \rho n)|}{q^n} \geqslant q^{n(H_q(\rho)-1)-o(n)},$$

where the inequality follows from Proposition 2.1. Thus, the expected number of codewords within a Hamming ball of radius ρn around $\mathbf{y}$ is at least

$$|C| \cdot q^{n(H_q(\rho)-1)-o(n)} = q^{n(R-(1-H_q(\rho)))-o(n)},$$

which by the value of R is $q^{\Omega(n)}$. Since the expected number of codewords is exponential, this implies that there exists a received word $\mathbf{y}$ that has exponentially many codewords from C within a distance ρn from it. Thus, C cannot be $(\rho, L(n))$-list decodable for any polynomially bounded (in fact any subexponential) function $L(\cdot)$. □

List decoding capacity

Theorems 2.1 and 2.3 say that to correct a ρ fraction of errors using list decoding with small list sizes the best rate that one can hope for and can achieve is $1 - H_q(\rho)$. We will call this quantity the *list-decoding capacity*.

The terminology is inspired by the connection of the results above to Shannon's theorem for the special case of the q-symmetric channel (which we will denote by qSC_ρ). In this channel, every symbol (from $[q]$) remains untouched with probability $1 - \rho$ while it is changed to each of the other symbols in $[q]$ with probability $\frac{\rho}{q-1}$. Shannon's theorem states that one can have reliable communication with code of rate less than $1 - H_q(\rho)$ but not with rates larger than $1 - H_q(\rho)$. Thus, Shannon's capacity for qSC_ρ is $1 - H_q(\rho)$, which matches the expression for the list-decoding capacity.

Note that in qSC_ρ, the expected fraction of errors when a codeword is transmitted is ρ. Further, as the errors on each symbol occur independently, the Chernoff bound implies that with high probability the fraction of errors is concentrated around ρ. However, Shannon's proof crucially uses the fact that these (roughly) ρ fraction of errors occur randomly. What Theorems 2.1 and 2.3 say is that even with a ρ fraction of *adversarial* errors[1] one can

[1]Where both the *location* and the *nature* of errors are arbitrary.

have reliable communication via codes of rate $1 - H_q(\rho)$ with list decoding using lists of sufficiently large constant size.

We now consider the list-decoding capacity in some more detail. First we note the following special case of the expression for list-decoding capacity for large enough alphabets. When q is $2^{\Theta(1/\varepsilon)}$, $1 - \rho - \varepsilon$ is a good approximation of $H_q(\rho)$ (see Proposition 2.2). Recall that in Section 1.2, we saw that $1 - \rho$ is the information theoretic limit for codes over any alphabet. The discussion above states that we match this bound for large alphabets.

The proof of Theorem 2.1 uses a general random code. A natural question to ask is if one can prove Theorem 2.1 for special classes of codes: for example, linear codes. For $q = 2$ it is known that Theorem 2.1 is true for linear codes [51]. However, unlike general codes, where Theorem 2.1 (with $\delta < 1$) holds for random codes with high probability, the result in [51] does *not* hold with high probability. For $q > 2$, it is only known that random linear codes (with high probability) are (ρ, L)-list decodable with rate at least $1 - H_q(\rho) - \frac{1}{\log_q(L+1)} - o(1)$.

Achieving List-Decoding Capacity with Explicit Codes

There are two unsatisfactory aspects of Theorem 2.1: (i) The codes are not explicit and (ii) There is no efficient list-decoding algorithm. In light of Theorem 2.1, we can formalize the challenge of list decoding that was posed in Section 1.2.3 as follows:

Grand Challenge. *Let $q \geqslant 2$ and let $0 < \rho < 1 - 1/q$ and $\varepsilon > 0$ be reals. Give an explicit construction[2] of a q-ary code C with rate $1 - H_q(\rho) - \varepsilon$ that is $(\rho, O(1/\varepsilon))$-list decodable. Further, design a polynomial time list-decoding algorithm that can correct ρ fraction of errors while using lists of size $O(1/\varepsilon)$.*

We still do not know how to meet the above grand challenge in its entirety. In Chapter 3, we will show how to meet the challenge above for large enough alphabets (with lists of larger size).

2.2.2 Results Related to the q-ary Entropy Function

We conclude our discussion on list decoding by recording few properties of the q-ary entropy function that will be useful later.

We first start with a calculation where the q-ary entropy function naturally pops up. This hopefully will give the reader a feel for the function (and as a bonus will pretty much prove the lower bound in Proposition 2.1). Let $0 \leqslant x \leqslant 1$ and $q \geqslant 2$. We claim that the quantity $\binom{n}{nx}(q-1)^{nx}$ is approximated very well by $q^{H_q(x)n}$ for large enough n. To see this, let us

[2]By explicit construction, we mean an algorithm that in time polynomial in the block length of the code can output some succinct description of the code. For a linear code, such a description could be the generator matrix of the code.

first use Stirling's approximation of $m!$ by $(m/e)^m$ (for large enough m)[3] to approximate $\binom{n}{nx}$:

$$\binom{n}{nx} = \frac{n!}{(nx)!(n-nx)!} \approx \frac{n^n e^{nx} e^{n-nx}}{(nx)^{nx}(n-nx)^{n-nx}e^n} = \frac{q^{n\log_q n}}{q^{nx\log_q(nx)}q^{n(1-x)\log_q(n(1-x))}}$$
$$= q^{-n(x\log_q x+(1-x)\log_q(1-x))}.$$

Thus, we have

$$\binom{n}{nx}(q-1)^{nx} \approx q^{-n(x\log_q x+(1-x)\log_q(1-x))} \cdot q^{nx\log_q(q-1)} = q^{H_q(x)n},$$

as desired.

Figure 2.1 gives a pictorial view of the q-ary function for the first few values of q.

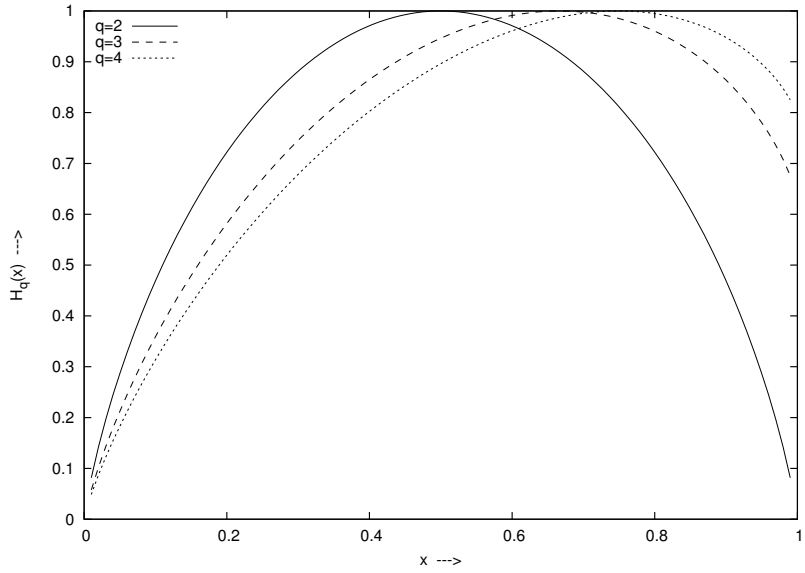

Figure 2.1: A plot of $H_q(x)$ for $q = 2, 3$ and 4. The maximum value of 1 is achieved at $x = 1 - 1/q$.

We now look at the q-ary entropy function for large q.

Proposition 2.2. *For small enough ε, $1 - H_q(\rho) \geqslant 1 - \rho - \varepsilon$ for every $0 < \rho \leqslant 1 - 1/q$ if and only if q is $2^{\Theta(1/\varepsilon)}$.*

[3]There is a $\sqrt{2\pi n}$ factor that we are ignoring.

Proof. We first note that $H_q(\rho) = \rho\log_q(q-1) - \rho\log_q\rho - (1-\rho)\log_q(1-\rho) = \rho\log_q(q-1) + H(\rho)/\log_2 q$. Now if $q = 2^{1/\varepsilon}$, we get that $H_q(\rho) \leqslant \rho + \varepsilon$ as $\log_q(q-1) \leqslant 1$ and $H(\rho) \leqslant 1$. Next we claim that for small enough ε, if $q \geqslant 1/\varepsilon^2$ then $\log_q(q-1) \geqslant 1-\varepsilon$. Indeed, $\log_q(q-1) = 1 + (1/\ln q)\ln(1-1/q) = 1 - O\left(\frac{1}{q\ln q}\right)$, which is at least $1-\varepsilon$ for $q \geqslant 1/\varepsilon^2$. Finally, if $q = 2^{o\left(\frac{1}{\varepsilon}\right)}$ (but $q \geqslant 1/\varepsilon^2$), then for fixed ρ, $H(\rho)/\log q = \varepsilon\cdot\omega(1)$. Then $\rho\log_q(q-1) + H(\rho)/\log q \geqslant \rho - \varepsilon + \varepsilon\cdot\omega(1) > \rho+\varepsilon$, which implies that $1 - H_q(\rho) < 1-\rho-\varepsilon$, as desired. □

Next, we look at the entropy function when its value is very close to 1.

Proposition 2.3. *For small enough $\varepsilon > 0$,*

$$H_q\left(1 - \frac{1}{q} - \varepsilon\right) \leqslant 1 - c_q\varepsilon^2,$$

c_q is constant that only depends on q.

Proof. The intuition behind the proof is the following. Since the derivate of $H_q(x)$ is zero at $x = 1-1/q$, in the Taylor expansion of $H_q(1-1/q-\varepsilon)$ the ε term will vanish. We will now make this intuition more concrete. We will think of q as fixed and $1/\varepsilon$ as growing. In particular, we will assume that $\varepsilon < 1/q$. Consider the following equalities:

$$\begin{aligned}
H_q(1-1/q-\varepsilon) =& -\left(1 - \frac{1}{q} - \varepsilon\right)\log_q\left(\frac{1-1/q-\varepsilon}{q-1}\right) - \left(\frac{1}{q}+\varepsilon\right)\log_q\left(\frac{1}{q}+\varepsilon\right) \\
=& -\log_q\left(\frac{1}{q}\left(1 - \frac{\varepsilon q}{q-1}\right)\right) + \left(\frac{1}{q}+\varepsilon\right)\log_q\left(\frac{1-(\varepsilon q)/(q-1)}{1+\varepsilon q}\right) \\
=& 1 - \frac{1}{\ln q}\left[\ln\left(1 - \frac{\varepsilon q}{q-1}\right) - \left(\frac{1}{q}+\varepsilon\right)\ln\left(\frac{1-(\varepsilon q)/(q-1)}{1+\varepsilon q}\right)\right] \\
=& 1 + o(\varepsilon^2) - \frac{1}{\ln q}\left[-\frac{\varepsilon q}{q-1} - \frac{\varepsilon^2 q^2}{2(q-1)^2} - \left(\frac{1}{q}+\varepsilon\right)\left(-\frac{\varepsilon q}{q-1}\right.\right. \\
& \qquad \left.\left. - \frac{\varepsilon^2 q^2}{2(q-1)^2} - \varepsilon q + \frac{\varepsilon^2 q^2}{2}\right)\right] \qquad (2.3) \\
=& 1 + o(\varepsilon^2) - \frac{1}{\ln q}\left[-\frac{\varepsilon q}{q-1} - \frac{\varepsilon^2 q^2}{2(q-1)^2}\right. \\
& \qquad \left. - \left(\frac{1}{q}+\varepsilon\right)\left(-\frac{\varepsilon q^2}{q-1} + \frac{\varepsilon^2 q^3(q-2)}{2(q-1)^2}\right)\right] \\
=& 1 + o(\varepsilon^2) - \frac{1}{\ln q}\left[-\frac{\varepsilon^2 q^2}{2(q-1)^2} + \frac{\varepsilon^2 q^2}{q-1} - \frac{\varepsilon^2 q^2(q-2)}{2(q-1)^2}\right] \qquad (2.4) \\
=& 1 - \frac{\varepsilon^2 q^2}{2\ln q(q-1)} + o(\varepsilon^2)
\end{aligned}$$

$$\leqslant 1 - \frac{\varepsilon^2 q^2}{4 \ln q(q-1)} \tag{2.5}$$

(2.3) follows from the fact that for $|x| < 1$, $\ln(1+x) = x - x^2/2 + x^3/3 - \ldots$ and by collecting the ε^3 and smaller terms in $o(\varepsilon^2)$. (2.4) follows by rearranging the terms and by absorbing the ε^3 term in $o(\varepsilon^2)$. The last step is true assuming ε is small enough. □

We will also work with the inverse of the q-ary entropy function. Note that $H_q(\cdot)$ on the domain $[0, 1-1/q]$ is an bijective map into $[0,1]$. Thus, we define $H_q^{-1}(y) = x$ such that $H_q(x) = y$ and $0 \leqslant x \leqslant 1 - 1/q$. Finally, we will need the following lower bound.

Lemma 2.4. *For every $0 \leqslant y \leqslant 1 - 1/q$ and for every small enough $\varepsilon > 0$,*

$$H_q^{-1}(y - \varepsilon^2/c_q') \geqslant H_q^{-1}(y) - \varepsilon,$$

where $c_q' \geqslant 1$ is a constant that depends only on q.

Proof. It is easy to check that $H_q^{-1}(y)$ is a strictly increasing convex function in the range $y \in [0,1]$. This implies that the derivate of $H_q^{-1}(y)$ increases with y. In particular, $(H_q^{-1})'(1) \geqslant (H_q^{-1})'(y)$ for every $0 \leqslant y \leqslant 1$. In other words, for every $0 < y \leqslant 1$, and (small enough) $\delta > 0$, $\frac{H_q^{-1}(y) - H_q^{-1}(y-\delta)}{\delta} \leqslant \frac{H_q^{-1}(1) - H_q^{-1}(1-\delta)}{\delta}$. Proposition 2.3 along with the facts that $H_q^{-1}(1) = 1 - 1/q$ and H_q^{-1} is increasing completes the proof if one picks $c_q' = \max(1, 1/c_q)$ and $\delta = \varepsilon^2/c_q'$. □

2.3 Definitions Related to Property Testing of Codes

We first start with some generic definitions. Let $q \geqslant 2$, $n \geqslant 1$ be integers and let $0 < \varepsilon < 1$ be a real. Given a vector $\mathbf{x} \in [q]^n$ and a subset $S \subseteq [q]^n$, we say that $\mathbf{x}$ is *ε-close* to S if there exist a $\mathbf{y} \in S$ such that $\delta(\mathbf{x}, \mathbf{y}) \leqslant \varepsilon$, where $\delta(\mathbf{x}, \mathbf{y}) = \Delta(\mathbf{x}, \mathbf{y})/n$ is the *relative Hamming distance* between $\mathbf{x}$ and $\mathbf{y}$. Otherwise, $\mathbf{x}$ is *ε-far* from S.

Given a q-ary code C of block length n, an integer $r \geqslant 1$ and real $0 < \varepsilon < 1$, we say that a randomized algorithm T_C is an (r, ε)-tester for C if the following conditions hold:

- (Completeness) For every codeword $\mathbf{y} \in C$, $\Pr[T_C(\mathbf{y}) = 1] = 1$, that is, T_C always *accepts* a codeword.
- (Soundness) For every $\mathbf{y} \in [q]^n$ that is ε-far from C, $\Pr[T_C(\mathbf{y}) = 1] \leqslant 1/3$, that is, with probability at least $2/3$, T_C *rejects* $\mathbf{y}$.
- (Query Complexity) For every random choice made by T_C, the tester only probes at most r positions in $\mathbf{y}$.

We remark that the above definition only makes sense when C has large distance. Otherwise we could choose $C = [q]^n$ and the trivial tester that accepts all received words is a $(0, \varepsilon)$-tester. For this thesis, we will adopt the convention that whenever we are taking about testers for a code C, C will have some non trivial distance (in most cases C will have linear distance).

The above kind of tester is also called a *one-sided tester* as it never makes a mistake in the completeness case. Also, the choice of $2/3$ in the soundness case is arbitrary in the following sense. The probability of rejection can be made $1 - \delta$ for any $\delta > 0$, as long as we are happy with $O(r)$ many queries, which is fine for this thesis as we will be interested in the asymptotics of the query complexity. The number of *queries* (or r) can depend on n. Note that there is a gap in the definition of the completeness and soundness of a tester. In particular, the tester can have arbitrary output when the received word $\mathbf{y}$ is not a codeword but is still ε-close to C. In particular, the tester can still reject (very) close-by codewords. We will revisit this in Chapter 8.

We say that a (r, ε) tester is a *local tester* if it makes sub-linear number of queries[4], that is, $r = o(n)$ and ε is some small enough constant. A code is called a *Locally Testable Code* (or *LTC*), if it has a local tester. We also say that a local tester for a code C allows for locally testing C.

2.4 Common Families of Codes

In this section, we will review some code families that will be used frequently in this thesis.

2.4.1 Reed-Solomon Codes

Reed-Solomon codes (named after their inventors [90]) is a linear code that is based on univariate polynomials over finite fields. More formally, an $[n, k+1]_q$ Reed-Solomon code with $k < n$ and $q \geqslant n$ is defined as follows. Let $\alpha_1, \ldots, \alpha_n$ be distinct elements from $\mathbb{F}_q$ (which is why we needed $q \geqslant n$). Every message $\mathbf{m} = \langle m_0, \ldots, m_k \rangle \in \mathbb{F}_q^{k+1}$ is thought of as a degree k polynomial over $\mathbb{F}_q$ by assigning the $k+1$ symbols to the $k+1$ coefficients of a degree k polynomial. In other words, $P_{\mathbf{m}}(X) = m_0 + m_1 X + \cdots + m_k X^k$. The codeword corresponding to $\mathbf{m}$ is defined as follows

$$RS(\mathbf{m}) = \langle P_{\mathbf{m}}(\alpha_1), \ldots, P_{\mathbf{m}}(\alpha_n) \rangle.$$

Now a degree k polynomial can have at most k roots in any field. This implies that any two distinct degree k polynomials can agree in at most k places. In other words,

Proposition 2.5. *An $[n, k+1]_q$ Reed-Solomon code is an $[n, k+1, d = n-k]_q$ code.*

[4]Recall that in this thesis we are implicitly dealing with code families.

By the Singleton bound (see for example [80]), the distance of any code of dimension $k+1$ and length n is at most $n-k$. Thus, Reed-Solomon codes have the optimal distance: such codes are called *Maximum Distance Separable* (or *MDS*) codes. The MDS property along with its nice algebraic structure has made Reed-Solomon code the center of a lot of research in coding theory. In particular, the algebraic properties of these codes have been instrumental in the algorithmic progress in list decoding [97, 63, 85]. In addition to their nice theoretical applications, Reed-Solomon codes have found widespread use in practical applications. In particular, these codes are used in CDs, DVDs and other storage media, deep space communications, DSL and paper bar codes. We refer the reader to [105] for more details on some of these applications of Reed-Solomon codes.

2.4.2 Reed-Muller Codes

Reed-Muller codes are generalization of Reed-Solomon codes. For integers $\ell \geqslant 1$ and $m \geqslant 1$, the message space is the set of all polynomials over $\mathbb{F}_q$ in ℓ variables that have total degree at most m. The codeword corresponding to a message is the evaluation of the corresponding ℓ-variate polynomial over n distinct points in $\mathbb{F}_q^\ell$ (note that this requires $q^\ell \geqslant n$). Finally, note that when $\ell = 1$ and $m = k$, we get an $[n, k+1]_q$ Reed-Solomon code. Interestingly, Reed-Muller codes [82, 89] were discovered before Reed-Solomon codes.

2.5 Basic Finite Field Algebra

We will be using a fair amount of finite field algebra in the thesis. In this section, we recap some basic notions and facts about finite fields.

A field consists of a set of elements that is closed under addition, multiplication and (both additive and multiplicative) inversion. It also has two special elements 0 and 1, which are the additive and multiplicative identities respectively. A field is called a *finite field* if its set of elements is finite. The set of integers modulo some prime p, form the finite field $\mathbb{F}_p$.

The ring of univariate polynomials with coefficients from $\mathbb{F}$ will be denoted by $\mathbb{F}[X]$. A polynomial $E(X)$ is said to be irreducible if for every way of writing $E(X) = A(X) \cdot B(X)$, either $A(X)$ or $B(X)$ is a constant polynomial. A polynomial is called *monic*, if the coefficient of its leading term is 1.

If $E(X)$ is an irreducible polynomial of degree d over a field $\mathbb{F}$, then the quotient ring $\mathbb{F}[X]/(E(X))$, consisting of all polynomials in $\mathbb{F}[X]$ modulo $E(X)$ is itself a finite field and is called *field extension* of $\mathbb{F}$. The extension field also forms a vector space of dimension d over $\mathbb{F}$.

All finite fields are either $\mathbb{F}_p$ for prime p or is an extension of a prime field. Thus, the number of elements in a finite field is a prime power. Further, for any prime power q there exists only one finite field (up to isomorphism). For any q that is a power of prime p, the field $\mathbb{F}_q$ has *characteristic* of p. The multiplicative groups of non-zero elements of a field

$\mathbb{F}_q$, denoted by $\mathbb{F}_q^*$, is known to be cyclic. In other words, $\mathbb{F}_q^* = \{1, \gamma, \gamma^2, \ldots, \gamma^{q-2}\}$ for some element $\gamma \in \mathbb{F}_q \setminus \{0\}$. γ is also called the *primitive element* or *generator* of $\mathbb{F}_q^*$.

The following property of finite fields will be crucial. Any polynomial $f(X)$ of degree at most d in $\mathbb{F}[X]$ has at most d roots, where $\alpha \in \mathbb{F}$ is a root of $f(X)$ if $f(\alpha) = 0$. We would be also interested in finding roots of univariate polynomials (over extension fields) for which we will use a classical algorithm due to Berlekamp [16].

Theorem 2.4 ([16]). *Let p be a prime. There exists a deterministic algorithm that on input a polynomial in $\mathbb{F}_{p^t}[X]$ of degree d, can find all the irreducible factors (and hence the roots) in time polynomial in d, p and t.*