Essential Coding Theory

Venkatesan Guruswami Atri Rudra[1] Madhu Sudan

July 27, 2018

[1]Department of Computer Science and Engineering, University at Buffalo, SUNY. Work supported by NSF CAREER grant CCF-0844796.

Foreword

This chapter is based on lecture notes from coding theory courses taught by Venkatesan Guruswami at University at Washington and CMU; by Atri Rudra at University at Buffalo, SUNY and by Madhu Sudan at Harvard and MIT.

This version is dated **July 27, 2018**. For the latest version, please go to

http://www.cse.buffalo.edu/faculty/atri/courses/coding-theory/book/

The material in this chapter is supported in part by the National Science Foundation under CAREER grant CCF-0844796. Any opinions, findings and conclusions or recomendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation (NSF).

Contents

List of Figures

List of Tables

List of Algorithms

Part I

The Basics

Chapter 1

The Fundamental Question

1.1 Overview

Communication is a fundamental need of our modern lives. In fact, communication is something that humans have been doing for a long time. For simplicity, let us restrict ourselves to English. It is quite remarkable that different people speaking English can be understood pretty well: even if e.g. the speaker has an accent. This is because English has some built-in redundancy, which allows for “errors" to be tolerated. This came to fore for one of the authors when his two and a half year old son, Akash, started to speak his own version of English, which we will dub “Akash English." As an example,

Figure 1.1: Decoding in Akash English, one gets “I need little little (trail)mix."

With some practice Akash’s parents were able to “decode" what Akash really meant. In fact,

Akash could communicate even if he did not say an entire word properly and gobbled up part(s) of word(s).

The above example shows that having redundancy in a language allows for communication even in the presence of (small amounts of) differences and errors. Of course in our modern digital world, all kinds of entities communicate (and most of the entities do not communicate in English or any natural language for that matter.) Errors are also present in the digital world, so these digital communications also use redundancy.

Error-correcting codes (henceforth, just codes) are clever ways of representing data so that one can recover the original information even if parts of it are corrupted. The basic idea is to judiciously introduce redundancy so that the original information can be recovered even when parts of the (redundant) data have been corrupted.

For example, when packets are transmitted over the Internet, some of the packets get corrupted or dropped. Packet drops are resolved by the TCP layer by a combination of sequence numbers and ACKs. To deal with data corruption, multiple layers of the TCP/IP stack use a form of error correction called CRC Checksum [59]. From a theoretical point of view, the checksum is a terrible code (for that matter so is English). However, on the Internet, the current dominant mode of operation is to detect errors and if errors have occurred, then ask for retransmission. This is the reason why the use of checksum has been hugely successful in the Internet. However, there are other communication applications, where re-transmission is not an option. Codes are used when transmitting data over the telephone line or via cell phones. They are also used in deep space communication and in satellite broadcast (for example, TV signals are transmitted via satellite). Indeed, asking the Mars Rover to re-send an image just because it got corrupted during transmission is not an option– this is the reason that for such applications, the codes used have always been very sophisticated.

Codes also have applications in areas not directly related to communication. In particular, in the applications above, we want to communicate over space. Codes can also be used to communicate over time. For example, codes are used heavily in data storage. CDs and DVDs work fine even in presence of scratches precisely because they use codes. Codes are used in Redundant Array of Inexpensive Disks (RAID) [9] and error correcting memory [8]. (Sometimes in the Blue Screen of Death displayed by Microsoft Windows family of operating systems, you might see a line saying something along the lines of “parity check failed"– this happens when the code used in the error-correcting memory cannot recover from error(s). Also for certain consumers of memory, e.g. banks, do not want to suffer from even one bit flipping (this e.g. could mean someone's bank balance either got halved or doubled– neither of which are welcome.[1]) Codes are also deployed in other applications such as paper bar codes, for example, the bar code used by UPS called MaxiCode [7]. Unlike the Internet example, in all of these applications, there is no scope for “re-transmission."

In this book, we will mainly think of codes in the communication scenario. In this framework, there is a sender who wants to send (say) k message symbols over a noisy channel. The sender first *encodes* the k message symbols into n symbols (called a *codeword*) and then sends

[1] This is a bit tongue-in-cheek: in real life banks have more mechanisms to prevent one bit flip from wreaking havoc.

it over the *channel*. The receiver gets a *received word* consisting of n symbols. The receiver then tries to *decode* and recover the original k message symbols. Thus, encoding is the process of adding redundancy and decoding is the process of removing errors.

Unless mentioned otherwise, in this book we will make the following assumption

The sender and the receiver only communicate via the channel.[a] In other words, other then some setup information about the code, the sender and the receiver do not have any other information exchange (other than of course what was transmitted over the channel). In particular, no message is more likely to be transmitted over another.

[a] The scenario where the sender and receiver have a "side-channel" is an interesting topic that has been studied but it outside the scope of this book.

The fundamental question that will occupy our attention for almost the entire book is the tradeoff between the amount of redundancy used and the number of errors that can be corrected by a code. In particular, we would like to understand

Question 1.1.1. *How much redundancy do we need to correct a given amount of errors? (We would like to correct as many errors as possible with as little redundancy as possible.)*

Intuitively, maximizing error correction and minimizing redundancy are contradictory goals: a code with higher redundancy should be able to tolerate more number of errors. By the end of this chapter, we will see a formalization of this question.

Once we determine the optimal tradeoff, we will be interested in achieving the optimal tradeoff with codes with *efficient* encoding and decoding. (A DVD player that tells its consumer that it will recover from a scratch on a DVD by tomorrow is not going to be exactly a best-seller.) In this book, we will primarily define efficient algorithms to be ones that run in polynomial time.[2]

1.2 Some definitions and codes

To formalize Question 1.1.1, we begin with the definition of a code.

Definition 1.2.1 (Code)**.** A code of *block length* n over an *alphabet* Σ is a subset of Σ^n. Typically, we will use q to denote $|\Sigma|$.[3]

Remark 1.2.1. We note that the ambient space Σ^n, viewed as a set of sequences, vectors or functions. In other words, we can think of a vector $(v_1, \ldots, v_n) \in \Sigma^n$ as just the sequence $v_1, \ldots, v_n$ (in order) or a vector tuple $(v_1, \ldots, v_n)$ or as the function $f : [n] \to \Sigma$ such that $f(i) = v_i$. Sequences

[2] We are not claiming that this is the correct notion of efficiency in practice. However, we believe that it is a good definition as the "first cut"– quadratic or cubic time algorithms are definitely more desirable than exponential time algorithms: see Section C.4 for more on this.

[3] Note that q need not be a constant and can depend on n: we'll see codes in this book where this is true.

assume least structure on Σ and hence, are most generic. Vectors work well when Σ has some structure (and in particular is what is known as a *field*, which we will see next chapter). Functions work when the set of coordinates has structure (e.g., [n] may come from a finite field of size n). In such cases functional representation will be convenient. For now however the exact representation does not matter and the reader can work with representation as sequences.

We will also frequently use the following alternate way of looking at a code. Given a code $C \subseteq \Sigma^n$, with $|C| = M$, we will think of C as a mapping of the following form:

$$C : [M] \to \Sigma^n,$$

where $[x]$ for any integer $x \geq 1$ denotes the set $\{1, 2, , \ldots, x\}$.

We will also need the notion of *dimension* of a code.

Definition 1.2.2 (Dimension of a code)**.** Given a code $C \subseteq \Sigma^n$, its *dimension* is given by

$$k \stackrel{\text{def}}{=} \log_q |C|.$$

Let us begin by looking at two specific codes. Both codes are defined over $\Sigma = \{0, 1\}$ (also known as *binary codes*). In both cases $|C| = 2^4$ and we will think of each of the 16 messages as a 4 bit vector.

We first look at the so called *parity code*, which we will denote by $C_{\oplus}$. Given a message $(x_1, x_2, x_3, x_4) \in \{0, 1\}^4$, its corresponding codeword is given by

$$C_{\oplus}(x_1, x_2, x_3, x_4) = (x_1, x_2, x_3, x_4, x_1 \oplus x_2 \oplus x_3 \oplus x_4),$$

where the $\oplus$ denotes the EXOR (also known as the XOR or Exclusive-OR) operator. In other words, the parity code appends the parity of the message bits (or taking the remainder of the sum of the message bits when divided by 2) at the end of the message. Note that such a code uses the minimum amount of non-zero redundancy.

The second code we will look at is the so called *repetition code*. This is a very natural code (and perhaps the first code one might think of). The idea is to repeat every message bit a fixed number of times. For example, we repeat each of the 4 message bits 3 times and we use $C_{3,rep}$ to denote this code.

Let us now try to look at the tradeoff between the amount of redundancy and the number of errors each of these codes can correct. Even before we begin to answer the question, we need to define how we are going to measure the amount of redundancy. One natural way to define redundancy for a code with dimension k and block length n is by their difference $n - k$. By this definition, the parity code uses the least amount of redundancy. However, one "pitfall" of such a definition is that it does not distinguish between a code with $k = 100$ and $n = 102$ and another code with dimension and block length 2 and 4 respectively. Intuitively the latter code is using more redundancy. This motivates the following notion of measuring redundancy.

Definition 1.2.3 (Rate of a code)**.** The *rate* of a code with dimension k and block length n is given by

$$R \stackrel{\text{def}}{=} \frac{k}{n}.$$

Note that higher the rate, lesser the amount of redundancy in the code. Also note that as $k \leq n$, $R \leq 1$.[4] Intuitively, the rate of a code is the average amount of real information in each of the n symbols transmitted over the channel. So in some sense rate captures the complement of redundancy. However, for historical reasons we will deal with the rate R (instead of the more obvious $1-R$) as our notion of redundancy. Given the above definition, $C_{\oplus}$ and $C_{3,rep}$ have rates of $\frac{4}{5}$ and $\frac{1}{3}$. As was to be expected, the parity code has a higher rate than the repetition code.

We have formalized the notion of redundancy as the rate of a code. To formalize Question 1.1.1, we need to formally define what it means to correct errors. We do so next.

1.3 Error correction

Before we formally define error correction, we will first formally define the notion of *encoding*.

Definition 1.3.1 (Encoding function)**.** Let $C \subseteq \Sigma^n$. An equivalent description of the code C is by an injective mapping $E : [|C|] \rightarrow \Sigma^n$ called the encoding function.

Next we move to error correction. Intuitively, we can correct a received word if we can recover the transmitted codeword (or equivalently the corresponding message). This "reverse" process is called *decoding*.

Definition 1.3.2 (Decoding function)**.** Let $C \subseteq \Sigma^n$ be a code. A mapping $D : \Sigma^n \rightarrow [|C|]$ is called a decoding function for C.

The definition of a decoding function by itself does not give anything interesting. What we really need from a decoding function is that it recovers the transmitted message. This notion is captured next.

Definition 1.3.3 (Error Correction)**.** Let $C \subseteq \Sigma^n$ and let $t \geq 1$ be an integer. C is said to be *t-error-correcting* if there exists a decoding function D such that for every message $\mathbf{m} \in [|C|]$ and error pattern $\mathbf{e}$ with at most t errors, $D(C(\mathbf{m}) + \mathbf{e})) = \mathbf{m}$.

Figure 1.3 illustrates how the definitions we have examined so far interact.

We will also very briefly look at a weaker form of error recovery called *error detection*.

Definition 1.3.4 (Error detection)**.** Let $C \subseteq \Sigma^n$ and let $t \geq 1$ be an integer. C is said to be *t-error-detecting* if there exists a detecting procedure D such that for every message $\mathbf{m}$ and every error pattern $\mathbf{e}$ with at most t errors, D outputs a 1 if $(C(\mathbf{m}) + \mathbf{e}) \in C$ and 0 otherwise.

Note that a t-error correcting code is also a t-error detecting code (but not necessarily the other way round): see Exercise 1.1. Although error detection might seem like a weak error recovery model, it is useful in settings where the receiver can ask the sender to re-send the message. For example, error detection is used quite heavily in the Internet.

With the above definitions in place, we are now ready to look at the error correcting capabilities of the codes we looked at in the previous section.

[4] Further, in this book, we will always consider the case $k > 0$ and $n < \infty$ and hence, we can also assume that $R > 0$.

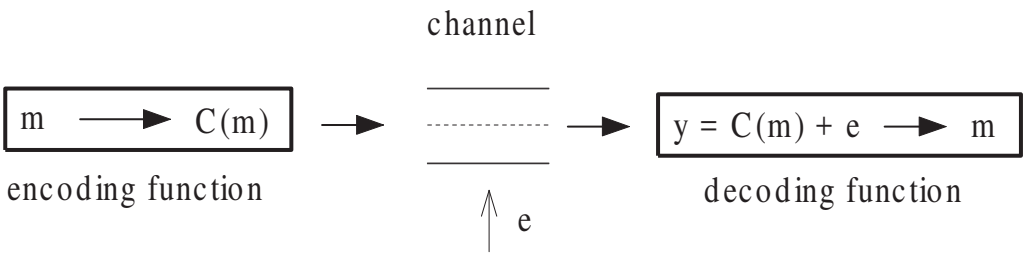


Figure 1.2: Coding process

1.3.1 Error-Correcting Capabilities of Parity and Repetition codes

In Section 1.2, we looked at examples of parity code and repetition code with the following properties:

$$C_{\oplus} : q = 2, k = 4, n = 5, R = 4/5.$$

$$C_{3,rep} : q = 2, k = 4, n = 12, R = 1/3.$$

We will start with the repetition code. To study its error-correcting capabilities, we will consider the following natural decoding function. Given a received word $\mathbf{y} \in \{0,1\}^{12}$, divide it up into four consecutive blocks (y_1, y_2, y_3, y_4) where every block consists of three bits. Then, for every block y_i $(1 \le i \le 4)$, output the majority bit as the message bit. We claim this decoding function can correct any error pattern with at most 1 error. (See Exercise 1.2.) For example, if a block of 010 is received, since there are two 0's we know the original message bit was 0. In other words, we have argued that

Proposition 1.3.1. *$C_{3,rep}$ is a 1-error correcting code.*

However, it is not too hard to see that $C_{3,rep}$ cannot correct two errors. For example, if both of the errors happen in the same block and a block in the received word is 010, then the original block in the codeword could have been either 111 or 000. Therefore in this case, no decoder can successfully recover the transmitted message.[5]

Thus, we have pin-pointed the error-correcting capabilities of the $C_{3,rep}$ code: it can correct one error, but not two or more. However, note that the argument assumed that the error positions can be located arbitrarily. In other words, we are assuming that the channel noise behaves arbitrarily (subject to a bound on the total number of errors). Obviously, we can model the noise differently. We now briefly digress to look at this issue in slightly more detail.

Digression: Channel Noise. As was mentioned above, until now we have been assuming the following noise model, which was first studied by Hamming:

[5] Recall we are assuming that the decoder has no side information about the transmitted message.

> Any error pattern can occur during transmission as long as the total number of errors is bounded. Note that this means that the location as well as the nature[6] of the errors is arbitrary.

We will frequently refer to Hamming's model as the Adversarial Noise Model. It is important to note that the atomic unit of error is a symbol from the alphabet. So for example, if the error pattern is $(1,0,1,0,0,0)$ and we consider the alphabet to be $\{0,1\}$, then the pattern has two errors. However, if our alphabet is $\{0,1\}^3$ (i.e. we think of the vector above as $((1,0,1),(0,0,0))$, with $(0,0,0)$ corresponding to the zero element in $\{0,1\}^3$), then the pattern has only one error. Thus, by increasing the alphabet size we can also change the adversarial noise model. As the book progresses, we will see how error correction over a larger alphabet is easier than error correction over a smaller alphabet.

However, the above is not the only way to model noise. For example, we could also have following error model:

> No more than 1 error can happen in any contiguous three-bit block.

First note that, for the channel model above, no more than four errors can occur when a codeword in $C_{3,rep}$ is transmitted. (Recall that in $C_{3,rep}$, each of the four bits is repeated three times.) Second, note that the decoding function that takes the majority vote of each block can successfully recover the transmitted codeword for *any* error pattern, while in the worst-case noise model it could only correct at most one error. This channel model is admittedly contrived, but it illustrates the point that the error-correcting capabilities of a code (and a decoding function) are crucially dependent on the noise model.

A popular alternate noise model is to model the channel as a stochastic process. As a concrete example, let us briefly mention the *binary symmetric channel with crossover probability* $0 \leq p \leq 1$, denoted by BSC_p, which was first studied by Shannon. In this model, when a (binary) codeword is transferred through the channel, every bit flips independently with probability p.

Note that the two noise models proposed by Hamming and Shannon are in some sense two extremes: Hamming's model assumes *no* knowledge about the channel (except that a bound on the total number of errors is known[7] while Shannon's noise model assumes *complete* knowledge about how noise is produced. In this book, we will consider only these two extreme noise models. In real life, the situation often is somewhere in between.

For real life applications, modeling the noise model correctly is an extremely important task, as we can tailor our codes to the noise model at hand. However, in this book we will not study this aspect of designing codes at all, and will instead mostly consider the worst-case noise model. Intuitively, if one can communicate over the worst-case noise model, then one could use the same code to communicate over pretty much every other noise model with the same amount of noise.

[6] For binary codes, there is only one kind of error: a bit flip. However, for codes over a larger alphabet, say $\{0,1,2\}$, 0 being converted to a 1 and 0 being converted into a 2 are both errors, but are different kinds of errors.

[7] A bound on the total number of errors is necessary; otherwise, error correction would be impossible: see Exercise 1.3.

We now return to $C_\oplus$ and examine its error-correcting capabilities in the worst-case noise model. We claim that $C_\oplus$ cannot correct even one error. Suppose 01000 is the received word. Then we know that an error has occurred, but we do not know which bit was flipped. This is because the two codewords 00000 and 01001 differ from the received word 01000 in exactly one bit. As we are assuming that the receiver has no side information about the transmitted codeword, no decoder can know what the transmitted codeword was.

Thus, since it cannot correct even one error, from an error-correction point of view, $C_\oplus$ is a terrible code (as it cannot correct even 1 error). However, we will now see that $C_\oplus$ can *detect* one error. Consider the following algorithm. Let $\mathbf{y} = (y_1, y_2, y_3, y_4, y_5)$ be the received word. Compute $b = y_1 \oplus y_2 \oplus y_3 \oplus y_4 \oplus y_5$ and declare an error if $b = 1$. Note that when no error has occurred during transmission, $y_i = x_i$ for $1 \leq i \leq 4$ and $y_5 = x_1 \oplus x_2 \oplus x_3 \oplus x_4$, in which case $b = 0$ as required. If there is a single error then either $y_i = x_i \oplus 1$ (for exactly one $1 \leq i \leq 4$) or $y_5 = x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus 1$. It is easy to check that in this case $b = 1$. In fact, one can extend this argument to obtain the following result (see Exercise 1.4).

Proposition 1.3.2. *The parity code $C_\oplus$ can* detect *an odd number of errors.*

Let us now revisit the example that showed that one cannot correct one error using $C_\oplus$. Consider two codewords in $C_\oplus$, $\mathbf{u} = 00000$ and $\mathbf{v} = 10001$ (which are codewords corresponding to messages 0000 and 1000, respectively). Now consider the scenarios in which $\mathbf{u}$ and $\mathbf{v}$ are each transmitted and a single error occurs resulting in the received word $\mathbf{r} = 10000$. Thus, given the received word $\mathbf{r}$ and the fact that at most one error can occur, the decoder has no way of knowing whether the original transmitted codeword was $\mathbf{u}$ or $\mathbf{v}$. Looking back at the example, it is clear that the decoder is "confused" because the two codewords $\mathbf{u}$ and $\mathbf{v}$ do not differ in many positions. This notion is formalized in the next section.

1.4 Distance of a code

We now define a notion of distance that captures the concept that the two vectors $\mathbf{u}$ and $\mathbf{v}$ are "close-by."

Definition 1.4.1 (Hamming distance)**.** Given $\mathbf{u}, \mathbf{v} \in \Sigma^n$ (i.e. two vectors of length n) the *Hamming distance* between $\mathbf{u}$ and $\mathbf{v}$, denoted by $\Delta(\mathbf{u}, \mathbf{v})$, is defined to be the number of positions in which $\mathbf{u}$ and $\mathbf{v}$ differ.

The Hamming distance is a distance in a very formal mathematical sense: see Exercise 1.5. Note that the definition of Hamming distance just depends on the *number* of differences and not the nature of the difference. For example, for the vectors $\mathbf{u}$ and $\mathbf{v}$ from the previous section, $\Delta(\mathbf{u}, \mathbf{v}) = 2$, which is equal to the Hamming distance $\Delta(\mathbf{u}, \mathbf{w})$, where $\mathbf{w} = 01010$, even though the vectors $\mathbf{v}$ and $\mathbf{w}$ are not equal.

Armed with the notion of Hamming distance, we now define another important parameter of a code.

Definition 1.4.2 (Minimum distance)**.** Let $C \subseteq \Sigma^n$. The *minimum distance* (or just *distance*) of C is defined to be

$$d = \min_{\mathbf{c}_1 \neq \mathbf{c}_2 \in C} \Delta(\mathbf{c}_1, \mathbf{c}_2)$$

It is easy to check that the repetition code $C_{3,rep}$ has distance 3. (Indeed, any two distinct messages will differ in at least one of the message bits. After encoding, the difference in one message bit will translate into a difference of three bits in the corresponding codewords.) We now claim that the distance of $C_{\oplus}$ is 2. This is a consequence of the following observations. If two messages $\mathbf{m}_1$ and $\mathbf{m}_2$ differ in at least two places then $\Delta(C_{\oplus}(\mathbf{m}_1), C_{\oplus}(\mathbf{m}_2)) \geq 2$ (by only looking at the first four bits of the codewords). If two messages differ in exactly one place then the parity bits in the corresponding codewords are different which implies a Hamming distance of 2 between the codewords. Thus, $C_{\oplus}$ has smaller distance than $C_{3,rep}$ and can correct less number of errors than $C_{3,rep}$, which seems to suggest that a larger distance implies greater error-correcting capabilities. The next result formalizes this intuition.

Proposition 1.4.1. *The following statements are equivalent for a code C:*

1. *C has minimum distance $d \geq 2$,*
2. *If d is odd, C can correct $(d-1)/2$ errors.*
3. *C can detect $d-1$ errors.*
4. *C can correct $d-1$ erasures.*[8]

Remark 1.4.1. Property (2) above for even d is slightly different. In this case, one can correct up to $\frac{d}{2}-1$ errors but cannot correct $\frac{d}{2}$ errors. (See Exercise 1.6.)

Before we prove Proposition 1.4.1, let us apply it to the codes $C_{\oplus}$ and $C_{3,rep}$ which have distances of 2 and 3 respectively. Proposition 1.4.1 implies the following facts that we have already proved:

- $C_{3,rep}$ can correct 1 errors (Proposition 1.3.1).
- $C_{\oplus}$ can detect 1 error but cannot correct 1 error (Proposition 1.3.2).

The proof of Proposition 1.4.1 will need the following decoding function. *Maximum likelihood decoding* (MLD) is a well-studied decoding method for error correcting codes, which outputs the codeword closest to the received word in Hamming distance (with ties broken arbitrarily). More formally, the MLD function denoted by $D_{MLD} : \Sigma^n \to C$ is defined as follows. For every $\mathbf{y} \in \Sigma^n$,

$$D_{MLD}(\mathbf{y}) = \arg\min_{\mathbf{c} \in C} \Delta(\mathbf{c}, \mathbf{y}).$$

Algorithm 1 is a naive implementation of the MLD.

[8] In the erasure noise model, the receiver *knows* where the errors have occurred. In this model, an erroneous symbol is denoted by "?", with the convention that any non-? symbol is a correct symbol.

Algorithm 1 Naive Maximum Likelihood Decoder

INPUT: Received word $\mathbf{y} \in \Sigma^n$
OUTPUT: $D_{MLD}(\mathbf{y})$

1: Pick an arbitrary $\mathbf{c} \in C$ and assign $\mathbf{z} \leftarrow \mathbf{c}$
2: FOR every $\mathbf{c}' \in C$ such that $\mathbf{c} \neq \mathbf{c}'$ DO
3: IF $\Delta(\mathbf{c}', \mathbf{y}) < \Delta(\mathbf{z}, \mathbf{y})$ THEN
4: $\mathbf{z} \leftarrow \mathbf{c}'$
5: RETURN $\mathbf{z}$

Proof of Proposition 1.4.1 We will complete the proof in two steps. First, we will show that if property 1 is satisfied then so are properties 2,3 and 4. Then we show that if property 1 is not satisfied then none of properties 2,3 or 4 hold.

1. **implies** 2. Assume C has distance d. We first prove 2 (for this case assume that $d = 2t+1$). We now need to show that there exists a decoding function such that for all error patterns with at most t errors it always outputs the transmitted message. We claim that the MLD function has this property. Assume this is not so and let $\mathbf{c}_1$ be the transmitted codeword and let $\mathbf{y}$ be the received word. Note that

$$\Delta(\mathbf{y}, \mathbf{c}_1) \leq t. \tag{1.1}$$

As we have assumed that MLD does not work, $D_{MLD}(\mathbf{y}) = \mathbf{c}_2 \neq \mathbf{c}_1$. Note that by the definition of MLD,

$$\Delta(\mathbf{y}, \mathbf{c}_2) \leq \Delta(\mathbf{y}, \mathbf{c}_1). \tag{1.2}$$

Consider the following set of inequalities:

$$\begin{aligned} \Delta(\mathbf{c}_1, \mathbf{c}_2) &\leq \Delta(\mathbf{c}_2, \mathbf{y}) + \Delta(\mathbf{c}_1, \mathbf{y}) && (1.3)\\ &\leq 2\Delta(\mathbf{c}_1, \mathbf{y}) && (1.4)\\ &\leq 2t && (1.5)\\ &= d-1, && (1.6) \end{aligned}$$

where (1.3) follows from the triangle inequality (see Exercise 1.5), (1.4) follows from (1.2) and (1.5) follows from (1.1). (1.6) implies that the distance of C is at most $d-1$, which is a contradiction.

1. **implies** 3. We now show that property 3 holds, that is, we need to describe an algorithm that can successfully detect whether errors have occurred during transmission (as long as the total number of errors is bounded by $d-1$). Consider the following error detection algorithm: check if the received word $\mathbf{y} = \mathbf{c}$ for some $\mathbf{c} \in C$ (this can be done via an exhaustive check). If no errors occurred during transmission, $\mathbf{y} = \mathbf{c}_1$, where $\mathbf{c}_1$ was the transmitted codeword and the algorithm above will accept (as it should). On the other hand if $1 \leq \Delta(\mathbf{y}, \mathbf{c}_1) \leq d-1$, then by the fact that the distance of C is d, $\mathbf{y} \notin C$ and hence the algorithm rejects, as required.

1. **implies** 4. Finally, we prove that property 4 holds. Let $\mathbf{y} \in (\Sigma \cup \{?\})^n$ be the received word. First we claim that there is a unique $\mathbf{c} = (c_1, \ldots, c_n) \in C$ that agrees with $\mathbf{y}$ (i.e. $y_i = c_i$ for every i such that $y_i \neq ?$). (For the sake of contradiction, assume that this is not true, i.e. there exists two distinct codewords $\mathbf{c}_1, \mathbf{c}_2 \in C$ such that both $\mathbf{c}_1$ and $\mathbf{c}_2$ agree with $\mathbf{y}$ in the unerased positions. Note that this implies that $\mathbf{c}_1$ and $\mathbf{c}_2$ agree in the positions i such that $y_i \neq ?$. Thus, $\Delta(\mathbf{c}_1, \mathbf{c}_2) \leq |\{i | y_i = ?\}| \leq d - 1$, which contradicts the assumption that C has distance d.) Given the uniqueness of the codeword $\mathbf{c} \in C$ that agrees with $\mathbf{y}$ in the unerased position, here is an algorithm to find it: go through all the codewords in C and output the desired codeword.

$\neg 1$. **implies** $\neg 2$. For the other direction of the proof, assume that property 1 does not hold, that is, C has distance $d - 1$. We now show that property 2 cannot hold, that is, for every decoding function there exists a transmitted codeword $\mathbf{c}_1$ and a received word $\mathbf{y}$ (where $\Delta(\mathbf{y}, \mathbf{c}_1) \leq (d-1)/2$) such that the decoding function cannot output $\mathbf{c}_1$. Let $\mathbf{c}_1 \neq \mathbf{c}_2 \in C$ be codewords such that $\Delta(\mathbf{c}_1, \mathbf{c}_2) = d - 1$ (such a pair exists as C has distance $d - 1$). Now consider a vector $\mathbf{y}$ such that $\Delta(\mathbf{y}, \mathbf{c}_1) = \Delta(\mathbf{y}, \mathbf{c}_2) = (d - 1)/2$. Such a $\mathbf{y}$ exists as d is odd and by the choice of $\mathbf{c}_1$ and $\mathbf{c}_2$. Below is an illustration of such a $\mathbf{y}$ (same color implies that the vectors agree on those positions):

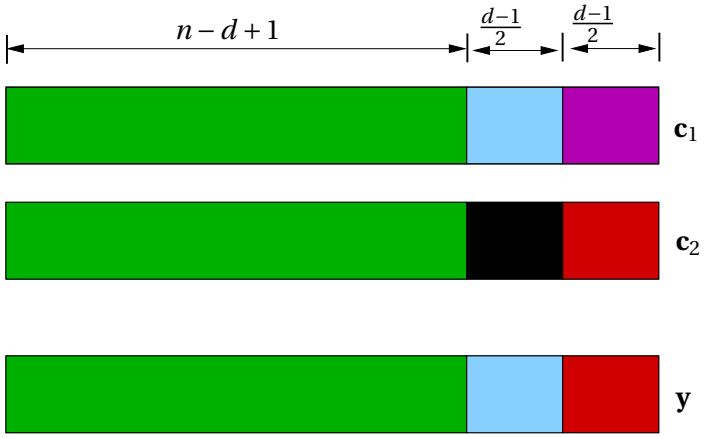


Figure 1.3: Bad example for unique decoding.

Now, since $\mathbf{y}$ could have been generated if *either* of $\mathbf{c}_1$ or $\mathbf{c}_2$ were the transmitted codeword, no decoding function can work in this case.[9]

$\neg 1$. **implies** $\neg 3$. For the remainder of the proof, assume that the transmitted word is $\mathbf{c}_1$ and there exists another codeword $\mathbf{c}_2$ such that $\Delta(\mathbf{c}_2, \mathbf{c}_1) = d - 1$. To see why property 3 is not true, let $\mathbf{y} = \mathbf{c}_2$. In this case, either the error detecting algorithm detects no error or it declares an error when $\mathbf{c}_2$ is the transmitted codeword and no error takes place during transmission.

[9] Note that this argument is just a generalization of the argument that $C_\oplus$ cannot correct 1 error.

$\neg 1$. **implies** $\neg 4$. We finally argue that property 4 does not hold. Let $\mathbf{y}$ be the received word in which the positions that are erased are exactly those where $\mathbf{c}_1$ and $\mathbf{c}_2$ differ. Thus, given $\mathbf{y}$ both $\mathbf{c}_1$ and $\mathbf{c}_2$ could have been the transmitted codeword and no algorithm for correcting (at most $d-1$) erasures can work in this case. ■

Proposition 1.4.1 implies that Question 1.1.1 can be reframed as

Question 1.4.1. *What is the largest rate R that a code with distance d can have?*

We have seen that the repetition code $C_{3,rep}$ has distance 3 and rate 1/3. A natural follow-up question (which is a special case of Question 1.4.1) is to ask

Question 1.4.2. *Can we have a code with distance 3 and rate $R > \frac{1}{3}$?*

1.5 Hamming Code

With the above question in mind, let us consider the so called *Hamming code*, which we will denote by C_H. Given a message $(x_1, x_2, x_3, x_4) \in \{0,1\}^4$, its corresponding codeword is given by

$$C_H(x_1, x_2, x_3, x_4) = (x_1, x_2, x_3, x_4, x_2 \oplus x_3 \oplus x_4, x_1 \oplus x_3 \oplus x_4, x_1 \oplus x_2 \oplus x_4).$$

It is easy to check that this code has the following parameters:

$$C_H : q = 2, k = 4, n = 7, R = 4/7.$$

We will show shortly that C_H has a distance of 3. We would like to point out that we could have picked the three parities differently. The reason we mention the three particular parities above is due to historical reasons. We leave it as an exercise to define alternate set of parities such that the resulting code still has a distance of 3: see Exercise 1.9.

Before we move on to determining the distance of C_H, we will need another definition.

Definition 1.5.1 (Hamming Weight)**.** Let $q \geq 2$. Given any vector $\mathbf{v} \in \{0,1,2,\ldots,q-1\}^n$, its Hamming weight, denoted by $wt(\mathbf{v})$ is the number of non-zero symbols in $\mathbf{v}$.

We now look at the distance of C_H.

Proposition 1.5.1. *C_H has distance 3.*

Proof. We will prove the claimed property by using two properties of C_H:

$$\min_{\mathbf{c} \in C_H, \mathbf{c} \neq \mathbf{0}} wt(\mathbf{c}) = 3, \tag{1.7}$$

and

$$\min_{\mathbf{c}\in C_H, \mathbf{c}\neq \mathbf{0}} wt(\mathbf{c}) = \min_{\mathbf{c}_1\neq \mathbf{c}_2 \in C_H} \Delta(\mathbf{c}_1, \mathbf{c}_2) \tag{1.8}$$

The proof of (1.7) follows from a case analysis on the Hamming weight of the message bits. Let us use $\mathbf{x} = (x_1, x_2, x_3, x_4)$ to denote the message vector.

- Case 0: If $wt(\mathbf{x}) = 0$, then $C_H(\mathbf{x}) = \mathbf{0}$, which means we do not have to consider this codeword.
- Case 1: If $wt(\mathbf{x}) = 1$ then at least two parity check bits in $(x_2 \oplus x_3 \oplus x_4, x_1 \oplus x_2 \oplus x_4, x_1 \oplus x_3 \oplus x_4)$ are 1. So in this case, $wt(C_H(\mathbf{x})) \geq 3$.
- Case 2: If $wt(\mathbf{x}) = 2$ then at least one parity check bit in $(x_2 \oplus x_3 \oplus x_4, x_1 \oplus x_2 \oplus x_4, x_1 \oplus x_3 \oplus x_4)$ is 1. So in this case, $wt(C_H(\mathbf{x})) \geq 3$.
- Case 3: If $wt(\mathbf{x}) \geq 3$ then obviously $wt(C_H(\mathbf{x})) \geq 3$.

Thus, we can conclude that $\min_{\mathbf{c}\in C_H, \mathbf{c}\neq \mathbf{0}} wt(\mathbf{c}) \geq 3$. Further, note that $wt(C_H(1,0,0,0)) = 3$, which along with the lower bound that we just obtained proves (1.7).

We now turn to the proof of (1.8). For the rest of the proof, let $\mathbf{x} = (x_1, x_2, x_3, x_4)$ and $\mathbf{y} = (y_1, y_2, y_3, y_4)$ denote the two distinct messages. Using associativity and commutativity of the $\oplus$ operator, we obtain that

$$C_H(\mathbf{x}) + C_H(\mathbf{y}) = C_H(\mathbf{x} + \mathbf{y}),$$

where the "+" operator is just the bit-wise $\oplus$ of the operand vectors. Further, it is easy to verify that for two vectors $\mathbf{u}, \mathbf{v} \in \{0,1\}^n$, $\Delta(\mathbf{u}, \mathbf{v}) = wt(\mathbf{u} + \mathbf{v})$ (see Exercise 1.10). Thus, we have

$$\begin{aligned}\min_{\mathbf{x}\neq\mathbf{y}\in\{0,1\}^4} \Delta(C_H(\mathbf{x}), C_H(\mathbf{y})) &= \min_{\mathbf{x}\neq\mathbf{y}\in\{0,1\}^4} wt(C_H(\mathbf{x}+\mathbf{y})) \\ &= \min_{\mathbf{x}\neq\mathbf{0}\in\{0,1\}^4} wt(C_H(\mathbf{x})),\end{aligned}$$

where the second equality follows from the observation that $\{\mathbf{x}+\mathbf{y} | \mathbf{x} \neq \mathbf{y} \in \{0,1\}^n\} = \{\mathbf{x} \in \{0,1\}^n | \mathbf{x} \neq \mathbf{0}\}$. Recall that $wt(C_H(\mathbf{x})) = 0$ if and only if $\mathbf{x} = \mathbf{0}$ and This completes the proof of (1.8). Combining (1.7) and (1.8), we conclude that C_H has a distance 3. □

The second part of the proof could also have been shown in the following manner. It can be verified easily that the Hamming code is the set $\{\mathbf{x} \cdot G_H | \mathbf{x} \in \{0,1\}^4\}$, where G_H is the following matrix (where we think $\mathbf{x}$ as a row vector).[10]

$$G_H = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

[10] Indeed $(x_1, x_2, x_3, x_4) \cdot G_H = (x_1, x_2, x_3, x_4, x_2 \oplus x_3 \oplus x_4, x_1 \oplus x_3 \oplus x_4, x_1 \oplus x_2 \oplus x_4)$, as desired.

In fact, any binary code (of dimension k and block length n) that is generated[11] by a $k \times n$ matrix is called a *binary linear code*. (Both $C_{\oplus}$ and $C_{3,rep}$ are binary linear codes: see Exercise 1.11.) This implies the following simple fact.

Lemma 1.5.2. *For any binary linear code C and any two messages $\mathbf{x}$ and $\mathbf{y}$, $C(\mathbf{x}) + C(\mathbf{y}) = C(\mathbf{x}+\mathbf{y})$.*

Proof. For any binary linear code, we have a generator matrix G. The following sequence of equalities (which follow from the distributivity and associativity properties of the boolean EXOR and AND operators) proves the lemma.

$$\begin{aligned} C(\mathbf{x}) + C(\mathbf{y}) &= \mathbf{x} \cdot G + \mathbf{y} \cdot G \\ &= (\mathbf{x}+\mathbf{y}) \cdot G \\ &= C(\mathbf{x}+\mathbf{y}) \end{aligned}$$

□

We stress that in the lemma above, $\mathbf{x}$ and $\mathbf{y}$ need *not* be distinct. Note that due to the fact that $b \oplus b = 0$ for every $b \in \{0,1\}$, $\mathbf{x}+\mathbf{x} = \mathbf{0}$, which along with the lemma above implies that $C(\mathbf{0}) = \mathbf{0}$.[12] We can infer the following result from the above lemma and the arguments used to prove (1.8) in the proof of Proposition 1.5.1.

Proposition 1.5.3. *For any binary linear code, minimum distance is equal to minimum Hamming weight of any non-zero codeword.*

Thus, we have seen that C_H has distance $d = 3$ and rate $R = \frac{4}{7}$ while $C_{3,rep}$ has distance $d = 3$ and rate $R = \frac{1}{3}$. Thus, the Hamming code is provably better than the repetition code (in terms of the tradeoff between rate and distance) and thus, answers Question 1.4.2 in the affirmative. The next natural question is

Question 1.5.1. *Can we have a distance 3 code with a rate higher than that of C_H?*

We will address this question in the next section.

1.6 Hamming Bound

Now we switch gears to present our first tradeoff between redundancy (in the form of dimension of a code) and its error-correction capability (in form of its distance). In particular, we will first prove a special case of the so called Hamming bound for a distance of 3.

We begin with another definition.

[11] That is, $C = \{\mathbf{x} \cdot G | \mathbf{x} \in \{0,1\}^k\}$, where addition is the $\oplus$ operation and multiplication is the AND operation.

[12] This of course should not be surprising as for any matrix G, we have $\mathbf{0} \cdot G = \mathbf{0}$.

Definition 1.6.1 (Hamming Ball)**.** For any vector $\mathbf{x} \in [q]^n$,

$$B(\mathbf{x}, e) = \{\mathbf{y} \in [q]^n | \Delta(\mathbf{x}, \mathbf{y}) \leq e\}.$$

Next we prove an upper bound on the dimension of *every* code with distance 3.

Theorem 1.6.1 (Hamming bound for $d = 3$)**.** *Every binary code with block length n, dimension k, distance $d = 3$ satisfies*

$$k \leq n - \log_2(n+1).$$

Proof. Given any two codewords, $\mathbf{c}_1 \neq \mathbf{c}_2 \in C$, the following is true (as C has distance[13] 3):

$$B(\mathbf{c}_1, 1) \cap B(\mathbf{c}_2, 1) = \emptyset. \tag{1.9}$$

See Figure 1.4 for an illustration. Note that for all $\mathbf{x} \in \{0,1\}^n$ (see Exercise 1.14),

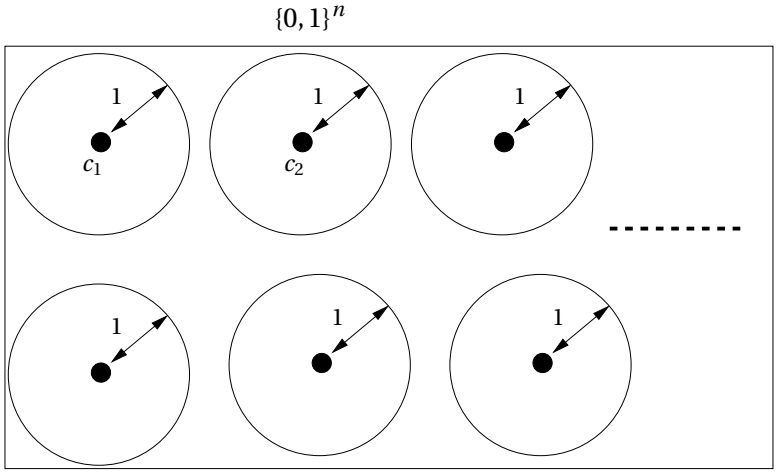


Figure 1.4: Hamming balls of radius 1 are disjoint. The figure is technically not correct: the balls above are actually balls in the Euclidean space, which is easier to visualize than the Hamming space.

$$|B(\mathbf{x}, 1)| = n + 1. \tag{1.10}$$

Now consider the union of all Hamming balls centered around some codeword. Obviously their union is a subset of $\{0,1\}^n$. In other words,

$$\left| \bigcup_{\mathbf{c} \in C} B(\mathbf{c}, 1) \right| \leq 2^n. \tag{1.11}$$

[13] Assume that $\mathbf{y} \in B(\mathbf{c}_1, 1) \cap B(\mathbf{c}_2, 1)$, that is $\Delta(\mathbf{y}, \mathbf{c}_1) \leq 1$ and $\Delta(\mathbf{y}, \mathbf{c}_2) \leq 1$. Thus, by the triangle inequality $\Delta(\mathbf{c}_1, \mathbf{c}_2) \leq 2 < 3$, which is a contradiction.

As (1.9) holds for every pair of distinct codewords,

$$\begin{aligned}\left|\bigcup_{\mathbf{c}\in C} B(\mathbf{c},1)\right| &= \sum_{\mathbf{c}\in C} |B(\mathbf{c},1)| \\ &= 2^k \cdot (n+1), \end{aligned} \tag{1.12}$$

where (1.12) follows from (1.10) and the fact that C has dimension k. Combining (1.12) and (1.11) and taking $\log_2$ of both sides we will get the desired bound:

$$k \leq n - \log_2(n+1).$$

□

Thus, Theorem 1.6.1 shows that for $n = 7$, C_H has the largest possible dimension for any binary code of block length 7 and distance 3 (as for $n = 7$, $n - \log_2(n+1) = 4$). In particular, it also answers Question 1.5.1 for $n = 7$ in the negative. Next, will present the general form of Hamming bound.

1.7 Generalized Hamming Bound

We start with a new notation.

Definition 1.7.1. A code $C \subseteq \Sigma^n$ with dimension k and distance d will be called a $(n,k,d)_\Sigma$ code. We will also refer it to as a $(n,k,d)_{|\Sigma|}$ code.

We now proceed to generalize Theorem 1.6.1 to any distance d.

Theorem 1.7.1 (Hamming Bound for any d)**.** *For every $(n,k,d)_q$ code*

$$k \leq n - \log_q \left(\sum_{i=0}^{\left\lfloor \frac{(d-1)}{2} \right\rfloor} \binom{n}{i} (q-1)^i \right).$$

Proof. The proof is a straightforward generalization of the proof of Theorem 1.6.1. For notational convenience, let $e = \left\lfloor \frac{(d-1)}{2} \right\rfloor$. Given any two codewords, $\mathbf{c}_1 \neq \mathbf{c}_2 \in C$, the following is true (as C has distance[14] d):

$$B(\mathbf{c}_1, e) \cap B(\mathbf{c}_2, e) = \emptyset. \tag{1.13}$$

We claim that for all $\mathbf{x} \in [q]^n$,

$$|B(\mathbf{x}, e)| = \sum_{i=0}^{e} \binom{n}{i} (q-1)^i. \tag{1.14}$$

[14] Assume that $\mathbf{y} \in B(\mathbf{c}_1, e) \cap B(\mathbf{c}_2, e)$, that is $\Delta(\mathbf{y}, \mathbf{c}_1) \leq e$ and $\Delta(\mathbf{y}, \mathbf{c}_2) \leq e$. Thus, by the triangle inequality, $\Delta(\mathbf{c}_1, \mathbf{c}_2) \leq 2e \leq d-1$, which is a contradiction.

Indeed any vector in $B(\mathbf{x}, e)$ must differ from $\mathbf{x}$ in exactly $0 \leq i \leq e$ positions. In the summation $\binom{n}{i}$ is the number of ways of choosing the differing i positions and in each such position, a vector can differ from $\mathbf{x}$ in $q-1$ ways.

Now consider the union of all Hamming balls centered around some codeword. Obviously their union is a subset of $[q]^n$. In other words,

$$\left| \bigcup_{\mathbf{c} \in C} B(\mathbf{c}, e) \right| \leq q^n. \tag{1.15}$$

As (1.13) holds for every pair of distinct codewords,

$$\begin{aligned} \left| \bigcup_{\mathbf{c} \in C} B(\mathbf{c}, e) \right| &= \sum_{\mathbf{c} \in C} |B(\mathbf{c}, e)| \\ &= q^k \sum_{i=0}^{e} \binom{n}{i} (q-1)^i, \end{aligned} \tag{1.16}$$

where (1.16) follows from (1.14) and the fact that C has dimension k. Combining (1.16) and (1.15) and taking $\log_q$ of both sides we will get the desired bound:

$$k \leq n - \log_q \left(\sum_{i=0}^{e} \binom{n}{i} (q-1)^i \right).$$

□

The Hamming bound leads to the following definition:

Definition 1.7.2. Codes that meet Hamming bound are called *perfect codes*.

Intuitively, a perfect code leads to the following perfect "packing": if one constructs Hamming balls of radius $\left\lfloor \frac{d-1}{2} \right\rfloor$ around all the codewords, then we would cover the entire ambient space, i.e. every possible vector will lie in one of these Hamming balls.

One example of perfect code is the $(7,4,3)_2$ Hamming code that we have seen in this chapter (so is the family of general Hamming codes that we will see in the next chapter). A natural question to ask is if

Question 1.7.1. *Other than the Hamming codes, are there any other perfect (binary) codes?*

We will see the answer shortly.

1.8 Exercises

Exercise 1.1. Show that any t-error correcting code is also t-error detecting but not necessarily the other way around.

Exercise 1.2. Prove Proposition 1.3.1.

Exercise 1.3. Show that for every integer n, there is no code with block length n that can handle arbitrary number of errors.

Exercise 1.4. Prove Proposition 1.3.2.

Exercise 1.5. A distance function on Σ^n (i.e. $d : \Sigma^n \times \Sigma^n \to \mathbb{R}$) is called a *metric* if the following conditions are satisfied for every $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \Sigma^n$:

1. $d(\mathbf{x}, \mathbf{y}) \geq 0$.
2. $d(\mathbf{x}, \mathbf{y}) = 0$ if and only if $\mathbf{x} = \mathbf{y}$.
3. $d(\mathbf{x}, \mathbf{y}) = d(\mathbf{y}, \mathbf{x})$.
4. $d(\mathbf{x}, \mathbf{z}) \leq d(\mathbf{x}, \mathbf{y}) + d(\mathbf{y}, \mathbf{z})$. (This property is called the *triangle inequality*.)

Prove that the Hamming distance is a metric.

Exercise 1.6. Let C be a code with distance d for even d. Then argue that C can correct up to $d/2 - 1$ many errors but cannot correct $d/2$ errors. Using this or otherwise, argue that if a code C is t-error correctable then it either has a distance of $2t + 1$ or $2t + 2$.

Exercise 1.7. In this exercise, we will see that one can convert arbitrary codes into code with slightly different parameters:

1. Let C be an $(n, k, d)_2$ code with d odd. Then it can be converted into an $(n + 1, k, d + 1)_2$ code.
2. Let C be an $(n, k, d)_\Sigma$ code. Then it can be converted into an $(n - 1, k, d - 1)_\Sigma$ code.

Note: Other than the parameters of the code C, you should not assume anything else about the code. Also your conversion should work for every $n, k, d \geq 1$.

Exercise 1.8. In this problem we will consider a noise model that has both errors and erasures. In particular, let C be an $(n, k, d)_\Sigma$ code. As usual a codeword $\mathbf{c} \in C$ is transmitted over the channel and the received word is a vector $\mathbf{y} \in (\Sigma \cup \{?\})^n$, where as before a ? denotes an erasure. We will use s to denote the number of erasures in $\mathbf{y}$ and e to denote the number of (non-erasure) errors that occurred during transmission. To decode such a vector means to output a codeword $\mathbf{c} \in C$ such that the number of positions where $\mathbf{c}$ disagree with $\mathbf{y}$ in the $n - s$ non-erased positions is at most e. For the rest of the problem assume that

$$2e + s < d. \tag{1.17}$$

1. Argue that the output of the decoder for any C under (1.17) is unique.
2. Let C be a binary code (but not necessarily linear). Assume that there exists a decoder D that can correct from $< d/2$ many errors in $T(n)$ time. Then under (1.17) one can perform decoding in time $O(T(n))$.

Exercise 1.9. Define codes other than C_H with $k = 4, n = 7$ and $d = 3$.

Hint: Refer to the proof of Proposition 1.5.1 to figure out the properties needed from the three parities.

Exercise 1.10. Prove that for any $\mathbf{u}, \mathbf{v} \in \{0,1\}^n$, $\Delta(\mathbf{u}, \mathbf{v}) = wt(\mathbf{u} + \mathbf{v})$.

Exercise 1.11. Argue that $C_\oplus$ and $C_{3,rep}$ are binary linear codes.

Exercise 1.12. Let G be a generator matrix of an $(n,k,d)_2$ binary linear code. Then G has at least kd ones in it.

Exercise 1.13. Argue that in any binary linear code, either all all codewords begin with a 0 of exactly half of the codewords begin with a 0.

Exercise 1.14. Prove (1.10).

Exercise 1.15. Show that there is no binary code with block length 4 that achieves the Hamming bound.

Exercise 1.16. $^{(*)}$ There are n people in a room, each of whom is given a black/white hat chosen uniformly at random (and independent of the choices of all other people). Each person can see the hat color of all other people, but not their own. Each person is asked if (s)he wishes to guess their own hat color. They can either guess, or abstain. Each person makes their choice without knowledge of what the other people are doing. They either win collectively, or lose collectively. They win if all the people who don't abstain guess their hat color correctly *and* at least one person does not abstain. They lose if all people abstain, or if some person guesses their color incorrectly. Your goal below is to come up with a strategy that will allow the n people to win with pretty high probability. We begin with a simple warmup:

(a) Argue that the n people can win with probability at least $\frac{1}{2}$.

Next we will see how one can really bump up the probability of success with some careful modeling, and some knowledge of Hamming codes. (Below are assuming knowledge of the general Hamming code (see Section 2.4). If you do not want to skip ahead, you can assume that $n = 7$ in the last part of this problem.

(b) Lets say that a directed graph G is a subgraph of the n-dimensional hypercube if its vertex set is $\{0,1\}^n$ and if $u \to v$ is an edge in G, then u and v differ in at most one coordinate. Let $K(G)$ be the number of vertices of G with in-degree at least one, and out-degree zero. Show that the probability of winning the hat problem equals the maximum, over directed subgraphs G of the n-dimensional hypercube, of $K(G)/2^n$.

(c) Using the fact that the out-degree of any vertex is at most n, show that $K(G)/2^n$ is at most $\frac{n}{n+1}$ for any directed subgraph G of the n-dimensional hypercube.

(d) Show that if $n = 2^r - 1$, then there exists a directed subgraph G of the n-dimensional hypercube with $K(G)/2^n = \frac{n}{n+1}$.

Hint: This is where the Hamming code comes in.

1.9 Bibliographic Notes

Coding theory owes its original to two remarkable papers: one by Shannon and the other by Hamming [39] both of which were published within a couple of years of each other. Shannon's paper defined the BSC_p channel (among others) and defined codes in terms of its encoding function. Shannon's paper also explicitly defined the decoding function. Hamming's work defined the notion of codes as in Definition 1.2.1 as well as the notion of Hamming distance. Both the Hamming bound and the Hamming code are (not surprisingly) due to Hamming. The specific definition of Hamming code that we used in this book was the one proposed by Hamming and is also mentioned in Shannon's paper (even though Shannon's paper pre-dates Hamming's).

The notion of erasures was defined by Elias.

One hybrid model to account for the fact that in real life the noise channel is somewhere in between the extremes of the channels proposed by Hamming and Shannon is the *Arbitrary Varying Channel* (the reader is referred to the survey by Lapidoth and Narayan [50]).

Chapter 2

A Look at Some Nicely Behaved Codes: Linear Codes

Let us now pause for a bit and think about how we can represent a code. In general, a code $C : [q]^k \longrightarrow [q]^n$ can be stored using nq^k symbols from $[q]$ (n symbols for each of the q^k codewords) or $nq^k \log q$ bits. For constant rate codes, this is exponential space, which is prohibitive even for modest values of k like $k = 100$. A natural question is whether we can do better. Intuitively, the code must have some extra structure that would facilitate a succinct representation of the code. We will now look at a class of codes called *linear codes* that have more structure than general codes which leads to some other nice properties. We have already seen binary linear codes in Section 1.5, that is, $C \subseteq \{0,1\}^n$ is linear code if for all $\mathbf{c}_1, \mathbf{c}_2 \in C$, $\mathbf{c}_1 + \mathbf{c}_2 \in C$, where the "+" denotes bit-wise EXOR.

Definition 2.0.1 (Linear Codes)**.** Let q be a prime power (i.e. $q = p^s$ for some prime p and integer $s \geq 1$). $C \subseteq \{0,1,...,q-1\}^n$ is a *linear code* if it is a *linear subspace* of $\{0,1,...,q-1\}^n$. If C has dimension k and distance d then it will be referred to as an $[n,k,d]_q$ or just an $[n,k]_q$ code.

Of course the above definition is not complete because we have not defined a linear subspace yet. We do that next.

2.1 Finite Fields

To define linear subspaces, we will need to work with (finite) fields. At a high level we need finite fields as when we talk about codes, we deal with finite symbols/numbers and we want to endow these symbols with the same math that makes arithmetic over reals work. Finite fields accomplish this precise task. We begin with a quick overview of fields.

Informally speaking, a field is a set of elements on which one can do addition, subtraction, multiplication and division and still stay in the set. More formally,

Definition 2.1.1. A field $\mathbb{F}$ is given by a triple $(S,+,\cdot)$, where S is the set of elements containing special elements 0 and 1 and $+,\cdot$ are functions $\mathbb{F} \times \mathbb{F} \to \mathbb{F}$ with the following properties:

- Closure: For every $a, b \in S$, we have both $a + b \in S$ and $a \cdot b \in S$.
- Associativity: $+$ and $\cdot$ are associative, that is, for every $a, b, c \in S$, $a + (b + c) = (a + b) + c$ and $a \cdot (b \cdot c) = (a \cdot b) \cdot c$.
- Commutativity: $+$ and $\cdot$ are commutative, that is, for every $a, b \in S$, $a + b = b + a$ and $a \cdot b = b \cdot a$.
- Distributivity: $\cdot$ distributes over $+$, that is for every $a, b, c \in S$, $a \cdot (b + c) = a \cdot b + a \cdot c$.
- Identities: For every $a \in S$, $a + 0 = a$ and $a \cdot 1 = a$.
- Inverses: For every $a \in S$, there exists its unique *additive inverse* $-a$ such that $a + (-a) = 0$. Also for every $a \in S \setminus \{0\}$, there exists its unique *multiplicative inverse* a^{-1} such that $a \cdot a^{-1} = 1$.

With the usual semantics for $+$ and $\cdot$, $\mathbb{R}$ (set of real number) is a field but $\mathbb{Z}$ (set of integers) is not a field as division of two integers can give rise to a rational number (the set of rational numbers itself is a field though– see Exercise 2.1). In this course, we will exclusively deal with *finite fields*. As the name suggests these are fields with a finite size set of elements. (We will overload notation and denote the size of a field $|\mathbb{F}| = |S|$.) The following is a well known result.

Theorem 2.1.1 (Size of Finite Fields)**.** *The size of any finite field is p^s for prime p and integer $s \geq 1$.*

One example of finite fields that we have seen is the field of two elements $\{0, 1\}$, which we will denote by $\mathbb{F}_2$ (we have seen this field in the context of binary linear codes). For $\mathbb{F}_2$, addition is the XOR operation, while multiplication is the AND operation. The additive inverse of an element in $\mathbb{F}_2$ is the number itself while the multiplicative inverse of 1 is 1 itself.

Let p be a prime number. Then the integers modulo p form a field, denoted by $\mathbb{F}_p$ (and also by $\mathbb{Z}_p$), where the addition and multiplication are carried out $\mod p$. For example, consider $\mathbb{F}_7$, where the elements are $\{0, 1, 2, 3, 4, 5, 6\}$. So we have $4 + 3 \mod 7 = 0$ and $4 \cdot 4 \mod 7 = 2$. Further, the additive inverse of 4 is 3 as $3 + 4 \mod 7 = 0$ and the multiplicative inverse of 4 is 2 as $4 \cdot 2 \mod 7 = 1$.

More formally, we prove the following result.

Lemma 2.1.2. *Let p be a prime. Then $\mathbb{F}_p = (\{0, 1, \ldots, p-1\}, +_p, \cdot_p)$ is a field, where $+_p$ and $\cdot_p$ are addition and multiplication* $\mod p$.

Proof. The properties of associativity, commutativity, distributivity and identities hold for integers and hence, they hold for $\mathbb{F}_p$. The closure property follows since both the "addition" and "multiplication" are done $\mod p$, which implies that for any $a, b \in \{0, \ldots, p-1\}$, $a +_p b, a \cdot_p b \in \{0, \ldots, p-1\}$. Thus, to complete the proof, we need to prove the existence of unique additive and multiplicative inverses.

Fix an arbitrary $a \in \{0, \ldots, p-1\}$. Then we claim that its additive inverse is $p - a \mod p$. It is easy to check that $a + p - a = 0 \mod p$. Next we argue that this is the unique additive inverse.

To see this note that the sequence $a, a+1, a+2, \ldots, a+p-1$ are p consecutive numbers and thus, exactly one of them is a multiple of p, which happens for $b = p - a \mod p$, as desired.

Now fix an $a \in \{1, \ldots, p-1\}$. Next we argue for the existence of a unique multiplicative universe a^{-1}. Consider the set of numbers $\{a \cdot_p b\}_{b \in \{1,\ldots,p-1\}}$. We claim that all these numbers are unique. To see this, note that if this is not the case, then there exist $b_1 \neq b_2 \in \{0, 1, \ldots, p-1\}$ such that $a \cdot b_1 = a \cdot b_2 \mod p$, which in turn implies that $a \cdot (b_1 - b_2) = 0 \mod p$. Since a and $b_1 - b_2$ are non-zero numbers, this implies that p divides $a \cdot (b_1 - b_2)$. Further, since a and $|b_1 - b_2|$ are both at most $p-1$, this implies that factors of a and $(b_1 - b_2) \mod p$ when multiplied together results in p, which is a contradiction since p is prime. Thus, this implies that there exists a unique element b such that $a \cdot b = 1 \mod p$ and thus, b is the required a^{-1}. □

One might think that there could be different fields with the same number of elements. However, this is not the case:

Theorem 2.1.3. *For every prime power q there is a* unique *finite field with q elements (up to isomorphism[1]).*

Thus, we are justified in just using $\mathbb{F}_q$ to denote a finite field on q elements.

2.2 Linear Subspaces

We are finally ready to define the notion of linear subspace.

Definition 2.2.1 (Linear Subspace)**.** $S \subseteq {\mathbb{F}_q}^n$ is a linear subspace if the following properties hold:

1. For every $\mathbf{x}, \mathbf{y} \in S$, $\mathbf{x} + \mathbf{y} \in S$, where the addition is vector addition over $\mathbb{F}_q$ (that is, do addition component wise over $\mathbb{F}_q$).
2. For every $a \in \mathbb{F}_q$ and $\mathbf{x} \in S$, $a \cdot \mathbf{x} \in S$, where the multiplication is over $\mathbb{F}_q$.

Here is a (trivial) example of a linear subspace of $\mathbb{F}_5^3$:

$$S_1 = \{(0,0,0), (1,1,1), (2,2,2), (3,3,3), (4,4,4)\}. \tag{2.1}$$

Note that for example $(1,1,1) + (3,3,3) = (4,4,4) \in S_1$ and $2 \cdot (4,4,4) = (3,3,3) \in S_1$ as required by the definition. Here is another somewhat less trivial example of a linear subspace over $\mathbb{F}_3^3$:

$$S_2 = \{(0,0,0), (1,0,1), (2,0,2), (0,1,1), (0,2,2), (1,1,2), (1,2,0), (2,1,0), (2,2,1). \tag{2.2}$$

Note that $(1,0,1) + (0,2,2) = (1,2,0) \in S_2$ and $2 \cdot (2,0,2) = (1,0,1) \in S_2$ as required.

Remark 2.2.1. Note that the second property implies that $\mathbf{0}$ is contained in every linear subspace. Further for any subspace over $\mathbb{F}_2$, the second property is redundant: see Exercise 2.4.

Before we state some properties of linear subspaces, we state some relevant definitions.

[1] An isomorphism $\phi : S \to S'$ is a map (such that $\mathbb{F} = (S, +, \cdot)$ and $\mathbb{F}' = (S', \oplus, \circ)$ are fields) where for every $a_1, a_2 \in S$, we have $\phi(a_1 + a_2) = \phi(a_1) \oplus \phi(a_2)$ and $\phi(a_1 \cdot a_2) = \phi(a_1) \circ \phi(a_2)$.

Definition 2.2.2 (Span)**.** Given a set $B = \{\mathbf{v}_1, \ldots, \mathbf{v}_\ell\}$. The *span* of B is the set of vectors

$$\left\{ \sum_{i=1}^{\ell} a_i \cdot \mathbf{v}_i | a_i \in \mathbb{F}_q \text{ for every } i \in [\ell] \right\}.$$

Definition 2.2.3 (Linear independence of vectors)**.** We say that $\mathbf{v}_1, \mathbf{v}_2, \ldots \mathbf{v}_k$ are *linearly independent* if for every $1 \leq i \leq k$ and for every $k-1$-tuple $(a_1, a_2, \ldots, a_{i-1}, a_{i+1}, \ldots, a_k) \in \mathbb{F}_q^{k-1}$,

$$\mathbf{v}_i \neq a_1\mathbf{v}_1 + \ldots + a_{i-1}\mathbf{v}_{i-1} + a_{i+1}\mathbf{v}_{i+1} + \ldots + a_k\mathbf{v}_k.$$

In other words, $\mathbf{v}_i$ is not in the span of the set $\{\mathbf{v}_1, \ldots, \mathbf{v}_{i-1}, \mathbf{v}_{i+1}, \ldots, \mathbf{v}_n\}$.

For example the vectors $(1,0,1), (1,1,1) \in S_2$ are linearly independent.

Definition 2.2.4 (Rank of a matrix)**.** The *rank* of matrix in $\mathbb{F}_q^{k \times k}$ is the maximum number of linearly independent rows (or columns). A matrix in $\mathbb{F}_q^{k \times n}$ with rank $\min(k, n)$ is said to have *full* rank.

One can define the row (column) rank of a matrix as the maximum number of linearly independent rows (columns). However, it is a well-known theorem that the row rank of a matrix is the same as its column rank. For example, the matrix below over $\mathbb{F}_3$ has full rank (see Exercise 2.5):

$$G_2 = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}. \tag{2.3}$$

Any linear subspace satisfies the following properties (the full proof can be found in any standard linear algebra textbook).

Theorem 2.2.1. *If $S \subseteq {\mathbb{F}_q}^n$ is a linear subspace then*

1. *$|S| = q^k$ for some $k \geq 0$. The parameter k is called the* dimension *of S.*
2. *There exists $\mathbf{v}_1, ..., \mathbf{v}_k \in S$ called* basis elements *(which need not be unique) such that every $\mathbf{x} \in S$ can be expressed as $\mathbf{x} = a_1\mathbf{v}_1 + a_2\mathbf{v}_2 + ... + a_n\mathbf{v}_n$ where $a_i \in \mathbb{F}_q$ for $1 \leq i \leq k$. In other words, there exists a full rank $k \times n$ matrix G (also known as a* generator matrix*) with entries from $\mathbb{F}_q$ such that every $\mathbf{x} \in S$, $\mathbf{x} = (a_1, a_2, ...a_k) \cdot G$ where*

$$G = \begin{pmatrix} \longleftarrow \mathbf{v}_1 \longrightarrow \\ \longleftarrow \mathbf{v}_2 \longrightarrow \\ \vdots \\ \longleftarrow \mathbf{v}_k \longrightarrow \end{pmatrix}.$$

3. *There exists a full rank $(n-k) \times n$ matrix H (called a* parity check matrix*) such that for every $\mathbf{x} \in S$, $H\mathbf{x}^T = \mathbf{0}$.*
4. *G and H are* orthogonal*, that is, $G \cdot H^T = \mathbf{0}$.*

Proof Sketch.

Property 1. We begin with the proof of the first property. For the sake of contradiction, let us assume that $q^k < |S| < q^{k+1}$, for some $k \geq 0$. Iteratively, we will construct a set of linearly independent vectors $B \subseteq S$ such that $|B| \geq k+1$. Note that by the definition of a linear subspace the span of B should be contained in S. However, this is a contradiction as the size of the span of B is at least[2] $q^{k+1} > |S|$.

To complete the proof, we show how to construct the set B in a greedy fashion. In the first step pick $\mathbf{v}_1$ to be any non-zero vector in S and set $B \leftarrow \{\mathbf{v}_1\}$ (we can find such a vector as $|S| > q^k \geq 1$). Now say after the step t (for some $t \leq k$), $|B| = t$. Now the size of the span of the current B is $q^t \leq q^k < |S|$. Thus there exists a vector $\mathbf{v}_{t+1} \in S \setminus B$ that is linearly independent of vectors in B. Set $B \leftarrow B \cup \{\mathbf{v}_{t+1}\}$. Thus, we can continue building B till $|B| = k+1$, as desired.

Property 2. We first note that we can pick $B = \{\mathbf{v}_1, \ldots, \mathbf{v}_k\}$ to be any set of k linearly independent vectors– this just follows from the argument above for **Property** 1.1. This is because the span of B is contained in S. However, since $|S| = q^k$ and the span of B has q^k vectors, the two have to be the same.

Property 3. Property 3 above follows from another fact that every linear subspace S has a null space $N \subseteq \mathbb{F}_q^n$ such that for every $\mathbf{x} \in S$ and $\mathbf{y} \in N$, $\langle \mathbf{x}, \mathbf{y} \rangle = 0$. Further, it is known that N itself is a linear subspace of dimension $n-k$. (The claim that N is also a linear subspace follows from the following two facts: for every $\mathbf{x}, \mathbf{y}, \mathbf{z} \in \mathbb{F}_q^n$, (i) $\langle \mathbf{x}, \mathbf{y}+\mathbf{z} \rangle = \langle \mathbf{x}, \mathbf{y} \rangle + \langle \mathbf{x}, \mathbf{z} \rangle$ and (ii) for any $a \in \mathbb{F}_q$, $\langle \mathbf{x}, a\mathbf{y} \rangle = a \cdot \langle \mathbf{x}, \mathbf{y} \rangle$.) In other words, there exists a generator matrix H for it. This matrix H is called the parity check matrix of S.

Property 4. See Exercise 2.8. □

As examples, the linear subspace S_1 in (2.1) has as one of its generator matrices

$$G_1 = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix}$$

and as one of its parity check matrices

$$H_1 = \begin{pmatrix} 1 & 2 & 2 \\ 2 & 2 & 1 \end{pmatrix}.$$

Further, the linear subspace S_2 in (2.2) has G_2 as one of its generator matrices and has the following as one of its parity check matrices

$$H_2 = \begin{pmatrix} 1 & 1 & 2 \end{pmatrix}.$$

Finally, we state another property of linear subspaces that is useful.

[2] See Exercise 2.7.

Lemma 2.2.2. *Given matrix G of dimension $k \times n$ that is a generator matrix of subspace S_1 and matrix H of dimension $(n-k) \times n$ that is a parity check matrix of subspace S_2 such that $GH^T = \mathbf{0}$, then $S_1 = S_2$.*

Proof. We first prove that $S_1 \subseteq S_2$. Given any $\mathbf{c} \in S_1$, there exists $\mathbf{x} \in \mathbb{F}_q^k$ such that $\mathbf{c} = \mathbf{x}G$. Then,

$$\mathbf{c}H^T = \mathbf{x}GH^T = \mathbf{0},$$

which implies that $\mathbf{c} \in S_2$, as desired.

To complete the proof note that as H has full rank, its null space (or S_2) has dimension $n-(n-k) = k$ (this follows from a well known fact from linear algebra). Now as G has full rank, the dimension of S_1 is also k. Thus, as $S_1 \subseteq S_2$, it has to be the case that $S_1 = S_2$.[3] □

2.3 Properties of Linear Codes

The above theorem gives two alternate characterizations of an $[n,k]_q$ linear code C:

- C is generated by its $k \times n$ generator matrix G. As an example that we have already seen, the $[7,4,3]_2$ Hamming code has the following generator matrix:

$$G = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}$$

- C is also characterized by an $(n-k) \times n$ parity check matrix H. We claim that the following matrix is a parity check matrix of the $[7,4,3]_2$ Hamming code:

$$H = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

  Indeed, it can be easily verified that $G \cdot H^T = \mathbf{0}$. Then Lemma 2.2.2 proves that H is indeed a parity check matrix of the $[7,4,3]_2$ Hamming code.

We now look at some consequences of the above characterizations of an $[n,k]_q$ linear code C. We started this chapter with a quest for succinct representation of a code. Note that both the generator matrix and the parity check matrix can be represented using $O(n^2)$ symbols from $\mathbb{F}_q$ (which is much smaller than the exponential representation of a general code). More precisely (see Exercise 2.10),

Proposition 2.3.1. *Any $[n,k]_q$ linear code can be represented with $\min(nk, n(n-k))$ symbols from $\mathbb{F}_q$.*

[3] If not, $S_1 \subset S_2$ which implies that that $|S_2| \geq |S_1| + 1$. The latter is not possible if both S_1 and S_2 have the same dimension.

There is an encoding algorithm for C that runs in $O(n^2)$ (in particular $O(kn)$) time– given a message $\mathbf{m} \in \mathbb{F}_q^k$, the corresponding codeword $C(\mathbf{m}) = \mathbf{m} \cdot G$, where G is the generator matrix of C. (See Exercise 2.11.)

Proposition 2.3.2. *For any $[n,k]_q$ linear code, given its generator matrix, encoding can be done with $O(nk)$ operations over $\mathbb{F}_q$.*

There is an error-detecting algorithm for C that runs in $O(n^2)$. This is a big improvement over the naive brute force exponential time algorithm (that goes through all possible codewords $\mathbf{c} \in C$ and checks if $\mathbf{y} = \mathbf{c}$). (See Exercise 2.12.)

Proposition 2.3.3. *For any $[n,k]_q$ linear code, given its parity check matrix, error detection can be performed in $O(n(n-k))$ operations over $\mathbb{F}_q$.*

Next, we look at some alternate characterizations of the distance of a linear code.

2.3.1 On the Distance of a Linear Code

We start with the following property, which we have seen for the special case of binary linear codes (Proposition 1.5.3).

Proposition 2.3.4. *For a $[n,k,d]_q$ code C,*

$$d = \min_{\substack{\mathbf{c} \in C, \\ \mathbf{c} \neq \mathbf{0}}} wt(\mathbf{c}).$$

Proof. To show that d is the same as the minimum weight we show that d is no more than the minimum weight and d is no less than the minimum weight.

First, we show that d is no more than the minimum weight. We can see this by considering $\Delta(\mathbf{0}, \mathbf{c}')$ where $\mathbf{c}'$ is the non-zero codeword in C with minimum weight; its distance from $\mathbf{0}$ is equal to its weight. Thus, we have $d \leq wt(\mathbf{c}')$, as desired.

Now, to show that d is no less than the minimum weight, consider $\mathbf{c_1} \neq \mathbf{c_2} \in C$ such that $\Delta(\mathbf{c}_1, \mathbf{c}_2) = d$. Note that $\mathbf{c}_1 - \mathbf{c_2} \in C$ (this is because $-\mathbf{c_2} = -1 \cdot \mathbf{c}_2 \in C$, where -1 is the additive inverse of 1 in $\mathbb{F}_q$ and $\mathbf{c}_1 - \mathbf{c}_2 = \mathbf{c}_1 + (-\mathbf{c}_2)$, which by the definition of linear codes is in C). Now note that $wt(\mathbf{c_1} - \mathbf{c_2}) = \Delta(\mathbf{c_1}, \mathbf{c_2}) = d$, since the non-zero symbols in $\mathbf{c}_1 - \mathbf{c}_2$ occur exactly in the positions where the two codewords differ. Further, since $\mathbf{c}_1 \neq \mathbf{c}_2$, $\mathbf{c}_1 - \mathbf{c}_2 \neq \mathbf{0}$, which implies that the minimum Hamming weight of any non-zero codeword in C is at most d. □

Next, we look at another property implied by the parity check matrix of a linear code.

Proposition 2.3.5. *For any $[n,k,d]_q$ code C with parity check matrix H, d is the minimum number of linearly dependent columns in H.*

Proof. By Proposition 2.3.4, we need to show that the minimum weight of a non-zero codeword in C is the minimum number of linearly dependent columns. Let t be the minimum number of linearly dependent columns in H. To prove the claim we will show that $t \leq d$ and $t \geq d$.

For the first direction, Let $\mathbf{c} \neq \mathbf{0} \in C$ be a codeword with $wt(\mathbf{c}) = d$. Now note that, by the definition of the parity check matrix, $H \cdot \mathbf{c}^T = \mathbf{0}$. Working through the matrix multiplication, this gives us that $\sum_{i=1}^{n} c_i H^i$, where

$$H = \begin{pmatrix} \uparrow & \uparrow & & \uparrow & & \uparrow \\ H^1 & H^2 & \cdots & H^i & \cdots & H^n \\ \downarrow & \downarrow & & \downarrow & & \downarrow \end{pmatrix}$$

and $\mathbf{c} = (c_1, \ldots, c_n)$. Note that we can skip multiplication for those columns for which the corresponding bit c_i is zero, so for this to be zero, those H^i with $c_i \neq 0$ are linearly dependent. This means that $d \geq t$, as the columns corresponding to non-zero entries in $\mathbf{c}$ are one instance of linearly dependent columns.

For the other direction, consider the minimum set of columns from H, $H^{i_1}, H^{i_2}, \ldots, H^{i_t}$ that are linearly dependent. This implies that there exists non-zero elements $c'_{i_1}, \ldots, c'_{i_t} \in \mathbb{F}_q$ such that $c'_{i_i} H^{i_1} + \ldots + c'_{i_t} H^{i_t} = \mathbf{0}$. (Note that all the c'_{i_j} are non-zero as no set of less than t columns are linearly dependent.) Now extend $c'_{i_1}, \ldots, c'_{i_t}$ to the vector $\mathbf{c}'$ such that $c'_j = 0$ for $j \notin \{i_1, \ldots, i_t\}$. Note that $\mathbf{c}' \in C$ and thus, $d \leq wt(\mathbf{c}') = t$ (where recall t is the minimum number of linearly independent columns in H). □

2.4 Hamming Codes

We now change gears and look at the general family of linear codes, which were discovered by Hamming. So far we have seen the $[7,4,3]_2$ Hamming code (in Section 1.5). In fact for any $r \geq 2$, there is a $[2^r - 1, 2^r - r - 1, 3]_2$ Hamming code. Thus in Section 1.5, we have seen this code for $r = 3$.

Consider the $r \times (2^r - 1)$ matrix $\mathbf{H}_r$ over $\mathbb{F}_2$, where the ith column $\mathbf{H}_r^i$, $1 \leq i \leq 2^r - 1$, is the binary representation of i (note that such a representation is a vector in $\{0,1\}^r$). For example, for the case we have seen ($r = 3$),

$$\mathbf{H}_3 = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}.$$

Note that by its definition, the code that has $\mathbf{H}_r$ as its parity check matrix has block length $2^r - 1$ and dimension $2^r - r - 1$. This leads to the formal definition of the general Hamming code.

Definition 2.4.1. The $[2^r - 1, 2^r - r - 1]_2$ Hamming code, denoted by $C_{H,r}$ has parity check matrix $\mathbf{H}_r$.

In other words, the general $[2^r - 1, 2^r - r - 1]_2$ Hamming code is the code $\{\mathbf{c} \in \{0,1\}^{2^r - 1} | H_r \cdot \mathbf{c}^T = \mathbf{0}\}$.

Next we argue that the above Hamming code has distance 3 (in Proposition 1.5.1 we argued this for $r = 3$).

Proposition 2.4.1. *The Hamming code* $[2^r-1, 2^r-r-1, 3]_2$ *has distance 3.*

Proof. No two columns in $\mathbf{H}_r$ are linearly dependent. If they were, we would have $\mathbf{H}_r^i + \mathbf{H}_r^j = \mathbf{0}$, but this is impossible since they differ in at least one bit (being binary representations of integers, $i \neq j$). Thus, by Proposition 2.3.5, the distance is at least 3. It is at most 3, since (e.g.) $\mathbf{H}_r^1 + \mathbf{H}_r^2 + \mathbf{H}_r^3 = \mathbf{0}$. □

Now note that under the Hamming bound for $d = 3$ (Theorem 1.6.1), $k \leq n - \log_2(n+1)$, so for $n = 2^r - 1$, $k \leq 2^r - r - 1$. Hence, the Hamming code is a perfect code. (See Definition 1.7.2.)

In Question 1.7.1, we asked which codes are perfect codes. Interestingly, the only perfect binary codes are the following:

- The Hamming codes which we just studied.
- The trivial $[n, 1, n]_2$ codes for odd n (which have 0^n and 1^n as the only codewords): see Exercise 2.22.
- Two codes due to Golay [25].

2.5 Family of codes

Till now, we have mostly studied specific codes, that is, codes with *fixed* block lengths and dimension. The only exception was the "family" of $[2^r-1, 2^r-r-1, 3]_2$ Hamming codes (for $r \geq 2$) that we studied in the last section. We will see shortly that when we do an asymptotic study of codes (which is what we will do), it makes more sense to talk about a family of codes. First, we define the notion of family of codes:

Definition 2.5.1 (Family of codes)**.** $C = \{C_i\}_{i \geq 1}$ is a family of codes where C_i is a $(n_i, k_i, d_i)_q$ code for each i (and we assume $n_{i+1} > n_i$). The rate of C is defined as

$$R(C) = \lim_{i \to \infty} \left\{ \frac{k_i}{n_i} \right\}.$$

The relative distance of C is defined as

$$\delta(C) = \lim_{i \to \infty} \left\{ \frac{d_i}{n_i} \right\}.$$

For example, C_H the family of Hamming code is a family of codes with $n_i = 2^i - 1, k_i = 2^i - i - 1, d_i = 3$ and thus,

$$R(C_H) = \lim_{i \to \infty} 1 - \frac{i}{2^i - 1} = 1,$$

and

$$\delta(C_H) = \lim_{i \to \infty} \frac{3}{2^i - 1} = 0.$$

We will mostly work with family of codes from now on. This is necessary as we will study the asymptotic behavior of algorithms for codes, which does not make sense for a fixed code. For example, when we say that a decoding algorithm for a code C takes $O(n^2)$ time, we would be implicitly assuming that C is a family of codes and that the algorithm has an $O(n^2)$ running time when the block length is large enough. From now on, unless mentioned otherwise, whenever we talk about a code, we will be implicitly assuming that we are talking about a family of codes.

Given that we can only formally talk about asymptotic run time of algorithms, we now also state our formal notion of efficient algorithms:

We'll call an algorithm related to a code of block length n to be efficient, if it runs in time polynomial in n.

For all the specific codes that we will study in this book, the corresponding family of codes will be a "family" in a more natural sense. In other words, all the specific codes in a family of codes will be the "same" code except with different parameters. A bit more formally, we will consider families $\{C_i\}_i$, where given i, one can compute a sufficient description of C_i efficiently.[4]

Finally, the definition of a family of code allows us to present the final version of the the big motivating question for the book. The last formal version of the main question we considered was Question 1.4.1, where we were interested in the tradeoff of rate R and distance d. The comparison was somewhat unfair because R was a ratio while d was an integer. A more appropriate comparison should be between rate R and the relative distance δ. Further, we would be interested in tackling in the main motivating question for families of codes, which results in the following final version:

Question 2.5.1. *What is the optimal tradeoff between $R(C)$ and $\delta(C)$ that can be achieved by some code family C?*

2.6 Efficient Decoding of Hamming codes

We have shown that Hamming code has distance of 3 and thus, by Proposition 1.4.1, can correct one error. However, this is a *combinatorial* result and does not give us an efficient algorithm. One obvious candidate for decoding is the MLD function. Unfortunately, the only implementation of MLD that we know is the one in Algorithm 1, which will take time $2^{\Theta(n)}$, where n is the block length of the Hamming code. However, we can do much better. Consider the following simple algorithm: given the received word $\mathbf{y}$, first check if it is indeed a valid codeword. If it is, we are done. Otherwise, flip each of the n bits and check if the resulting vector is a valid codeword. If so, we have successfully decoded from one error. (If none of the checks are successful,

[4] We stress that this is not *always* going to be the case. In particular, we will consider "random" codes where this efficient constructibility will not be true.

then we declare a decoding failure.) Algorithm 2 formally presents this algorithm (where $C_{H,r}$ is the $[2^r-1, 2^r-r-1, 3]_2$ Hamming code).[5]

Algorithm 2 Naive Decoder for Hamming Code

INPUT: Received word $\mathbf{y}$
OUTPUT: $\mathbf{c}$ if $\Delta(\mathbf{y},\mathbf{c}) \leq 1$ else `Fail`

1: IF $\mathbf{y} \in C_{H,r}$ THEN
2: RETURN $\mathbf{y}$
3: FOR $i = 1 \ldots n$ DO
4: $\mathbf{y}' \leftarrow \mathbf{y} + \mathbf{e}_i$ ▷ $\mathbf{e}_i$ is the ith standard vector
5: IF $\mathbf{y}' \in C_{H,r}$ THEN
6: RETURN $\mathbf{y}'$
7: RETURN `Fail`

It is easy to check that Algorithm 2 can correct up to 1 error. If each of the checks $\mathbf{y}' \in C_{H,r}$ can be done in $T(n)$ time, then the time complexity of the proposed algorithm will be $O(nT(n))$. Note that since $C_{H,r}$ is a linear code (and dimension $k = n - O(\log n)$) by Proposition 2.3.3, we have $T(n) = O(n \log n)$. Thus, the proposed algorithm has running time $O(n^2 \log n)$.

Note that Algorithm 2 can be generalized to work for any linear code C with distance $2t+1$ (and hence, can correct up to t errors): go through all possible error vectors $\mathbf{z} \in [q]^n$ (with $wt(\mathbf{z}) \leq t$) and check if $\mathbf{y} - \mathbf{z}$ is in the code or not. Algorithm 3 presents the formal algorithm (where C is an $[n,k,2t+1]_q$ code). The number of error patterns $\mathbf{z}$ considered by Algorithm 3

Algorithm 3 Decoder for Any Linear Code

INPUT: Received word $\mathbf{y}$
OUTPUT: $\mathbf{c} \in C$ if $\Delta(\mathbf{y},\mathbf{c}) \leq t$ else `Fail`

1: FOR $i = 0 \ldots t$ DO
2: FOR $S \subseteq [n]$ such that $|S| = i$ DO
3: FOR $\mathbf{z} \in \mathbb{F}_q^n$ such that $wt(\mathbf{z}_S) = wt(\mathbf{z}) = i$ DO
4: IF $\mathbf{y} - \mathbf{z} \in C$ THEN
5: RETURN $\mathbf{y} - \mathbf{z}$
6: RETURN `Fail`

is[6] $\sum_{i=0}^{t} \binom{n}{i}(q-1)^i \leq O((nq)^t)$. Further by Proposition 2.3.3, Step 4 can be performed with $O(n^2)$ operations over $\mathbb{F}_q$. Thus, Algorithm 3 runs in with $O(n^{t+2}q^t)$ operations over $\mathbb{F}_q$, which for q being polynomial small in n, is $n^{O(t)}$ operations. In other words, the algorithm will have

[5] Formally speaking, a decoding algorithm should return the transmitted message $\mathbf{x}$ but Algorithm 2 actually returns $C_{H,r}(\mathbf{x})$. However, since $C_{H,r}$ is a linear code, it is not too hard to see that one can obtain $\mathbf{x}$ from $C_{H,r}(\mathbf{x})$ in $O(n^3)$ time– see Exercise 2.23. Further, for $C_{H,r}$ one can do this in $O(n)$ time– see Exercise 2.24.

[6] Recall (1.14).

polynomial running time for codes with constant distance (though the running time would not be practical even for moderate values of t).

However, it turns out that for Hamming codes there exists a decoding algorithm with an $O(n^2)$ running time. To see this first note that if the received word $\mathbf{y}$ has no errors then $H_r \cdot \mathbf{y}^T = \mathbf{0}$. If not, $\mathbf{y} = \mathbf{c} + \mathbf{e}_i$, where $\mathbf{c} \in C$ and $\mathbf{e}_i$ which is the unit vector with the only nonzero element at the i-th position. Thus, if H_r^i stands for the i-th column of H_r,

$$H_r \cdot \mathbf{y}^T = H_r \cdot \mathbf{c}^T + H_r \cdot (\mathbf{e}_i)^T = H_r \cdot (\mathbf{e}_i)^T = H_r^i,$$

where the second equality follows as $H_r \cdot \mathbf{c}^T = 0$, which in turn follows from the fact that $\mathbf{c} \in C$. In other words, $H_r \cdot \mathbf{y}^T$ gives the *location* of the error. This leads to Algorithm 4.

Algorithm 4 Efficient Decoder for Hamming Code

INPUT: Received word $\mathbf{y}$
OUTPUT: $\mathbf{c}$ if $\Delta(\mathbf{y}, \mathbf{c}) \leq 1$ else `Fail`

1: $\mathbf{b} \leftarrow H_r \cdot \mathbf{y}^T$.
2: Let $i \in [n]$ be the number whose binary representation is $\mathbf{b}$
3: IF $\mathbf{y} - \mathbf{e}_i \in C_H$ THEN
4: RETURN $\mathbf{y} - \mathbf{e}_i$
5: RETURN `Fail`

Since Step 1 in Algorithm 4 is a matrix vector multiplication (which can be done in $O(n \log n)$ time as the matrix is $O(\log n) \times n$) and Step 3 by Proposition 2.3.3 can be performed in $O(n \log n)$ time, Algorithm 4 runs in $O(n \log n)$ time. Thus,

Theorem 2.6.1. *The $[n = 2^r - 1, 2^r - r - 1, 3]_2$ Hamming code is 1-error correctable. Further, decoding can be performed in time $O(n \log n)$.*

2.7 Dual of a Linear Code

Till now, we have thought of parity check matrix as defining a code via its null space. However, we are not beholden to think of the parity check matrix in this way. A natural alternative is to use the parity check matrix as a generator matrix. The following definition addresses this question.

Definition 2.7.1 (Dual of a code)**.** Let H be a parity check matrix of a code C, then the code generated by H is called the dual of C. For any code C, its dual is denoted by $C^\perp$.

It is obvious from the definition that if C is an $[n, k]_q$ code then $C^\perp$ is an $[n, n-k]_q$ code. The first example that might come to mind is $C_{H,r}^\perp$, which is also known as the *Simplex code* (we will denote it by $C_{Sim,r}$). Adding an all 0's column to H_r and using the resulting matrix as a generating matrix, we get the *Hadamard* code (we will denote it by C_{Had}, r). We claim that $C_{Sim,r}$ and $C_{Had,r}$ are $[2^r - 1, r, 2^{r-1}]_2$ and $[2^r, r, 2^{r-1}]_2$ codes respectively. The claimed block length and dimension follow from the definition of the codes, while the distance follows from the following result.

Proposition 2.7.1. *$C_{Sim,r}$ and $C_{Had,r}$ both have a distance of 2^{r-1}.*

Proof. We first show the result for $C_{Had,r}$. In fact, we will show something stronger: every non-zero codeword in $C_{Had,r}$ has weight exactly equal to 2^{r-1} (the claimed distance follows from Proposition 2.3.4). Consider a message $\mathbf{x} \neq 0$. Let its ith entry be $x_i = 1$. $\mathbf{x}$ is encoded as

$$\mathbf{c} = (x_1, x_2, \ldots, x_r)(H_r^0, H_r^1, \ldots, H_r^{2^r-1}),$$

where H_r^j is the binary representation of $0 \leq j \leq 2^r - 1$ (that is, it contains all the vectors in $\{0,1\}^r$). Further note that the jth bit of the codeword $\mathbf{c}$ is $\langle \mathbf{x}, H_r^j \rangle$. Group all the columns of the generator matrix into pairs $(\mathbf{u}, \mathbf{v})$ such that $\mathbf{v} = \mathbf{u} + \mathbf{e}_i$ (i.e. $\mathbf{v}$ and $\mathbf{u}$ are the same except in the ith position). Notice that this partitions all the columns into 2^{r-1} disjoint pairs. Then,

$$\langle \mathbf{x}, \mathbf{v} \rangle = \langle \mathbf{x}, \mathbf{u} + \mathbf{e}_i \rangle = \langle \mathbf{x}, \mathbf{u} \rangle + \langle \mathbf{x}, \mathbf{e}_i \rangle = \langle \mathbf{x}, \mathbf{u} \rangle + x_i = \langle \mathbf{x}, \mathbf{u} \rangle + 1.$$

Thus we have that exactly one of $\langle \mathbf{x}, \mathbf{v} \rangle$ and $\langle \mathbf{x}, \mathbf{u} \rangle$ is 1. As the choice of the pair $(\mathbf{u}, \mathbf{v})$ was arbitrary, we have proved that for any non-zero codeword $\mathbf{c}$ such that $\mathbf{c} \in C_{Had}$, $wt(\mathbf{c}) = 2^{r-1}$.

For the simplex code, we observe that all codewords of $C_{Had,3}$ are obtained by padding a 0 to the beginning of the codewords in $C_{Sim,r}$, which implies that all non-zero codewords in $C_{Sim,r}$ also have a weight of 2^{r-1}, which completes the proof. □

We remark that the family of Hamming code has a rate of 1 and a (relative) distance of 0 while the families of Simplex/Hadamard codes have a rate of 0 and a relative distance of 1/2. Notice that both code families either have rate or relative distance equal to 0. Given this, the following question is natural special case of Question 2.5.1:

Question 2.7.1. *Does there exist a family of codes C such that $R(C) > 0$ and $\delta(C) > 0$ hold* simultaneously*?*

Codes that have the above property are called *asymptotically good*.

2.8 Exercises

Exercise 2.1. Prove that the set of rationals (i.e. the set of reals of the form $\frac{a}{b}$, where both a and $b \neq 0$ are integers), denoted by $\mathbb{Q}$, is a field.

Exercise 2.2. Let q be a prime power. Let $x \in \mathbb{F}_q$ such that $x \notin \{0,1\}$. Then prove that for any $n \leq q-1$:

$$\sum_{i=0}^{n} x^i = \frac{x^{n+1}-1}{x-1}.$$

Exercise 2.3. The main aim of this exercise is to prove the following identity that is true for any $\alpha \in \mathbb{F}_q$:

$$\alpha^q = \alpha \tag{2.4}$$

To make progress towards the above we will prove a sequence of properties of *groups*. A group G is a pair $(S, \circ)$ where the operator $\circ : G \times G \to G$ such that $\circ$ is commutative[7] and the elements of S are closed under $\circ$. Further, there is a special element $\iota \in S$ that is the identity element and every element $a \in S$ has an inverse element $b \in S$ such that $a \circ b = \iota$. Note that a finite field $\mathbb{F}_q$ consists of an *additive group* with the + operator (and 0 as additive identity) and a *multiplicative* group on the non-zero elements of $\mathbb{F}_q$ (which is also denoted by $\mathbb{F}_q^*$) with the $\cdot$ operator (and 1 as the multiplicative identity).[8]

For the rest of the problem let $G = (S, \cdot)$ be a multiplicative group with $|G| = m$. Prove the following statements.

1. For any $\beta \in G$, let $o(\beta)$ be the smallest integer o such that $\beta^o = 1$. Prove that such an $o \leq m$ always exists. Further, argue that $T = \{1, \beta, \ldots, \beta^{o-1}\}$ also forms a group. $(T, \cdot)$ is called a *sub-group* of G and $o(\beta)$ is called the *order* of β.

2. For any $g \in G$, define the *coset* (w.r.t. T) as

$$gT = \{g \cdot \beta | \beta \in T\}.$$

   Prove that if $g \cdot h^{-1} \in T$ then $gT = hT$ and $gT \cap hT = \emptyset$ otherwise. Further argue that these cosets partition the group G into disjoint sets.

3. Argue that for any $g \in G$, we have $|gT| = |T|$.

4. Using the above results or otherwise, argue that for any $\beta \in G$, we have

$$\beta^m = 1.$$

5. Prove (2.4).

Exercise 2.4. Prove that for $q = 2$, the second condition in Definition 2.2.1 is implied by the first condition.

Exercise 2.5. Prove that G_2 from (2.3) has full rank.

Exercise 2.6. In this problem we will look at the problem of solving a system of linear equations over $\mathbb{F}_q$. That is, one needs to solve for unknowns $x_1, \ldots, x_n$ given the following m linear equations (where $a_{i,j}, b_i \in \mathbb{F}_q$ for $1 \leq i \leq m$ and $1 \leq j \leq n$):

$$a_{1,1}x_1 + a_{1,2}x_2 + \cdots + a_{1,n}x_n = b_1.$$

$$a_{2,1}x_1 + a_{2,2}x_2 + \cdots + a_{2,n}x_n = b_2.$$

$$\vdots$$

$$a_{m,1}x_1 + a_{m,2}x_2 + \cdots + a_{m,n}x_n = b_m.$$

[7] Technically, G is an *abelian* group.

[8] Recall Definition 2.1.1.

1. (*Warm-up*) Convince yourself that the above problem can be stated as $A \cdot \mathbf{x}^T = \mathbf{b}^T$, where A is an $m \times n$ matrix over $\mathbb{F}_q$, $\mathbf{x} \in \mathbb{F}_q^n$ and $\mathbf{b} \in \mathbb{F}_q^m$.

2. (*Upper Triangular Matrix*) Assume $n = m$ and that A is upper triangular, i.e. all diagonal elements ($a_{i,i}$) are non-zero and all lower triangular elements ($a_{i,j}$, $i > j$) are 0. Then present an $O(n^2)$ time[9] algorithm to compute the unknown vector $\mathbf{x}$.

3. (*Gaussian Elimination*) Assume that A has *full rank* (or equivalently a *rank* of n.)

   (a) Prove that the following algorithm due to Gauss converts A into an upper triangular matrix. By permuting the columns if necessary make sure that $a_{1,1} \neq 0$. (Why can one assume w.l.o.g. that this can be done?) Multiply all rows $1 < i \leq n$ with $\frac{a_{1,1}}{a_{i,1}}$ and then subtract $a_{1,j}$ from the (i,j)th entry $1 \leq j \leq n$. Recurse with the same algorithm on the $(n-1) \times (n-1)$ matrix A' obtained by removing the first row and column from A. (Stop when $n = 1$.)

   (b) What happens if A does not have full rank? Show how one can modify the algorithm above to either upper triangulate a matrix or report that it does not have full rank. (Convince yourself that your modification works.)

   (c) Call a system of equations $A \cdot \mathbf{x}^T = \mathbf{b}^T$ *consistent* if there exists a solution to $\mathbf{x} \in \mathbb{F}_q^n$. Show that there exists an $O(n^3)$ algorithm that finds the solution if the system of equations is consistent and A has full rank (and report "fail" otherwise).

4. ($m < n$ *case*) Assume that A has full rank, i.e. has a rank of m. In this scenario either the system of equations is inconsistent or there are q^{n-m} solutions to $\mathbf{x}$. Modify the algorithm from above to design an $O(m^2 n)$ time algorithm to output the solutions (or report that the system is inconsistent).

   - Note that in case the system is consistent there will be q^{n-m} solutions, which might be much bigger than $O(m^2 n)$. Show that this is not a problem as one can represent the solutions as system of linear equations. (I.e. one can have $n - m$ "free" variables and m "bound" variables.)

5. ($m > n$ *case*) Assume that A has full rank, i.e. a rank of n. In this scenario either the system of equations is inconsistent or there is a unique solution to $\mathbf{x}$. Modify the algorithm from above to design an $O(m^2 n)$ time algorithm to output the solution (or report that the system is inconsistent).

6. (*Non-full rank case*) Give an $O(m^2 n)$ algorithm for the general case, i.e. the $m \times n$ matrix A need not have full rank. (The algorithm should either report that the system of equations is inconsistent or output the solution(s) to $\mathbf{x}$.)

Exercise 2.7. Prove that the span of k linearly independent vectors over $\mathbb{F}_q$ has size exactly q^k.

[9] For this problem, any basic operation over $\mathbb{F}_q$ takes unit time.

Exercise 2.8. Let G and H be a generator and parity check matrix of the same linear code of dimension k and block length n. Then $G \cdot H^T = \mathbf{0}$.

Exercise 2.9. Let C be an $[n,k]_q$ linear code with a generator matrix with no all zeros columns. Then for every position $i \in [n]$ and $\alpha \in \mathbb{F}_q$, the number of codewords $\mathbf{c} \in C$ such that $c_i = \alpha$ is exactly q^{k-1}.

Exercise 2.10. Prove Proposition 2.3.1.

Exercise 2.11. Prove Proposition 2.3.2.

Exercise 2.12. Prove Proposition 2.3.3.

Exercise 2.13. A set of vector $S \subseteq \mathbb{F}_q^n$ is called t-wise independent if for every set of positions I with $|I| = t$, the set S projected to I has each of the vectors in $\mathbb{F}_q^t$ appear the same number of times. (In other words, if one picks a vector $(s_1, \ldots, s_n)$ from S at random then any of the t random variables are uniformly and independently random over $\mathbb{F}_q$).

Prove that any linear code C whose dual $C^\perp$ has distance $d^\perp$ is $(d^\perp - 1)$-wise independent.

Exercise 2.14. A set of vectors $S \subseteq \mathbb{F}_2^k$ is called ε-biased sample space if the following property holds. Pick a vector $X = (x_1, \ldots, x_k)$ uniformly at random from S. Then X has *bias* at most ε, that is, for every $I \subseteq [k]$,

$$\left| \Pr\left(\sum_{i \in I} x_i = 0 \right) - \Pr\left(\sum_{i \in I} x_i = 1 \right) \right| \leq \varepsilon.$$

We will look at some connections of such sets to codes.

1. Let C be an $[n,k]_2$ code such that all non-zero codewords have Hamming weight in the range $\left[\left(\frac{1}{2} - \varepsilon\right) n, \left(\frac{1}{2} + \varepsilon\right) n\right]$. Then there exists an ε-biased space of size n.

2. Let C be an $[n,k]_2$ code such that all non-zero codewords have Hamming weight in the range $\left[\left(\frac{1}{2} - \gamma\right) n, \left(\frac{1}{2} + \gamma\right) n\right]$ for some constant $0 < \gamma < 1/2$. Then there exists an ε-biased space of size $n^{O(\gamma^{-1} \cdot \log(1/\varepsilon))}$.

Exercise 2.15. Let C be an $[n,k,d]_q$ code. Let $\mathbf{y} = (y_1, \ldots, y_n) \in (\mathbb{F}_q \cup \{?\})^n$ be a received word[10] such that $y_i = ?$ for at most $d-1$ values of i. Present an $O(n^3)$ time algorithm that outputs a codeword $\mathbf{c} = (c_1, \ldots, c_n) \in C$ that agrees with y in all un-erased positions (i.e., $c_i = y_i$ if $y_i \neq ?$) or states that no such $\mathbf{c}$ exists. (Recall that if such a $\mathbf{c}$ exists then it is unique.)

Exercise 2.16. In the chapter, we did not talk about how to obtain the parity check matrix of a linear code from its generator matrix. In this problem, we will look at this "conversion" procedure.

(a) Prove that any generator matrix $\mathbf{G}$ of an $[n,k]_q$ code C (recall that $\mathbf{G}$ is a $k \times n$ matrix) can be converted into another equivalent generator matrix of the form $\mathbf{G}' = [\mathbf{I}_k | \mathbf{A}]$, where $\mathbf{I}_k$ is the $k \times k$ identity matrix and $\mathbf{A}$ is some $k \times (n-k)$ matrix. By "equivalent," we mean that the code generated by $\mathbf{G}'$ has a linear bijective map to C.

[10] A ? denotes an erasure.

Note that the code generated by $\mathbf{G}'$ has the message symbols as its first k symbols in the corresponding codeword. Such codes are called *systematic codes*. In other words, every linear code can be converted into a systematic code. Systematic codes are popular in practice as they allow for immediate access to the message symbols.

(b) Given an $k \times n$ generator matrix of the form $[\mathbf{I}_k|\mathbf{A}]$, give a corresponding $(n-k) \times n$ parity check matrix. Briefly justify why your construction of the parity check matrix is correct.

Hint: Try to think of a parity check matrix that can be decomposed into two submatrices: one will be closely related to $\mathbf{A}$ and the other will be an identity matrix, though the latter might not be a $k \times k$ matrix).

(c) Use part (b) to present a generator matrix for the $[2^r - 1, 2^r - r - 1, 3]_2$ Hamming code.

Exercise 2.17. So far in this book we have seen that one can modify one code to get another code with interesting properties (for example, the construction of the Hadamard code from the Simplex code from Section 2.7 and Exercise 1.7). In this problem you will need to come up with more ways of constructing new codes from existing ones.

Prove the following statements (recall that the notation $(n,k,d)_q$ code is used for general codes with q^k codewords where k need not be an integer, whereas the notation $[n,k,d]_q$ code stands for a *linear code* of dimension k):

1. If there exists an $(n,k,d)_{2^m}$ code, then there also exists an $(nm, km, d' \geq d)_2$ code.

2. If there exists an $[n,k,d]_{2^m}$ code, then there also exists an $[nm, km, d' \geq d]_2$ code.

3. If there exists an $[n,k,d]_q$ code, then there also exists an $[n-d, k-1, d' \geq \lceil d/q \rceil]_q$ code.

4. If there exists an $[n,k,\delta n]_q$ code, then for every $m \geq 1$, there also exists an $\left(n^m, k/m, \left(1-(1-\delta)^m\right) \cdot n^m\right)_{q^m}$ code.

5. If there exists an $[n,k,\delta n]_2$ code, then for every *odd* $m \geq 1$, there also exists an $\left[n^m, k, \frac{1}{2} \cdot \left(1-(1-2\delta)^m\right) \cdot n^m\right]$ code.

Note: In all the parts, the only things that you can assume about the original code are only the parameters given by its definition– nothing else!

Exercise 2.18. Let C_1 be an $[n,k_1,d_1]_q$ code and C_2 be an $[n,k_2,d_2]_q$ code. Then define a new code as follows:

$$C_1 \ominus C_2 = \{(\mathbf{c}_1, \mathbf{c}_1 + \mathbf{c}_2) | \mathbf{c}_1 \in C_1, \mathbf{c}_2 \in C_2\}.$$

Next we will prove interesting properties of this operations on codes:

1. If G_i is the generator matrix for C_i for $i \in [2]$, what is a generator matrix for $C_1 \ominus C_2$?

2. Argue that $C_1 \ominus C_2$ is an $[2n, k_1 + k_2, d \stackrel{\text{def}}{=} \min(2d_1, d_2)]_q$ code.

3. Assume there exists algorithms $\mathscr{A}_i$ for code C_i for $i \in [2]$ such that: (i) $\mathscr{A}_1$ can decode from e errors and s erasures such that $2e+s < d_1$ and (ii) $\mathscr{A}_2$ can decode from $\lfloor (d_2-1)/2 \rfloor$ errors. Then argue that one can correct $\lfloor (d-1)/2 \rfloor$ errors for $C_1 \ominus C_2$.
   Hint: Given a received word $(\mathbf{y}_1, \mathbf{y}_2) \in \mathbb{F}_q^n \times \mathbb{F}_{q'}^n$, first apply $\mathscr{A}_2$ on $\mathbf{y}_2 - \mathbf{y}_1$. Then create an intermediate received word for $\mathscr{A}_1$.

4. We will now consider a recursive construction of a binary linear code that uses the $\ominus$ operator. For integers $0 \leq r \leq m$, we define the code $C(r,m)$ as follows:

   - $C(r,r) = \mathbb{F}_2^r$ and $C(0,r)$ is the code with only two codewords: the all ones and all zeroes vector in $\mathbb{F}_2^r$.
   - For $1 < r < m$, $C(r,m) = C(r,m-1) \ominus C(r-1,m-1)$.

   Determine the parameters of the code $C(r,m)$.

Exercise 2.19. Let C_1 be an $[n_1,k_1,d_1]_2$ binary linear code, and C_2 an $[n_2,k_2,d_2]$ binary linear code. Let $C \subseteq \mathbb{F}_2^{n_1 \times n_2}$ be the subset of $n_1 \times n_2$ matrices whose columns belong to C_1 and whose rows belong to C_2. C is called the tensor of C_1 and C_2 and is denoted by $C_1 \otimes C_2$.

Prove that C is an $[n_1 n_2, k_1 k_2, d_1 d_2]_2$ binary linear code.

Exercise 2.20. In Section 2.4 we considered the *binary* Hamming code. In this problem we will consider the more general q-ary Hamming code. In particular, let q be a prime power and $r \geq 1$ be an integer. Define the following $r \times n$ matrix $H_{q,r}$, where each column is an non-zero vector from $\mathbb{F}_q^r$ such that the first non-zero entry is 1. For example,

$$H_{3,2} = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 2 \end{pmatrix}$$

In this problem we will derive the parameters of the code. Define the generalized Hamming code $C_{H,r,q}$ to be the linear code whose parity check matrix is $H_{q,r}$. Argue that

1. The block length of $C_{H,r,q}$ is $n = \frac{q^r-1}{q-1}$.
2. $C_{H,q,r}$ has dimension $n-r$.
3. $C_{H,q,r}$ has distance 3.

Exercise 2.21. Design the best 6-ary code (family) with distance 3 that you can.

Hint: Start with a 7-ary Hamming code.

Exercise 2.22. Prove that the $[n,1,n]_2$ code for odd n (i.e. the code with the all zeros and all ones vector as it only two codewords) attains the Hamming bound (Theorem 1.7.1).

Exercise 2.23. Let C be an $[n,k]_q$ code with generator matrix G. Then given a codeword $\mathbf{c} \in C$ one can compute the corresponding message in time $O(kn^2)$.

Exercise 2.24. Given a $\mathbf{c} \in C_{H,r}$, one can compute the corresponding message in time $O(n)$.

Exercise 2.25. Let C be an $(n,k)_q$ code. Prove that if C can be decoded from e errors in time $T(n)$, then it can be decoded from $n+c$ errors in time $O((nq)^c \cdot T(n))$.

Exercise 2.26. Show that the bound of kd of the number of ones in the generator matrix of any binary linear code (see Exercise 1.12) cannot be improved for every code.

Exercise 2.27. Let C be a linear code. Then prove that $\left(C^{\perp}\right)^{\perp} = C$.

Exercise 2.28. Note that for any linear code C, the codewords $\mathbf{0}$ is in both C and $C^{\perp}$. Show that there exists a linear code C such that it shares a non-zero codeword with $C^{\perp}$.

Exercise 2.29. We go into a bit of diversion and look at how finite fields are different from infinite fields (e.g. $\mathbb{R}$). Most of the properties of linear subspaces that we have used for linear codes (e.g. notion of dimension, the existence of generator and parity check matrices, notion of duals) also hold for linear subspaces over $\mathbb{R}$.[11] One trivial property that holds for linear subspaces over finite fields that does not hold over $\mathbb{R}$ is that linear subspaces over $\mathbb{F}_q$ with dimension k has size q^k (though this is a trivial consequence that F_q are finite field while $\mathbb{R}$ is an infinite field). Next, we consider a more subtle distinction.

Let $S \subseteq \mathbb{R}^n$ be a linear subspace over $\mathbb{R}$ and let $S^{\perp}$ is the dual of S. Then show that

$$S \cap S^{\perp} = \{\mathbf{0}\}.$$

By contrast, linear subspaces over finite fields can have non-trivial intersection with their duals (see e.g. Exercise 2.28).

Exercise 2.30. A linear code C is called *self-orthogonal* if $C \subseteq C^{\perp}$. Show that

1. The binary repetition code with even number of repetitions is self-orthogonal.

2. The Hadamard code $C_{Had,r}$ is self-orthogonal.

Exercise 2.31. A linear code C is called self dual if $C = C^{\perp}$. Show that for

1. Any self dual code has dimension $n/2$.

2. Prove that the following code is self-dual

$$\{(\mathbf{x},\mathbf{x}) | \mathbf{x} \in \mathbb{F}_2^k\}.$$

Exercise 2.32. Given a code C a *puncturing* of C is another code C' where the same set of positions are dropped in all codewords of C. More precisely, if $C \subseteq \Sigma^n$ and the set of punctured positions is $P \subseteq [n]$, then the punctured code is $\{(c_i)_{i \notin P} | (c_1, \ldots, c_n) \in C\}$.

Prove that a linear code with no repetitions (i.e. there are no two positions $i \neq j$ such that for every codeword $\mathbf{c} \in C$, $c_i = c_i$) is a puncturing of the Hadamard code. Hence, Hadamard code is the "longest" linear code that does not repeat.

[11] A linear subspace $S \subseteq \mathbb{R}^n$ is the same as in Definition 2.2.1 where all occurrences of the finite field $\mathbb{F}_q$ is replaced by $\mathbb{R}$.

Exercise 2.33. In this problem we will consider the *long code*. For the definition, we will use the functional way of looking at the ambient space as mentioned in Remark 1.2.1. A long code of dimension k is a binary code such that the codeword corresponding to $\mathbf{x} = \mathbb{F}_2^k$, is the function $f : \{0,1\}^{2^k} \to \{0,1\}$ defined as follows. For any $\mathbf{m} \in \{0,1\}^{\mathbb{F}_2^k}$, we have $f((m_\alpha)_{\alpha \in \mathbb{F}_2^k}) = m_{\mathbf{x}}$. Derive the parameters of the long code.

Finally, argue that the long code is the code with the longest block length such that the codewords do not have a repeated coordinate (i.e. there does not exists $i \neq j$ such that for every codeword $\mathbf{c}$, $c_i = c_j$). (Contrast this with the property of Hadamard code above.)

2.9 Bibliographic Notes

Finite fields are also called Galois fields (another common notation for $\mathbb{F}_q$ is $GF(q)$), named after Évariste Galois, whose worked laid the foundations of their theory. (Galois led an extremely short and interesting life, which ended in death from a duel.) For a more thorough treatment refer to any standard text on algebra or the book on finite fields by Lidl and Niederreiter [53].

The answer to Question 1.7.1 was proved by van Lint [75] and Tietavainen [74].

Chapter 3

Probability as Fancy Counting and the q-ary Entropy Function

In the first half of this chapter, we will develop techniques that will allow us to answer questions such as

Question 3.0.1. *Does there exist a* $[2,2,1]_2$ *code?*

We note that the answer to the above question is trivially yes: just pick the generator matrix to be the 2×2 identity matrix. However, we will use the above as a simple example to illustrate a powerful technique called the *probabilistic method*.

As the name suggests, the method uses probability. Before we talk more about the probabilistic method, we do a quick review of the basics of probability that we will need in this book.

3.1 A Crash Course on Probability

In this book, we will only consider probability distributions defined over finite spaces. In particular, given a finite domain $\mathbb{D}$, a probability distribution is defined as a function

$$p : \mathbb{D} \to [0,1] \text{ such that } \sum_{x \in \mathbb{D}} p(x) = 1,$$

where $[0,1]$ is shorthand for the interval of all real numbers between 0 and 1. In this book, we will primarily deal with the following special distribution:

Definition 3.1.1 (Uniform Distribution)**.** The *uniform distribution* over $\mathbb{D}$, denoted by $\mathcal{U}_{\mathbb{D}}$, is given by

$$\mathcal{U}_{\mathbb{D}}(x) = \frac{1}{|\mathbb{D}|} \text{ for every } x \in \mathbb{D}.$$

Typically we will drop the subscript when the domain $\mathbb{D}$ is clear from the context.

G	$\mathscr{U}(G)$	V_{00}	V_{01}	V_{10}	V_{11}
$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$	$\frac{1}{16}$	0	0	0	0
$\begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$	$\frac{1}{16}$	0	1	0	1
$\begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix}$	$\frac{1}{16}$	0	1	0	1
$\begin{pmatrix} 0 & 0 \\ 1 & 1 \end{pmatrix}$	$\frac{1}{16}$	0	2	0	2
$\begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$	$\frac{1}{16}$	0	0	1	1
$\begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix}$	$\frac{1}{16}$	0	1	1	0
$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$	$\frac{1}{16}$	0	1	1	2
$\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$	$\frac{1}{16}$	0	2	1	1

G	$\mathscr{U}(G)$	V_{00}	V_{01}	V_{10}	V_{11}
$\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$	$\frac{1}{16}$	0	0	1	1
$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$	$\frac{1}{16}$	0	1	1	2
$\begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix}$	$\frac{1}{16}$	0	1	1	0
$\begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$	$\frac{1}{16}$	0	2	1	1
$\begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix}$	$\frac{1}{16}$	0	0	2	2
$\begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$	$\frac{1}{16}$	0	1	2	1
$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$	$\frac{1}{16}$	0	1	2	1
$\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$	$\frac{1}{16}$	0	2	2	0

Table 3.1: Uniform distribution over $\mathbb{F}_2^{2\times 2}$ along with values of four random variables.

For example, consider the domain $\mathbb{D} = \mathbb{F}_2^{2\times 2}$, i.e. the set of all 2×2 matrices over $\mathbb{F}_2$. (Note that each such matrix is a generator matrix of some $[2,2]_2$ code.) The first two columns of Table 3.1 list the elements of this $\mathbb{D}$ along with the corresponding probabilities for the uniform distribution.

Typically, we will be interested in a real-valued function defined on $\mathbb{D}$ and how it behaves under a probability distribution defined over $\mathbb{D}$. This is captured by the notion of a random variable:

Definition 3.1.2 (Random Variable)**.** Let $\mathbb{D}$ be a finite domain and $I \subset \mathbb{R}$ be a finite[1] subset. Let p be a probability distribution defined over $\mathbb{D}$. A *random variable* is a function:

$$V : \mathbb{D} \to I.$$

The *expectation* of V is defined as

$$\mathbb{E}[V] = \sum_{x \in \mathbb{D}} p(x) \cdot V(x).$$

For example, given $(i, j) \in \{0,1\}^2$, let V_{ij} denote the random variable $V_{ij}(G) = wt\big((i, j) \cdot G\big)$, for any $G \in \mathbb{F}_2^{2\times 2}$. The last four columns of Table 3.1 list the values of these four random variables.

In this book, we will mainly consider binary random variables, i.e., with $I = \{0,1\}$. In particular, given a predicate or *event* E over $\mathbb{D}$, we will define its *indicator variable* $\mathbb{1}_E$ to be 1 if E is

[1] In general, I need not be finite. However, for this book this definition suffices.

true and 0 if E is false. Sometimes, we will abuse notation and use E instead of $\mathbb{1}_E$. For example, consider the expectations of the four indicator variables:

$$\begin{aligned}
\mathbb{E}\left[\mathbb{1}_{V_{00}=0}\right] &= 16 \cdot \frac{1}{16} = 1. \\
\mathbb{E}\left[\mathbb{1}_{V_{01}=0}\right] &= 4 \cdot \frac{1}{16} = \frac{1}{4}. && (3.1) \\
\mathbb{E}\left[\mathbb{1}_{V_{10}=0}\right] &= 4 \cdot \frac{1}{16} = \frac{1}{4}. && (3.2) \\
\mathbb{E}\left[\mathbb{1}_{V_{11}=0}\right] &= 4 \cdot \frac{1}{16} = \frac{1}{4}. && (3.3)
\end{aligned}$$

3.1.1 Some Useful Results

Before we proceed, we record a simple property of indicator variables that will be useful. (See Exercise 3.1.)

Lemma 3.1.1. *Let E be any event. Then*

$$\mathbb{E}[\mathbb{1}_E] = \Pr[E \textit{ is true}].$$

Next, we state a simple yet useful property of expectation of a sum of random variables:

Proposition 3.1.2 (Linearity of Expectation)**.** *Given random variables $V_1, \ldots, V_m$ defined over the same domain $\mathbb{D}$ and with the same probability distribution p, we have*

$$\mathbb{E}\left[\sum_{i=1}^{m} V_i\right] = \sum_{i=1}^{m} \mathbb{E}[V_i].$$

Proof. For notational convenience, define $V = V_1 + \cdots + V_m$. Thus, we have

$$\begin{aligned}
\mathbb{E}[V] &= \sum_{x \in \mathbb{D}} V(x) \cdot p(x) && (3.4) \\
&= \sum_{x \in \mathbb{D}} \left(\sum_{i=1}^{m} V_i(x) \right) \cdot p(x) && (3.5) \\
&= \sum_{i=1}^{m} \sum_{x \in \mathbb{D}} V_i(x) \cdot p(x) && (3.6) \\
&= \sum_{i=1}^{m} \mathbb{E}[V_i]. && (3.7)
\end{aligned}$$

In the equalities above, (3.4) and (3.7) follow from the definition of expectation of a random variable. (3.5) follows from the definition of V and (3.6) follows by switching the order of the two summations. □

As an example, we have

$$\mathbb{E}\left[\mathbb{1}_{V_{01}=0} + \mathbb{1}_{V_{10}=0} + \mathbb{1}_{V_{11}=0}\right] = \frac{3}{4} \tag{3.8}$$

Frequently, we will need to deal with the probability of the "union" of events. We will use the following result to upper bound such probabilities:

Proposition 3.1.3 (Union Bound)**.** *Given m binary random variables $A_1, \ldots, A_m$, we have*

$$\Pr\left[\left(\bigvee_{i=1}^{m} A_i\right) = 1\right] \leq \sum_{i=1}^{m} \Pr\left[A_i = 1\right].$$

Proof. For every $i \in [m]$, define

$$S_i = \{x \in \mathbb{D} | A_i(x) = 1\}.$$

Then we have

$$\Pr\left[\left(\bigvee_{i=1}^{m} A_i\right) = 1\right] = \sum_{x \in \cup_{i=1}^{m} S_i} p(x) \tag{3.9}$$

$$\leq \sum_{i=1}^{m} \sum_{x \in S_i} p(x) \tag{3.10}$$

$$= \sum_{i=1}^{m} \Pr[A_i = 1]. \tag{3.11}$$

In the above, (3.9) and (3.11) follow from the definition of S_i. (3.10) follows from the fact that some of the $x \in \cup_i S_i$ get counted more than once. □

We remark that the union bound is tight when the events are *disjoint*. (In other words, using the notation in the proof above, when $S_i \cap S_j = \emptyset$ for every $i \neq j$.)

As an example, let $A_1 = \mathbb{1}_{V_{01}=0}$, $A_2 = \mathbb{1}_{V_{10}=0}$ and $A_3 = \mathbb{1}_{V_{11}=0}$. Note that in this case the event $A_1 \vee A_2 \vee A_3$ is the same as the event that there exists a non-zero $\mathbf{m} \in \{0,1\}^2$ such that $wt(\mathbf{m} \cdot G) = 0$. Thus, the union bound implies (that under the uniform distribution over $\mathbb{F}_2^{2\times 2}$)

$$\Pr\left[\text{There exists an } \mathbf{m} \in \{0,1\}^2 \setminus \{(0,0)\}, \text{ such that } wt(\mathbf{m}G) = 0\right] \leq \frac{3}{4}. \tag{3.12}$$

Finally, we present two bounds on the probability of a random variable deviating significantly from its expectation. The first bound holds for any random variable:

Lemma 3.1.4 (Markov Bound)**.** *Let V be a non-zero random variable. Then for any $t > 0$,*

$$\Pr[V \geq t] \leq \frac{\mathbb{E}[V]}{t}.$$

In particular, for any $a \geq 1$,

$$\Pr[V \geq a \cdot \mathbb{E}[V]] \leq \frac{1}{a}.$$

Proof. The second bound follows from the first bound by substituting $t = a \cdot \mathbb{E}[V]$. Thus, to complete the proof, we argue the first bound. Consider the following sequence of relations:

$$\mathbb{E}[V] = \sum_{i \in [0,t)} i \cdot \Pr[V = i] + \sum_{i \in [t,\infty)} i \cdot \Pr[V = i] \tag{3.13}$$

$$\geq \sum_{i \geq t} i \cdot \Pr[V = i] \tag{3.14}$$

$$\geq t \cdot \sum_{i \geq t} \Pr[V = i] \tag{3.15}$$

$$= t \cdot \Pr[V \geq t]. \tag{3.16}$$

In the above relations, (3.13) follows from the definition of expectation of a random variable and the fact that V is positive. (3.14) follows as we have dropped some non-negative terms. (3.15) follows by noting that in the summands $i \geq t$. (3.16) follows from the definition of $\Pr[V \geq t]$.

The proof is complete by noting that (3.16) implies the claimed bound. □

The second bound works only for sums of *independent* random variables. We begin by defining independent random variables:

Definition 3.1.3 (Independence)**.** Two random variables A and B are called *independent* if for every a and b in the ranges of A and B, we have

$$\Pr[A = a \wedge B = b] = \Pr[A = a] \cdot \Pr[B = b].$$

For example, for the uniform distribution in Table 3.1, let A denote the bit $G_{0,0}$ and B denote the bit $G_{0,1}$. It can be verified that these two random variables are independent. In fact, it can be verified all the random variables corresponding to the four bits in G are independent random variables. (We'll come to a related comment shortly.)

Another related concept that we will use is that of probability of an event happening conditioned on another event happening:

Definition 3.1.4 (Conditional Probability)**.** Given two events A and B defined over the same domain and probability distribution, we define the probability of A *conditioned on* B as

$$\Pr[A|B] = \frac{\Pr[A \text{ and } B]}{\Pr[B]}.$$

For example, note that

$$\Pr[\mathbb{1}_{V_{01}=1} | G_{0,0} = 0] = \frac{4/16}{1/2} = \frac{1}{2}.$$

The above definition implies that two events A and B are independent if and only if $\Pr[A] = \Pr[A|B]$. We will also use the following result later on in the book (see Exercise 3.2):

Lemma 3.1.5. *For any two events A and B defined on the same domain and the probability distribution:*

$$\Pr[A] = \Pr[A|B] \cdot \Pr[B] + \Pr[A|\neg B] \cdot \Pr[\neg B].$$

Next, we state the deviation bound. (We only state it for sums of binary random variables, which is the form that will be needed in the book.)

Theorem 3.1.6 (Chernoff Bound)**.** *Let $X_1, \ldots, X_m$ be independent binary random variables and define $X = \sum X_i$. Then the* multiplicative *Chernoff bound sates that for $0 < \varepsilon \leq 1$,*

$$\Pr[|X - \mathbb{E}(X)| > \varepsilon \mathbb{E}(X)] < 2e^{-\varepsilon^2 \mathbb{E}(X)/3},$$

and the additive *Chernoff bound states that*

$$\Pr[|X - \mathbb{E}(X)| > \varepsilon m] < 2e^{-\varepsilon^2 m/2}.$$

We omit the proof, which can be found in any standard textbook on randomized algorithms.

Finally, we present an alternate view of uniform distribution over "product spaces" and then use that view to prove a result that we will use later in the book. Given probability distributions p_1 and p_2 over domains $\mathbb{D}_1$ and $\mathbb{D}_2$ respectively, we define the product distribution $p_1 \times p_2$ over $\mathbb{D}_1 \times \mathbb{D}_2$ as follows: every element $(x, y) \in \mathbb{D}_1 \times \mathbb{D}_2$ under $p_1 \times p_2$ is picked by choosing x from $\mathbb{D}_1$ according to p_1 and y is picked *independently* from $\mathbb{D}_2$ under p_2. This leads to the following observation (see Exercise 3.3).

Lemma 3.1.7. *For any $m \geq 1$, the distribution $\mathcal{U}_{\mathbb{D}_1 \times \mathbb{D}_2 \times \cdots \times \mathbb{D}_m}$ is identical to the distribution $\mathcal{U}_{\mathbb{D}_1} \times \mathcal{U}_{\mathbb{D}_2} \times \cdots \times \mathcal{U}_{\mathbb{D}_m}$.*

For example, the uniform distribution in Table 3.1 can be described equivalently as follows: pick each of the four bits in G independently and uniformly at random from $\{0, 1\}$.

We conclude this section by proving the following result:

Lemma 3.1.8. *Given a non-zero vector $\mathbf{m} \in \mathbb{F}_q^k$ and a uniformly random $k \times n$ matrix G over $\mathbb{F}_q$, the vector $\mathbf{m} \cdot G$ is uniformly distributed over $\mathbb{F}_q^n$.*

Proof. Let the (j, i)th entry in G ($1 \leq j \leq k, 1 \leq i \leq n$) be denoted by g_{ji}. Note that as G is a random $k \times n$ matrix over $\mathbb{F}_q$, by Lemma 3.1.7, each of the g_{ji} is an independent uniformly random element from $\mathbb{F}_q$. Now, note that we would be done if we can show that for every $1 \leq i \leq n$, the ith entry in $\mathbf{m} \cdot G$ (call it b_i) is an independent uniformly random element from $\mathbb{F}_q$. To finish the proof, we prove this latter fact. If we denote $\mathbf{m} = (m_1, \ldots, m_k)$, then $b_i = \sum_{j=1}^{k} m_j g_{ji}$. Note that the disjoint entries of G participate in the sums for b_i and b_j for $i \neq j$. Given our choice of G, this implies that the random variables b_i and b_j are independent. Hence, to complete the proof we need to prove that b_i is a uniformly independent element of $\mathbb{F}_q$. The rest of the proof is a generalization of the argument we used in the proof of Proposition 2.7.1.

Note that to show that b_i is uniformly distributed over $\mathbb{F}_q$, it is sufficient to prove that b_i takes every value in $\mathbb{F}_q$ equally often over all the choices of values that can be assigned to $g_{1i}, g_{2i}, \ldots, g_{ki}$. Now, as $\mathbf{m}$ is non-zero, at least one of the its element is non-zero: without loss of generality assume that $m_1 \neq 0$. Thus, we can write $b_i = m_1 g_{1i} + \sum_{j=2}^{k} m_j g_{ji}$. Now, for every fixed assignment of values to $g_{2i}, g_{3i}, \ldots, g_{ki}$ (note that there are q^{k-1} such assignments), b_i takes a different value for each of the q distinct possible assignments to g_{1i} (this is where we use the assumption that $m_1 \neq 0$). Thus, over all the possible assignments of $g_{1i}, \ldots, g_{ki}$, b_i takes each of the values in $\mathbb{F}_q$ exactly q^{k-1} times, which proves our claim. □

3.2 The Probabilistic Method

The *probabilistic method* is a very powerful method in combinatorics which can be used to show the existence of objects that satisfy certain properties. In this course, we will use the probabilistic method to prove existence of a code $\mathscr{C}$ with certain property $\mathscr{P}$. Towards that end, we define a distribution $\mathscr{D}$ over all possible codes and prove that when $\mathscr{C}$ is chosen according to $\mathscr{D}$:

$$\Pr\left[\mathscr{C} \text{ has property } \mathscr{P}\right] > 0 \text{ or equivalently } \Pr\left[\mathscr{C} \text{ doesn't have property } \mathscr{P}\right] < 1.$$

Note that the above inequality proves the existence of $\mathscr{C}$ with property $\mathscr{P}$.

As an example consider Question 3.0.1. To answer this in the affirmative, we note that the set of all $[2,2]_2$ linear codes is covered by the set of all 2×2 matrices over $\mathbb{F}_2$. Then, we let $\mathscr{D}$ be the uniform distribution over $\mathbb{F}_2^{2\times 2}$. Then by Proposition 2.3.4 and (3.12), we get that

$$\Pr_{\mathscr{U}_{\mathbb{F}_2^{2\times 2}}}[\text{There is no } [2,2,1]_2 \text{ code}] \leq \frac{3}{4} < 1,$$

which by the probabilistic method answers the Question 3.0.1 in the affirmative.

For the more general case, when we apply the probabilistic method, the typical approach will be to define (sub-)properties $P_1, \ldots, P_m$ such that $\mathscr{P} = P_1 \wedge P_2 \wedge P_3 \ldots \wedge P_m$ and show that for every $1 \leq i \leq m$:

$$\Pr\left[\mathscr{C} \text{ doesn't have property } P_i\right] = \Pr\left[\overline{P_i}\right] < \frac{1}{m}.$$

Finally, by the union bound, the above will prove that[2] $\Pr\left[\mathscr{C} \text{ doesn't have property } \mathscr{P}\right] < 1$, as desired.

As an example, an alternate way to answer Question 3.0.1 in the affirmative is the following. Define $P_1 = \mathbb{1}_{V_{01}\geq 1}$, $P_2 = \mathbb{1}_{V_{10}\geq 1}$ and $P_3 = \mathbb{1}_{V_{11}\geq 1}$. (Note that we want a $[2,2]_2$ code that satisfies $P_1 \wedge P_2 \wedge P_3$.) Then, by (3.1), (3.2) and (3.3), we have for $i \in [3]$,

$$\Pr\left[\mathscr{C} \text{ doesn't have property } P_i\right] = \Pr\left[\overline{P_i}\right] = \frac{1}{4} < \frac{1}{3},$$

as desired.

Finally, we mention a special case of the general probabilistic method that we outlined above. In particular, let $\mathscr{P}$ denote the property that the randomly chosen $\mathscr{C}$ satisfies $f(\mathscr{C}) \leq b$. Then we claim (see Exercise 3.4) that $\mathbb{E}[f(C)] \leq b$ implies that $\Pr[\mathscr{C} \text{ has property } \mathscr{P}] > 0$. Note that this implies that $\mathbb{E}[f(C)] \leq b$ implies that there exists a code $\mathscr{C}$ such that $f(C) \leq b$.

3.3 The q-ary Entropy Function

We begin with the definition of a function that will play a central role in many of our combinatorial results.

[2] Note that $\overline{P} = \overline{P_1} \vee \overline{P_2} \vee \cdots \vee \overline{P_m}$.

Definition 3.3.1 (q-ary Entropy Function)**.** Let q be an integer and x be a real number such that $q \geq 2$ and $0 \leq x \leq 1$. Then the *q-ary entropy function* is defined as follows:

$$H_q(x) = x\log_q(q-1) - x\log_q(x) - (1-x)\log_q(1-x).$$

Figure 3.1 presents a pictorial representation of the H_q function for the first few values of q. For the special case of $q = 2$, we will drop the subscript from the entropy function and denote

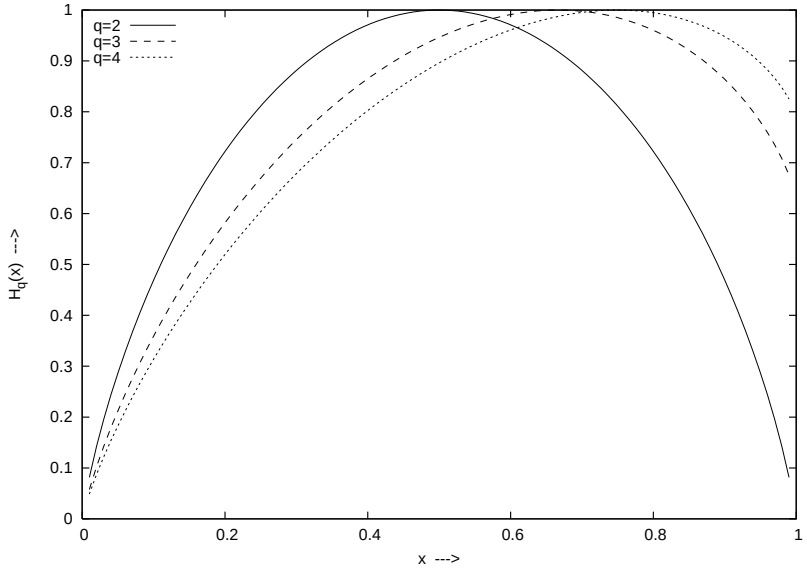


Figure 3.1: A plot of $H_q(x)$ for $q = 2, 3$ and 4. The maximum value of 1 is achieved at $x = 1 - 1/q$.

$H_2(x)$ by just $H(x)$, that is, $H(x) = -x\log x - (1-x)\log(1-x)$, where $\log x$ is defined as $\log_2(x)$ (we are going to follow this convention for the rest of the book).

Under the lens of Shannon's entropy function, $H(x)$ denotes the entropy of the distribution over $\{0,1\}$ that selects 1 with probability x and 0 with probability $1-x$. However, there is no similar analogue for the more general $H_q(x)$. The reason why this quantity will turn out to be so central in this book is that it is very closely related to the "volume" of a Hamming ball. We make this connection precise in the next subsection.

3.3.1 Volume of Hamming Balls

It turns out that in many of our combinatorial results, we will need good upper and lower bounds on the volume of a Hamming ball. Next we formalize the notion of the volume of a Hamming ball:

Definition 3.3.2 (Volume of a Hamming Ball)**.** Let $q \geq 2$ and $n \geq r \geq 1$ be integers. Then the volume of a Hamming ball of radius r is given by

$$Vol_q(r,n) = |B_q(\mathbf{0}, r)| = \sum_{i=0}^{r} \binom{n}{i} (q-1)^i.$$

The choice of $\mathbf{0}$ as the center for the Hamming ball above was arbitrary: since the volume of a Hamming ball is independent of its center (as is evident from the last equality above), we could have picked any center.

We will prove the following result:

Proposition 3.3.1. *Let $q \geq 2$ be an integer and $0 \leq p \leq 1 - \frac{1}{q}$ be a real. Then:*

(i) $Vol_q(pn, n) \leq q^{H_q(p)n}$; *and*

(ii) for large enough n, $Vol_q(pn, n) \geq q^{H_q(p)n - o(n)}$.

Proof. We start with the proof of *(i)*. Consider the following sequence of relations:

$$\begin{aligned}
1 &= (p + (1-p))^n \\
&= \sum_{i=0}^{n} \binom{n}{i} p^i (1-p)^{n-i} && (3.17) \\
&= \sum_{i=0}^{pn} \binom{n}{i} p^i (1-p)^{n-i} + \sum_{i=pn+1}^{n} \binom{n}{i} p^i (1-p)^{n-i} \\
&\geq \sum_{i=0}^{pn} \binom{n}{i} p^i (1-p)^{n-i} && (3.18) \\
&= \sum_{i=0}^{pn} \binom{n}{i} (q-1)^i \left(\frac{p}{q-1}\right)^i (1-p)^{n-i} \\
&= \sum_{i=0}^{pn} \binom{n}{i} (q-1)^i (1-p)^n \left(\frac{p}{(q-1)(1-p)}\right)^i \\
&\geq \sum_{i=0}^{pn} \binom{n}{i} (q-1)^i (1-p)^n \left(\frac{p}{(q-1)(1-p)}\right)^{pn} && (3.19) \\
&= \sum_{i=0}^{pn} \binom{n}{i} (q-1)^i \left(\frac{p}{q-1}\right)^{pn} (1-p)^{(1-p)n} && (3.20) \\
&\geq Vol_q(pn, n) q^{-H_q(p)n}. && (3.21)
\end{aligned}$$

In the above, (3.17) follows from the binomial expansion. (3.18) follows by dropping the second sum and (3.19) follows from that facts that $\frac{p}{(q-1)(1-p)} \leq 1$ (as[3] $p \leq 1 - 1/q$). Rest of the steps except (3.21) follow from rearranging the terms. (3.21) follows as $q^{-H_q(p)n} = \left(\frac{p}{q-1}\right)^{pn} (1-p)^{(1-p)n}$.

(3.21) implies that

$$1 \geq Vol_q(pn, n) q^{-H_q(p)n},$$

which proves *(i)*.

[3] Indeed, note that $\frac{p}{(q-1)(1-p)} \leq 1$ is true if $\frac{p}{1-p} \leq \frac{q-1}{1}$, which in turn is true if $p \leq \frac{q-1}{q}$, where the last step follows from Lemma B.2.1.

We now turn to the proof of part *(ii)*. For this part, we will need Stirling's approximation for $n!$ (Lemma B.1.2).

By the Stirling's approximation, we have the following inequality:

$$\begin{aligned}\binom{n}{pn} &= \frac{n!}{(pn)!((1-p)n)!} \\ &> \frac{(n/e)^n}{(pn/e)^{pn}((1-p)n/e)^{(1-p)n}} \cdot \frac{1}{\sqrt{2\pi p(1-p)n}} \cdot e^{\lambda_1(n)-\lambda_2(pn)-\lambda_2((1-p)n)} \\ &= \frac{1}{p^{pn}(1-p)^{(1-p)n}} \cdot \ell(n), \end{aligned} \tag{3.22}$$

where $\ell(n) = \frac{e^{\lambda_1(n)-\lambda_2(pn)-\lambda_2((1-p)n)}}{\sqrt{2\pi p(1-p)n}}$.

Now consider the following sequence of relations that complete the proof:

$$Vol_q(pn,n) \geq \binom{n}{pn}(q-1)^{pn} \tag{3.23}$$

$$> \frac{(q-1)^{pn}}{p^{pn}(1-p)^{(1-p)n}} \cdot \ell(n) \tag{3.24}$$

$$\geq q^{H_q(p)n-o(n)}. \tag{3.25}$$

In the above (3.23) follows by only looking at one term. (3.24) follows from (3.22) while (3.25) follows from the definition of $H_q(\cdot)$ and the fact that for large enough n, $\ell(n)$ is $q^{-o(n)}$. □

Next, we consider how the q-ary entropy function behaves for various ranges of its parameters.

3.3.2 Other Properties of the q-ary Entropy function

We begin by recording the behavior of q-ary entropy function for large q.

Proposition 3.3.2. *For small enough ε, $1 - H_q(\rho) \geq 1 - \rho - \varepsilon$ for every $0 < \rho \leq 1 - 1/q$ if and only if q is $2^{\Omega(1/\varepsilon)}$.*

Proof. We first note that by definition of $H_q\rho)$ and $H(\rho)$,

$$\begin{aligned} H_q(\rho) &= \rho \log_q(q-1) - \rho \log_q \rho - (1-\rho)\log_q(1-\rho) \\ &= \rho \log_q(q-1) + H(\rho)/\log_2 q. \end{aligned}$$

Now if $q \geq 2^{1/\varepsilon}$, we get that

$$H_q(\rho) \leq \rho + \varepsilon.$$

as $\log_q(q-1) \leq 1$ and $H(\rho) \leq 1$. Thus, we have argued that for $q \geq 2^{1/\varepsilon}$, we have $1 - H_q(\rho) \geq 1 - \rho - \varepsilon$, as desired.

Next, we consider the case when $q = 2^{o(1/\varepsilon)}$. We begin by claiming that for small enough ε,

$$\text{if } q \geq 1/\varepsilon^2 \text{ then } \log_q(q-1) \geq 1-\varepsilon.$$

Indeed, $\log_q(q-1) = 1 + (1/\ln q)\ln(1-1/q) = 1 - O\left(\frac{1}{q \ln q}\right)$,[4] which is at least $1-\varepsilon$ for $q \geq 1/\varepsilon^2$ (and small enough ε).

Finally, if $q = 2^{o\left(\frac{1}{\varepsilon}\right)}$, then for fixed ρ,

$$H(\rho)/\log q = \varepsilon \cdot \omega(1).$$

Then for $q = 2^{o\left(\frac{1}{\varepsilon}\right)}$ (but $q \geq 1/\varepsilon^2$) we have

$$\rho \log_q(q-1) + H(\rho)/\log q \geq \rho - \varepsilon + \varepsilon \cdot \omega(1) > \rho + \varepsilon,$$

which implies that

$$1 - H_q(\rho) < 1 - \rho - \varepsilon,$$

as desired. For $q \leq 1/\varepsilon^2$, Lemma 3.3.3 shows that $1 - H_q(\rho) \leq 1 - H_{1/\varepsilon^2}(\rho) < 1 - \rho - \varepsilon$, as desired. □

We will also be interested in how $H_q(x)$ behaves for fixed x and increasing q:

Lemma 3.3.3. *Let $q \geq 2$ be an integer and let $0 \leq \rho \leq 1 - 1/q$, then for any real $m \geq 1$ such that*

$$q^{m-1} \geq \left(1 + \frac{1}{q-1}\right)^{q-1}, \tag{3.26}$$

we have

$$H_q(\rho) \geq H_{q^m}(\rho).$$

Proof. Note that $H_q(0) = H_{q^m}(0) = 0$. Thus, for the rest of the proof we will assume that $\rho \in (0, 1-1/q]$.

As observed in the proof of Proposition 3.3.2, we have

$$H_q(\rho) = \rho \cdot \frac{\log(q-1)}{\log q} + H(\rho) \cdot \frac{1}{\log q}.$$

Using this, we obtain

$$H_q(\rho) - H_{q^m}(\rho) = \rho\left(\frac{\log(q-1)}{\log q} - \frac{\log(q^m-1)}{m \log q}\right) + H(\rho)\left(\frac{1}{\log q} - \frac{1}{m \log q}\right).$$

The above in turn implies that

$$\frac{1}{\rho} \cdot m \log q \cdot (H_q(\rho) - H_{q^m}(\rho)) = \log(q-1)^m - \log(q^m-1) + \frac{H(\rho)}{\rho}(m-1)$$

[4] The last equality follows from the fact that by Lemma B.2.2, for $0 < x < 1$, $\ln(1-x) = -O(x)$.

$$\geq \log(q-1)^m - \log(q^m-1) + \frac{H(1-1/q)}{1-1/q}(m-1) \tag{3.27}$$

$$= \log(q-1)^m - \log(q^m-1) + (m-1)\left(\log\frac{q}{q-1} + \frac{\log q}{q-1}\right)$$

$$= \log\left(\frac{(q-1)^m}{q^m-1}\cdot\left(\frac{q}{q-1}\right)^{m-1}\cdot q^{\frac{m-1}{q-1}}\right)$$

$$= \log\left(\frac{(q-1)\cdot q^{m-1}\cdot q^{\frac{m-1}{q-1}}}{q^m-1}\right)$$

$$\geq 0 \tag{3.28}$$

In the above (3.27) follows from the fact that $H(\rho)/\rho$ is decreasing[5] in ρ and that $\rho \leq 1-1/q$. (3.28) follows from the the claim that

$$(q-1)\cdot q^{\frac{m-1}{q-1}} \geq q.$$

Indeed the above follows from (3.26).

Finally, note that (3.28) completes the proof. □

Since $(1+1/x)^x \leq e$ (by Lemma B.2.5), we also have that (3.26) is also satisfied for $m \geq 1 + \frac{1}{\ln q}$. Further, we note that (3.26) is satisfied for every $m \geq 2$ (for any $q \geq 3$), which leads to the following (also see Exercise 3.5):

Corollary 3.3.4. *Let $q \geq 3$ be an integer and let $0 \leq \rho \leq 1-1/q$, then for any $m \geq 2$, we have*

$$H_q(\rho) \geq H_{q^m}(\rho).$$

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

Proposition 3.3.5. *For small enough $\varepsilon > 0$,*

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

where c_q is a constant that only depends on q.

Proof. The intuition behind the proof is the following. Since the derivative 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:

$$H_q(1-1/q-\varepsilon) = -\left(1-\frac{1}{q}-\varepsilon\right)\log_q\left(\frac{1-1/q-\varepsilon}{q-1}\right) - \left(\frac{1}{q}+\varepsilon\right)\log_q\left(\frac{1}{q}+\varepsilon\right)$$

[5] Indeed, $H(\rho)/\rho = \log(1/\rho) - (1/\rho - 1)\log(1-\rho)$. Note that the first term is deceasing in ρ. We claim that the second term is also decreasing in ρ– this e.g. follows from the observation that $-(1/\rho-1)\ln(1-\rho) = (1-\rho)(1+\rho/2!+\rho^2/3!+\cdots) = 1-\rho/2-\rho^2(1/2-1/3!)-\cdots$ is also decreasing in ρ.

$$
\begin{aligned}
&= -\log_q\left(\frac{1}{q}\left(1-\frac{\varepsilon q}{q-1}\right)\right)+\left(\frac{1}{q}+\varepsilon\right)\log_q\left(\frac{1-(\varepsilon q)/(q-1)}{1+\varepsilon q}\right)\\
&= 1-\frac{1}{\ln q}\left[\ln\left(1-\frac{\varepsilon q}{q-1}\right)-\left(\frac{1}{q}+\varepsilon\right)\ln\left(\frac{1-(\varepsilon q)/(q-1)}{1+\varepsilon q}\right)\right]\\
&= 1+o(\varepsilon^2)-\frac{1}{\ln q}\left[-\frac{\varepsilon q}{q-1}-\frac{\varepsilon^2 q^2}{2(q-1)^2}-\left(\frac{1}{q}+\varepsilon\right)\left(-\frac{\varepsilon q}{q-1}\right.\right.\\
&\qquad\left.\left.-\frac{\varepsilon^2 q^2}{2(q-1)^2}-\varepsilon q+\frac{\varepsilon^2 q^2}{2}\right)\right] \qquad (3.29)\\
&= 1+o(\varepsilon^2)-\frac{1}{\ln q}\left[-\frac{\varepsilon q}{q-1}-\frac{\varepsilon^2 q^2}{2(q-1)^2}\right.\\
&\qquad\left.-\left(\frac{1}{q}+\varepsilon\right)\left(-\frac{\varepsilon q^2}{q-1}+\frac{\varepsilon^2 q^3(q-2)}{2(q-1)^2}\right)\right]\\
&= 1+o(\varepsilon^2)-\frac{1}{\ln q}\left[-\frac{\varepsilon^2 q^2}{2(q-1)^2}+\frac{\varepsilon^2 q^2}{q-1}-\frac{\varepsilon^2 q^2(q-2)}{2(q-1)^2}\right] \qquad (3.30)\\
&= 1-\frac{\varepsilon^2 q^2}{2\ln q(q-1)}+o(\varepsilon^2)\\
&\le 1-\frac{\varepsilon^2 q^2}{4\ln q(q-1)}
\end{aligned}
$$

(3.31)

(3.29) follows from the fact that for $|x|<1$, $\ln(1+x)=x-x^2/2+x^3/3-\ldots$ (Lemma B.2.2) and by collecting the ε^3 and smaller terms in $o(\varepsilon^2)$. (3.30) follows by rearranging the terms and by absorbing the ε^3 terms in $o(\varepsilon^2)$. The last step is true assuming ε is small enough. □

Next, we look at the entropy function when its input is very close to 0.

Proposition 3.3.6. *For small enough $\varepsilon>0$,*

$$H_q(\varepsilon)=\Theta\left(\frac{1}{\log q}\cdot\varepsilon\log\left(\frac{1}{\varepsilon}\right)\right).$$

Proof. By definition

$$H_q(\varepsilon)=\varepsilon\log_q(q-1)+\varepsilon\log_q(1/\varepsilon)+(1-\varepsilon)\log_q(1/(1-\varepsilon)).$$

Since all the terms in the RHS are positive we have

$$H_q(\varepsilon)\ge\varepsilon\log(1/\varepsilon)/\log q. \qquad (3.32)$$

Further, by Lemma B.2.2, $(1-\varepsilon)\log_q(1/(1-\varepsilon))\le 2\varepsilon/\ln q$ for small enough ε. Thus, this implies that

$$H_q(\varepsilon)\le\frac{2+\ln(q-1)}{\ln q}\cdot\varepsilon+\frac{1}{\ln q}\cdot\varepsilon\ln\left(\frac{1}{\varepsilon}\right). \qquad (3.33)$$

(3.32) and (3.33) proves the claimed bound.

□

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 $0 \le x \le 1-1/q$. Finally, we will need the following lower bound.

Lemma 3.3.7. *For every $0 \le y \le 1-1/q$ and for every small enough $\varepsilon > 0$,*

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

where $c_q' \ge 1$ 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 derivative of $H_q^{-1}(y)$ increases with y. In particular, $(H_q^{-1})'(1) \ge (H_q^{-1})'(y)$ for every $0 \le y \le 1$. In other words, for every $0 < y \le 1$, and (small enough) $\delta > 0$, $\frac{H_q^{-1}(y) - H_q^{-1}(y-\delta)}{\delta} \le \frac{H_q^{-1}(1) - H_q^{-1}(1-\delta)}{\delta}$. Proposition 3.3.5 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 $\delta = \varepsilon^2/c_q'$. □

3.4 Exercises

Exercise 3.1. Prove Lemma 3.1.1.

Exercise 3.2. Prove Lemma 3.1.5.

Exercise 3.3. Prove Lemma 3.1.7.

Exercise 3.4. Let $\mathscr{P}$ denote the property that the randomly chosen $\mathscr{C}$ satisfies $f(\mathscr{C}) \le b$. Then $\mathbb{E}[f(C)] \le b$ implies that $\Pr[\mathscr{C} \text{ has property } \mathscr{P}] > 0$.

Exercise 3.5. Show that for any $Q \ge q \ge 2$ and $\rho \le 1-1/q$, we have $H_Q(\rho) \le H_q(\rho)$.

3.5 Bibliographic Notes

Shannon was one of the very early adopters of probabilistic method (and we will see one such use in Chapter 6). Later, the probabilistic method was popularized Erdős. For more on probabilistic method, see the book by Alon and Spencer [1].

Proofs of various concentration bounds can e.g. be found in [14].

Part II

The Combinatorics

Chapter 4

What Can and Cannot Be Done-I

In this chapter, we will try to tackle Question 2.5.1. We will approach this trade-off in the following way:

> If we fix the relative distance of the code to be δ, what is the best rate R that we can achieve?

Note that an upper bound on R is a *negative* result, while a lower bound on R is a *positive* result.

In this chapter, we will consider only one positive result, i.e. a lower bound on R called the Gilbert-Varshamov bound in Section 4.2. In Section 4.1, we recall a negative result that we have already seen– Hamming bound and state its asymptotic version to obtain an upper bound on R. We will consider two other upper bounds: the Singleton bound (Section 4.3), which gives a tight upper bound for large enough alphabets (but not binary codes) and the Plotkin bound (Section 4.4).

4.1 Asymptotic Version of the Hamming Bound

We have already seen an upper bound in Section 1.7 due to Hamming. However, we had stated this as an upper bound on the dimension k in terms of n, q and d. We begin by considering the trade-off between R and δ given by the Hamming bound. Recall that Theorem 1.7.1 states the following:

$$\frac{k}{n} \leq 1 - \frac{\log_q Vol_q\left(\left\lfloor \frac{d-1}{2} \right\rfloor, n\right)}{n}$$

Recall that Proposition 3.3.1 states following lower bound on the volume of a Hamming ball:

$$Vol_q\left(\left\lfloor \frac{d-1}{2} \right\rfloor, n\right) \geq q^{H_q\left(\frac{\delta}{2}\right)n - o(n)},$$

which implies the following asymptotic version of the Hamming bound:

$$R \leq 1 - H_q\left(\frac{\delta}{2}\right) + o(1).$$

See Figure 4.1 for a pictorial description of the Hamming bound for binary codes.

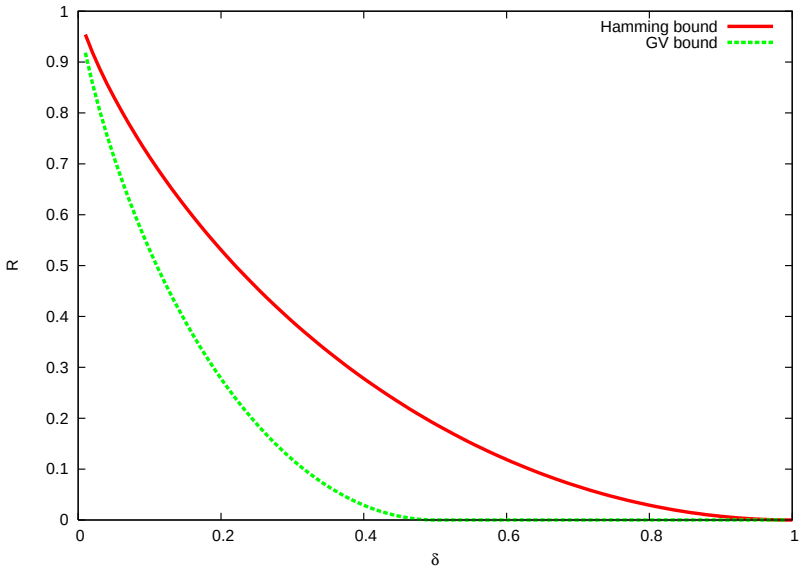


Figure 4.1: The Hamming and GV bounds for binary codes. Note that any point below the GV bound is achievable by some code while no point above the Hamming bound is achievable by any code. In this part of the book we would like to push the GV bound as much up as possible while at the same time try and push down the Hamming bound as much as possible.

4.2 Gilbert-Varshamov Bound

Next, we will switch gears by proving our first non-trivial lower bound on R in terms of δ. (In fact, this is the only positive result on the R vs δ tradeoff question that we will see in this book.) In particular, we will prove the following result:

Theorem 4.2.1 (Gilbert-Varshamov Bound)**.** *Let $q \geq 2$. For every $0 \leq \delta < 1 - \frac{1}{q}$, and $0 < \varepsilon \leq 1 - H_q(\delta)$, there exists a code with rate $R \geq 1 - H_q(\delta) - \varepsilon$ and relative distance δ.*

The bound is generally referred to as the GV bound. For a pictorial description of the GV bound for binary codes, see Figure 4.1. We will present the proofs for general codes and linear codes in Sections 4.2.1 and 4.2.2 respectively.

4.2.1 Greedy Construction

We will prove Theorem 4.2.1 for general codes by the following greedy construction (where $d = \delta n$): start with the empty code C and then keep on adding vectors not in C that are at Hamming distance at least d from all the existing codewords in C. Algorithm 5 presents a formal description of the algorithm and Figure 4.2 illustrates the first few executions of this algorithm.

We claim that Algorithm 5 terminates and the C that it outputs has distance d. The latter is true by step 2, which makes sure that in Step 3 we never add a vector $\mathbf{c}$ that will make the distance of C fall below d. For the former claim, note that, if we cannot add $\mathbf{v}$ at some point, we

Algorithm 5 Gilbert's Greedy Code Construction

INPUT: n, q, d

OUTPUT: A code $C \subseteq [q]^n$ of distance d

1: $C \leftarrow \emptyset$
2: WHILE there exists a $\mathbf{v} \in [q]^n$ such that $\Delta(\mathbf{v}, \mathbf{c}) \geq d$ for every $\mathbf{c} \in C$ DO
3: Add $\mathbf{v}$ to C
4: RETURN C

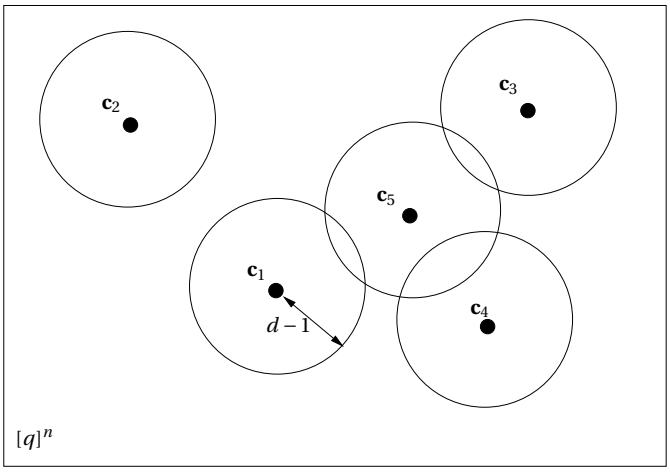


Figure 4.2: An illustration of Gilbert's greedy algorithm (Algorithm 5) for the first five iterations.

cannot add it later. Indeed, since we only add vectors to C, if a vector $\mathbf{v} \in [q]^n$ is ruled out in a certain iteration of Step 2 because $\Delta(\mathbf{c}, \mathbf{v}) < d$, then in all future iterations, we have $\Delta(\mathbf{v}, \mathbf{c}) < d$ and thus, this $\mathbf{v}$ will never be added in Step 3 in any future iteration.

The running time of Algorithm 5 is $q^{O(n)}$. To see this note that Step 2 in the worst-case could be repeated for every vector in $[q]^n$, that is at most q^n times. In a naive implementation, for each iteration, we cycle through all vectors in $[q]^n$ and for each vector $\mathbf{v} \in [q]^n$, iterate through all (at most q^n) vectors $\mathbf{c} \in C$ to check whether $\Delta(\mathbf{c}, \mathbf{v}) < d$. If no such $\mathbf{c}$ exists, then we add $\mathbf{v}$ to C otherwise, we move to the next $\mathbf{v}$. However, note that we can do slightly better– since we know that once a $\mathbf{v}$ is "rejected" in an iteration, it'll keep on being rejected in the future iterations, we can fix up an ordering of vectors in $[q]^n$ and for each vector $\mathbf{v}$ in this order, check whether it can be added to C or not. If so, we add $\mathbf{v}$ to C, else we move to the next vector in the order. This algorithm has time complexity $O(nq^{2n})$, which is still $q^{O(n)}$.

Further, we claim that after termination of Algorithm 5

$$\bigcup_{\mathbf{c} \in C} B(\mathbf{c}, d-1) = [q]^n.$$

This is because if not, then there exists a vector $\mathbf{v} \in [q]^n \setminus C$, such that $\Delta(\mathbf{v}, \mathbf{c}) \geq d$ and hence $\mathbf{v}$ can

be added to C. However, this contradicts the fact that Algorithm 5 has terminated. Therefore,

$$\left|\bigcup_{\mathbf{c}\in C} B(\mathbf{c}, d-1)\right| = q^n. \tag{4.1}$$

It is not too hard to see that

$$\sum_{\mathbf{c}\in C} |B(\mathbf{c}, d-1)| \geq \left|\bigcup_{\mathbf{c}\in C} B(\mathbf{c}, d-1)\right|,$$

which by (4.1) implies that

$$\sum_{\mathbf{c}\in C} |B(\mathbf{c}, d-1)| \geq q^n$$

or since the volume of a Hamming ball is translation invariant,

$$\sum_{\mathbf{c}\in C} Vol_q(d-1, n) \geq q^n.$$

Since $\sum_{\mathbf{c}\in C} Vol_q(d-1,n) = Vol_q(d-1,n)\cdot|C|$, we have

$$\begin{aligned} |C| &\geq \frac{q^n}{Vol_q(d-1,n)} \\ &\geq \frac{q^n}{q^{nH_q(\delta)}} \qquad (4.2) \\ &= q^{n(1-H_q(\delta))}, \end{aligned}$$

as desired. In the above, (4.2) follows from the fact that

$$\begin{aligned} Vol_q(d-1,n) &\leq Vol_q(\delta n, n) \\ &\leq q^{nH_q(\delta)}, \end{aligned} \tag{4.3}$$

where the second inequality follows from the upper bound on the volume of a Hamming ball in Proposition 3.3.1.

It is worth noting that the code from Algorithm 5 is not guaranteed to have any special structure. In particular, even storing the code can take exponential space. We have seen in Proposition 2.3.1 that linear codes have a much more succinct representation. Thus, a natural question is:

Question 4.2.1. *Do linear codes achieve the $R \geq 1 - H_q(\delta)$ tradeoff that the greedy construction achieves?*

Next, we will answer the question in the affirmative.

4.2.2 Linear Code Construction

Now we will show that a random linear code, with high probability, lies on the GV bound. The construction is a use of the probabilistic method (Section 3.2).

By Proposition 2.3.4, we are done if we can show that there exists a $k \times n$ matrix $\mathbf{G}$ of full rank (for $k = (1 - H_q(\delta) - \varepsilon)n$) such that

$$\text{For every } \mathbf{m} \in \mathbb{F}_q^k \setminus \{\mathbf{0}\}, wt(\mathbf{mG}) \geq d.$$

We will prove the existence of such a $\mathbf{G}$ by the probabilistic method. Pick a random linear code by picking a random $k \times n$ matrix $\mathbf{G}$ where each of kn entries is chosen uniformly and independently at random from $\mathbb{F}_q$. Fix $\mathbf{m} \in \mathbb{F}_q^k \setminus \{\mathbf{0}\}$. Recall that by Lemma 3.1.8, for a random $\mathbf{G}$, $\mathbf{mG}$ is a uniformly random vector from $\mathbb{F}_q^n$. Thus, we have

$$\begin{aligned} Pr[wt(\mathbf{mG}) < d] &= \frac{Vol_q(d-1,n)}{q^n} \\ &\leq \frac{q^{nH_q(\delta)}}{q^n}, \end{aligned} \tag{4.4}$$

where (4.4) follows from (4.3). Thus, by the union bound (Lemma 3.1.3)

$$\begin{aligned} Pr[\text{There exists a non-zero } \mathbf{m}, wt(\mathbf{mG}) < d] &\leq q^k q^{-n(1-H_q(\delta))} \\ &= q^{-\varepsilon \cdot n}, \end{aligned}$$

where the equality follows by choosing $k = (1 - H_q(\delta) - \varepsilon)n$. Since $q^{-\varepsilon n} \ll 1$, by the probabilistic method, there exists a linear code C with relative distance δ.

All that's left is to argue that the code C has dimension at least $k = (1 - H_q(\delta) - \varepsilon)n$. To show this we need to show that the chosen generator matrix $\mathbf{G}$ has full rank. Note that there is a non-zero probability that a uniformly matrix $\mathbf{G}$ does not have full rank. There are two ways to deal with this. First, we can show that with high probability a random $\mathbf{G}$ does have full rank, so that $|C| = q^k$. However, the proof above has already shown that, with high probability, the distance is greater than zero, which implies that distinct messages will be mapped to distinct codewords and thus $|C| = q^k$. In other words, C does indeed have dimension k, as desired

Discussion. We now digress a bit to discuss some consequences of the proofs of the GV bound.

We first note the probabilistic method proof shows something stronger than Theorem 4.2.1: *most* linear codes (with appropriate parameters) meet the Gilbert-Varshamov bound.

Note that we can also pick a random linear code by picking a random $(n-k) \times n$ parity check matrix. This also leads to a proof of the GV bound: see Exercise 4.1.

Finally, we note that Theorem 4.2.1 requires $\delta < 1 - \frac{1}{q}$. An inspection of Gilbert and Varshamov's proofs shows that the only reason the proof required that $\delta \leq 1 - \frac{1}{q}$ was because it is needed for the volume bound (recall the bound in Proposition 3.3.1): $Vol_q(\delta n, n) \leq q^{H_q(\delta)n}$– to hold. It is natural to wonder if the above is just an artifact of the proof or, for example,

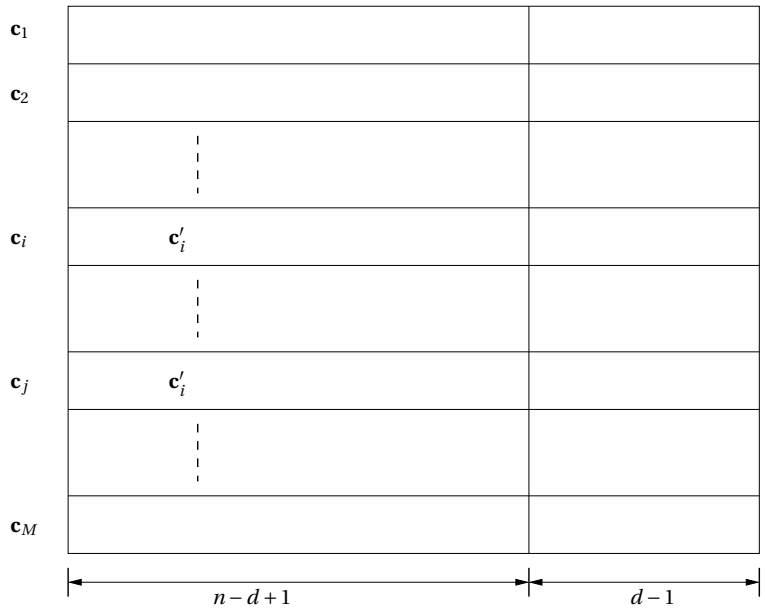


Figure 4.3: Construction of a new code in the proof of the Singleton bound.

Question 4.2.2. *Does there exists a code with $R > 0$ and $\delta > 1 - \frac{1}{q}$?*

We will return to this question in Section 4.4.

4.3 Singleton Bound

We will now change gears again and prove an upper bound on R (for fixed δ). We start by proving the Singleton bound.

Theorem 4.3.1 (Singleton Bound)**.** *For every $(n,k,d)_q$ code,*

$$k \leq n - d + 1.$$

Proof. Let $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_M$ be the codewords of an $(n,k,d)_q$ code C. Note that we need to show $M \leq q^{n-d+1}$. To this end, we define $\mathbf{c}'_i$ to be the prefix of the codeword $\mathbf{c}_i$ of length $n-d+1$ for every $i \in [M]$. See Figure 4.3 for a pictorial description.

We now claim that for every $i \neq j$, $\mathbf{c}'_i \neq \mathbf{c}'_j$. For the sake of contradiction, assume that there exits an $i \neq j$ such that $\mathbf{c}'_i = \mathbf{c}'_j$. Note that this implies that $\mathbf{c}_i$ and $\mathbf{c}_j$ agree in all the first $n-d+1$ positions, which in turn implies that $\Delta(\mathbf{c}_i, \mathbf{c}_j) \leq d-1$. This contradicts the fact that C has distance d. Thus, M is the number of prefixes of codewords in C of length $n-d+1$, which implies that $M \leq q^{n-d+1}$ as desired. □

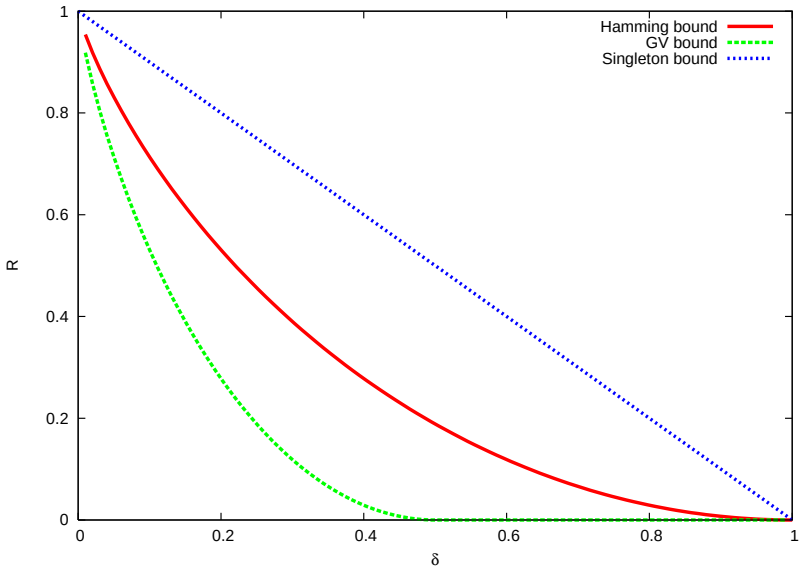


Figure 4.4: The Hamming, GV and Singleton bound for binary codes.

Note that the asymptotic version of the Singleton bound states that $k/n \leq 1 - d/n + 1/n$. In other words,

$$R \leq 1 - \delta + o(1).$$

Figure 4.4 presents a pictorial description of the asymptotic version of the Singleton bound. It is worth noting that the bound is *independent* of the alphabet size. As is evident from Figure 4.4, the Singleton bound is worse than the Hamming bound for binary codes. However, this bound is better for larger alphabet sizes. In fact, we will look at a family of codes called Reed-Solomon codes in Chapter 5 that meets the Singleton bound. However, the alphabet size of the Reed-Solomon codes increases with the block length n. Thus, a natural follow-up question is the following:

Question 4.3.1. *Given a fixed $q \geq 2$, does there exist a q-ary code that meets the Singleton bound?*

We'll see an answer to this question in the next section.

4.4 Plotkin Bound

In this section, we will study the Plotkin bound, which will answer Questions 4.2.2 and 4.3.1. We start by stating the bound.

Theorem 4.4.1 (Plotkin bound)**.** *The following holds for any code $C \subseteq [q]^n$ with distance d:*

1. *If $d = \left(1 - \frac{1}{q}\right) n$, $|C| \leq 2qn$.*
2. *If $d > \left(1 - \frac{1}{q}\right) n$, $|C| \leq \frac{qd}{qd-(q-1)n}$.*

Note that the Plotkin bound (Theorem 4.4.1) implies that a code with relative distance $\delta \geq 1 - \frac{1}{q}$, must necessarily have $R = 0$, which answers Question 4.2.2 in the negative.

Before we prove Theorem 4.4.1, we make couple of remarks. We first note that the upper bound in the first part of Theorem 4.4.1 can be improved to $2n$ for $q = 2$. (See Exercise 4.12.) Second, it can be shown that this bound is tight– see Exercise 4.13. Third, the statement of Theorem 4.4.1 gives a trade-off only for relative distance greater than $1 - 1/q$. However, as the following corollary shows, the result can be extended to work for $0 \leq \delta \leq 1 - 1/q$. (See Figure 4.5 for an illustration for binary codes.)

Corollary 4.4.2. *For any q-ary code with distance δ, $R \leq 1 - \left(\frac{q}{q-1}\right)\delta + o(1)$.*

Proof. The proof proceeds by shortening the codewords. We group the codewords so that they agree on the first $n - n'$ symbols, where $n' = \left\lfloor \frac{qd}{q-1} \right\rfloor - 1$. (We will see later why this choice of n' makes sense.) In particular, for any $\mathbf{x} \in [q]^{n-n'}$, define

$$C_{\mathbf{x}} = \{(c_{n-n'+1}, \ldots c_n) \mid (c_1 \ldots c_N) \in C, (c_1 \ldots c_{n-n'}) = \mathbf{x}\}.$$

Define $d = \delta n$. For all $\mathbf{x}$, $C_{\mathbf{x}}$ has distance d as C has distance d.[1] Additionally, it has block length $n' < (\frac{q}{q-1})d$ and thus, $d > \left(1 - \frac{1}{q}\right) n'$. By Theorem 4.4.1, this implies that

$$|C_{\mathbf{x}}| \leq \frac{qd}{qd - (q-1)n'} \leq qd, \tag{4.5}$$

where the second inequality follows from the fact that $qd - (q-1)n'$ is an integer.

Note that by the definition of $C_{\mathbf{x}}$:

$$|C| = \sum_{\mathbf{x} \in [q]^{n-n'}} |C_{\mathbf{x}}|,$$

which by (4.5) implies that

$$|C| \leq \sum_{\mathbf{x} \in [q]^{n-n'}} qd = q^{n-n'} \cdot qd \leq q^{n - \frac{q}{q-1}d + o(n)} = q^{n\left(1 - \delta \cdot \frac{q}{q-1} + o(1)\right)}.$$

In other words, $R \leq 1 - \left(\frac{q}{q-1}\right)\delta + o(1)$ as desired. □

[1] If for some $\mathbf{x}$, $\mathbf{c}_1 \neq \mathbf{c}_2 \in C_{\mathbf{x}}$, $\Delta(\mathbf{c}_1, \mathbf{c}_2) < d$, then $\Delta((\mathbf{x}, \mathbf{c}_1), (\mathbf{x}, \mathbf{c}_2)) < d$, which implies that the distance of C is less than d (as by definition of $C_{\mathbf{x}}$, both $(\mathbf{x}, \mathbf{c}_1), (\mathbf{x}, \mathbf{c}_2) \in C$).

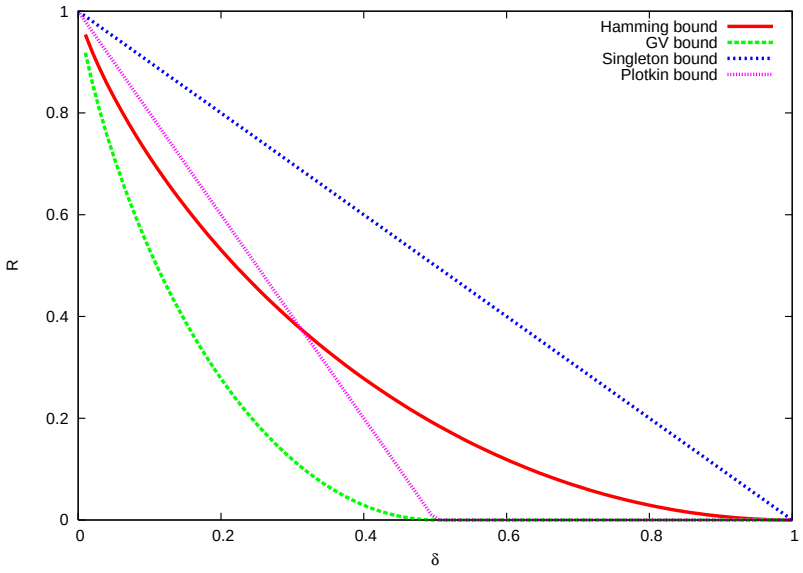


Figure 4.5: The current bounds on the rate R vs. relative distance δ for binary codes. The GV bound is a lower bound on rate while the other three bounds are upper bounds on R.

Note that Corollary 4.4.2 implies that for any q-ary code of rate R and relative distance δ (where q is a *constant* independent of the block length of the code), $R < 1 - \delta$. In other words, this answers Question 4.3.1 in the negative.

Let us pause for a bit at this point and recollect the bounds on R versus δ that we have proved till now. Figure 4.5 depicts all the bounds we have seen till now (for $q = 2$). The GV bound is the best known lower bound at the time of writing of this book. Better upper bounds are known and we will see one such trade-off (called the Elias-Bassalygo bound) in Section 8.1.

Now, we turn to the proof of Theorem 4.4.1, for which we will need two more lemmas.

The first lemma deals with vectors over real spaces. We quickly recap the necessary definitions. Consider a vector $\mathbf{v}$ in $\mathbb{R}^n$, that is, a tuple of n real numbers. This vector has (Euclidean) norm $\|\mathbf{v}\| = \sqrt{v_1^2 + v_2^2 + \ldots + v_n^2}$, and is a unit vector if and only if its norm is 1. The inner product of two vectors, $\mathbf{u}$ and $\mathbf{v}$, is $\langle \mathbf{u}, \mathbf{v} \rangle = \sum_i u_i \cdot v_i$. The following lemma gives a bound on the number of vectors that can exist such that every pair is at an obtuse angle with each other.

Lemma 4.4.3 (Geometric Lemma)**.** *Let $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_m \in \mathbb{R}^N$ be non-zero vectors.*

1. *If $\langle \mathbf{v}_i, \mathbf{v}_j \rangle \leq 0$ for all $i \neq j$, then $m \leq 2N$.*
2. *Let $\mathbf{v}_i$ be unit vectors for $1 \leq i \leq m$. Further, if $\langle \mathbf{v}_i, \mathbf{v}_j \rangle \leq -\varepsilon < 0$ for all $i \neq j$, then $m \leq 1 + \frac{1}{\varepsilon}$.*[2]

(Item 1 is tight: see Exercise 4.14.) The proof of the Plotkin bound will need the existence of a map from codewords to real vectors with certain properties, which the next lemma guarantees.

[2]Note that since $\mathbf{v}_i$ and $\mathbf{v}_j$ are both unit vectors, $\langle \mathbf{v}_i, \mathbf{v}_j \rangle$ is the cosine of the angle between them.

Lemma 4.4.4 (Mapping Lemma)**.** *Let $C \subseteq [q]^n$. Then there exists a function $f : C \longrightarrow \mathbb{R}^{nq}$ such that*

1. *For every $\mathbf{c} \in C$, $\|f(\mathbf{c})\| = 1$.*
2. *For every $\mathbf{c}_1 \neq \mathbf{c}_2$ such that $\mathbf{c}_1, \mathbf{c}_2 \in C$, $\langle f(\mathbf{c}_1), f(\mathbf{c}_2)\rangle = 1 - \left(\frac{q}{q-1}\right)\left(\frac{\Delta(\mathbf{c}_1,\mathbf{c}_2)}{n}\right)$.*

We defer the proofs of the lemmas above to the end of the section. We are now in a position to prove Theorem 4.4.1.

Proof of Theorem 4.4.1 Let $C = \{\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_m\}$. For all $i \neq j$,

$$\left\langle f(\mathbf{c}_i), f(\mathbf{c}_j)\right\rangle \leq 1 - \left(\frac{q}{q-1}\right)\frac{\Delta(\mathbf{c}_i,\mathbf{c}_j)}{n} \leq 1 - \left(\frac{q}{q-1}\right)\frac{d}{n}.$$

The first inequality holds by Lemma 4.4.4, and the second holds as C has distance d.

For part 1, if $d = \left(1 - \frac{1}{q}\right)n = \frac{(q-1)n}{q}$, then for all $i \neq j$,

$$\left\langle f(\mathbf{c}_i), f(\mathbf{c}_j)\right\rangle \leq 0$$

and so by the first part of Lemma 4.4.3, $m \leq 2nq$, as desired.

For part 2, $d > \left(\frac{q-1}{q}\right)n$ and so for all $i \neq j$,

$$\langle f(\mathbf{c}_i), f(\mathbf{c}_j)\rangle \leq 1 - \left(\frac{q}{q-1}\right)\frac{d}{n} = -\left(\frac{qd-(q-1)n}{(q-1)n}\right)$$

and, since $\varepsilon \stackrel{\text{def}}{=} \left(\frac{qd-(q-1)n}{(q-1)n}\right) > 0$, we can apply the second part of Lemma 4.4.3. Thus, $m \leq 1 + \frac{(q-1)n}{qd-(q-1)n} = \frac{qd}{qd-(q-1)n}$, as desired □

4.4.1 Proof of Geometric and Mapping Lemmas

Next, we prove Lemma 4.4.3.

Proof of Lemma 4.4.3. We begin with a proof of the first result. The proof is by induction on n. Note that in the base case of $N = 0$, we have $m = 0$, which satisfies the claimed inequality $m \leq 2N$.

In the general case, we have $m \geq 1$ non-zero vectors $\mathbf{v}_1, \ldots, \mathbf{v}_m \in \mathbb{R}^N$ such that for every $i \neq j$,

$$\langle \mathbf{v}_i, \mathbf{v}_j\rangle \leq 0. \tag{4.6}$$

Since rotating all the vectors by the same amount does not change the sign of the inner product (nor does scaling any of the vectors), w.l.o.g. we can assume that $\mathbf{v}_m = \langle 1, 0, \ldots, 0\rangle$. For $1 \leq i \leq m-1$, denote the vectors as $\mathbf{v}_i = \langle \alpha_i, \mathbf{y}_i\rangle$, for some $\alpha_i \in \mathbb{R}$ and $\mathbf{y}_i \in \mathbb{R}^{N-1}$. Now, for any $i \neq 1$, $\langle \mathbf{v}_1, \mathbf{v}_i\rangle = 1 \cdot \alpha_i + \sum_{i=2}^{m} 0 = \alpha_i$. However, note that (4.6) implies that $\langle \mathbf{v}_1, \mathbf{v}_i\rangle \leq 0$, which in turn implies that

$$\alpha_i \leq 0. \tag{4.7}$$

Next, we claim that at most one of $\mathbf{y}_1, \ldots, \mathbf{y}_{m-1}$ can be the all zeroes vector, $\mathbf{0}$. If not, assume w.l.o.g., that $\mathbf{y}_1 = \mathbf{y}_2 = \mathbf{0}$. This in turn implies that

$$\begin{aligned}\langle \mathbf{v}_1, \mathbf{v}_2 \rangle &= \alpha_1 \cdot \alpha_2 + \langle \mathbf{y}_1, \mathbf{y}_2 \rangle \\ &= \alpha_1 \cdot \alpha_2 + 0 \\ &= \alpha_1 \cdot \alpha_2 \\ &> 0,\end{aligned}$$

where the last inequality follows from the subsequent argument. As $\mathbf{v}_1 = \langle \alpha_1, \mathbf{0} \rangle$ and $\mathbf{v}_2 = \langle \alpha_2, \mathbf{0} \rangle$ are non-zero, this implies that $\alpha_1, \alpha_2 \neq 0$. (4.7) then implies that $\alpha_1, \alpha_2 < 0$. However, $\langle \mathbf{v}_1, \mathbf{v}_2 \rangle > 0$ contradicts (4.6).

Thus, w.l.o.g., assume that $\mathbf{v}_1, \ldots, \mathbf{v}_{m-2}$ are all non-zero vectors. Further, note that for every $i \neq j \in [m-2]$, $\langle \mathbf{y}_i, \mathbf{y}_j \rangle = \langle \mathbf{v}_i, \mathbf{v}_j \rangle - \alpha_i \cdot \alpha_j \leq \langle \mathbf{v}_i, \mathbf{v}_j \rangle \leq 0$. Thus, we have reduced problem on m vectors with dimension N to an equivalent problem on $m-2$ vectors with dimension dimension $N-1$. If we continue this process, we can conclude that every loss in dimension of the vector results in twice in loss in the numbers of the vectors in the set. Induction then implies that $m \leq 2N$, as desired.

We now move on to the proof of the second part. Define $\mathbf{z} = \mathbf{v}_1 + \ldots + \mathbf{v}_m$. Now consider the following sequence of relationships:

$$\|\mathbf{z}\|^2 = \sum_{i=1}^{m} \|\mathbf{v}_i\|^2 + 2 \sum_{i<j} \langle \mathbf{v}_i, \mathbf{v}_j \rangle \leq m + 2 \cdot \binom{m}{2} \cdot (-\varepsilon) = m(1 - \varepsilon m + \varepsilon).$$

The inequality follows from the facts that each $\mathbf{v}_i$ is a unit vector and the assumption that for every $i \neq j$, $\langle \mathbf{v}_i . \mathbf{v}_j \rangle \leq -\varepsilon$. As $\|\mathbf{z}\|^2 \geq 0$,

$$m(1 - \varepsilon m + \varepsilon) \geq 0.$$

Thus, we have $m \leq 1 + \frac{1}{\varepsilon}$, as desired. □

Finally, we prove Lemma 4.4.4.

Proof of Lemma 4.4.4. We begin by defining a map $\phi : [q] \to \mathbb{R}^q$ with certain properties. Then we apply ϕ to all the coordinates of a codeword to define the map $f : \mathbb{R}^q \to \mathbb{R}^{nq}$ that satisfies the claimed properties. We now fill in the details.

Define $\phi : [q] \to \mathbb{R}^q$ as follows. For every $i \in [q]$, we define

$$\phi(i) = \left\langle \frac{1}{q}, \frac{1}{q}, \ldots, \underbrace{\frac{-(q-1)}{q}}_{i^{\text{th}} \text{position}}, \ldots \frac{1}{q} \right\rangle.$$

That is, all but the i'th position in $\phi(i) \in \mathbb{R}^q$ has a value of $1/q$ and the ith position has value $-(q-1)/q$.

Next, we record two properties of ϕ that follow immediately from its definition. For every $i \in [q]$,

$$\phi(i)^2 = \frac{(q-1)}{q^2} + \frac{(q-1)^2}{q^2} = \frac{(q-1)}{q}. \tag{4.8}$$

Also for every $i \neq j \in [q]$,

$$\langle \phi(i), \phi(j) \rangle = \frac{(q-2)}{q^2} - \frac{2(q-1)}{q^2} = -\frac{1}{q}. \tag{4.9}$$

We are now ready to define our final map $f : C \to \mathbb{R}^{nq}$. For every $\mathbf{c} = (c_1, \ldots, c_n) \in C$, define

$$f(\mathbf{c}) = \sqrt{\frac{q}{n(q-1)}} \cdot \big(\phi(c_1), \phi(c_2), \ldots, \phi(c_n)\big).$$

(The multiplicative factor $\sqrt{\frac{q}{n(q-1)}}$ is to ensure that $f(\mathbf{c})$ for any $\mathbf{c} \in C$ is a unit vector.)

To complete the proof, we will show that f satisfies the claimed properties. We begin with condition 1. Note that

$$\|f(\mathbf{c})\|^2 = \frac{q}{(q-1)n} \cdot \sum_{i=1}^{n} \big|\phi(i)\big|^2 = 1,$$

where the first equality follows from the definition of f and the second equality follows from (4.8).

We now turn to the second condition. For notational convenience define $\mathbf{c}_1 = (x_1, \ldots, x_n)$ and $\mathbf{c}_2 = (y_1, \ldots, y_n)$. Consider the following sequence of relations:

$$\begin{aligned}
\big\langle f(\mathbf{c}_1), f(\mathbf{c}_2) \big\rangle &= \sum_{\ell=1}^{n} \big\langle f(x_\ell), f(y_\ell) \big\rangle \\
&= \left[\sum_{\ell : x_\ell \neq y_\ell} \big\langle \phi(x_\ell), \phi(y_\ell) \big\rangle + \sum_{\ell : x_\ell = y_\ell} \big\langle \phi(x_\ell), \phi(y_\ell) \big\rangle \right] \cdot \left(\frac{q}{n(q-1)} \right) \\
&= \left[\sum_{\ell : x_\ell \neq y_\ell} \left(\frac{-1}{q} \right) + \sum_{\ell : x_\ell = y_\ell} \left(\frac{q-1}{q} \right) \right] \cdot \left(\frac{q}{n(q-1)} \right) \qquad (4.10) \\
&= \left[\Delta(\mathbf{c}_1, \mathbf{c}_2) \left(\frac{-1}{q} \right) + (n - \Delta(\mathbf{c}_1, \mathbf{c}_2)) \left(\frac{q-1}{q} \right) \right] \cdot \left(\frac{q}{n(q-1)} \right) \qquad (4.11) \\
&= 1 - \Delta(\mathbf{c}_1, \mathbf{c}_2) \left(\frac{q}{n(q-1)} \right) \left[\frac{1}{q} + \frac{q-1}{q} \right] \\
&= 1 - \left(\frac{q}{q-1} \right) \left(\frac{\Delta(\mathbf{c}_1, \mathbf{c}_2)}{n} \right),
\end{aligned}$$

as desired. In the above, (4.10) is obtained using (4.9) and (4.8) while (4.11) follows from the definition of the Hamming distance. □

4.5 Exercises

Exercise 4.1. Pick a $(n-k) \times n$ matrix H over $\mathbb{F}_q$ at random. Show that with high probability the code whose parity check matrix is H achieves the GV bound.

Exercise 4.2. Recall the definition of an ε-biased space from Exercise 2.14. Show that there exists an ε-biased space of size $O(k/\varepsilon^2)$.

Hint: Recall part 1 of Exercise 2.14.

Exercise 4.3. Argue that a random linear code as well as its dual both lie on the corresponding GV bound.

Exercise 4.4. In Section 4.2.2, we saw that random *linear* code meets the GV bound. It is natural to ask the question for general random codes. (By a random $(n,k)_q$ code, we mean the following: for each of the q^k messages, pick a random vector from $[q]^n$. Further, the choices for each codeword is independent.) We will do so in this problem.

1. Prove that a random q-ary code with rate $R > 0$ with high probability has relative distance $\delta \geq H_q^{-1}(1-2R-\varepsilon)$. Note that this is worse than the bound for random linear codes in Theorem 4.2.1.

2. Prove that with high probability the relative distance of a random q-ary code of rate R is at most $H_q^{-1}(1-2R)+\varepsilon$. In other words, general random codes are worse than random linear codes in terms of their distance.
   Hint: Use Chebyshev's inequality.

Exercise 4.5. We saw that Algorithm 5 can compute an $(n,k)_q$ code on the GV bound in time $q^{O(n)}$. Now the construction for linear codes is a randomized construction and it is natural to ask how quickly can we compute an $[n,k]_q$ code that meets the GV bound. In this problem, we will see that this can also be done in $q^{O(n)}$ deterministic time, though the deterministic algorithm is not that straight-forward anymore.

1. Argue that Theorem 4.2.1 gives a $q^{O(kn)}$ time algorithm that constructs an $[n,k]_q$ code on the GV bound. (Thus, the goal of this problem is to "shave" off a factor of k from the exponent.)

2. A $k \times n$ *Toeplitz Matrix* $A = \{A_{i,j}\}_{i=1,\,j=1}^{k,\,n}$ satisfies the property that $A_{i,j} = A_{i-1,j-1}$. In other words, any diagonal has the same value. For example, the following is a 4×6 Toeplitz matrix:
$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 7 & 1 & 2 & 3 & 4 & 5 \\ 8 & 7 & 1 & 2 & 3 & 4 \\ 9 & 8 & 7 & 1 & 2 & 3 \end{pmatrix}$$
   A *random* $k \times n$ Toeplitz matrix $T \in \mathbb{F}_q^{k \times n}$ is chosen by picking the entries in the first row and column uniformly (and independently) at random.

Prove the following claim: For any non-zero $\mathbf{m} \in \mathbb{F}_q^k$, the vector $\mathbf{m} \cdot T$ is uniformly distributed over $\mathbb{F}_q^n$, that is for every $\mathbf{y} \in \mathbb{F}_q^n$, $\Pr\left[\mathbf{m} \cdot T = \mathbf{y}\right] = q^{-n}$.

3. Briefly argue why the claim in part 2 implies that a random code defined by picking its generator matrix as a random Toeplitz matrix with high probability lies on the GV bound.

4. Conclude that an $[n,k]_q$ code on the GV bound can be constructed in time $q^{O(k+n)}$.

Exercise 4.6. Show that one can construct the parity check matrix of an $[n,k]_q$ code that lies on the GV bound in time $q^{O(n)}$.

Exercise 4.7. So far in Exercises 4.5 and 4.6, we have seen two constructions of $[n,k]_q$ code on the GV bound that can be constructed in $q^{O(n)}$ time. For constant rate codes, at the time of writing of this book, this is fastest known construction of any code that meets the GV bound. For $k = o(n)$, there is a better construction known, which we explore in this exercise.

We begin with some notation. For the rest of the exercise we will target a distance of $d = \delta n$. Given a message $\mathbf{m} \in \mathbb{F}_q^k$ and an $[n,k]_q$ code C, define the indicator variable:

$$W_{\mathbf{m}}(C) = \begin{cases} 1 & \text{if } wt(C(\mathbf{m})) < d \\ 0 & \text{otherwise.} \end{cases}$$

Further, define

$$D(C) = \sum_{\mathbf{m} \in \mathbb{F}_q^k \setminus \{\mathbf{0}\}} W_{\mathbf{m}}(C).$$

We will also use $D(G)$ and $W_{\mathbf{m}}(G)$ to denote the variables above for the code C generated by G.

Given an $k \times n$ matrix M, we will use M^i to denote the ith column of M and $M^{\leq i}$ to denote the column submatrix of M that contains the first i columns. Finally below we will use $\mathscr{G}$ to denote a uniformly random $k \times n$ generator matrix and G to denote a specific instantiation of the generator matrix. We will arrive at the final construction in a sequence of steps. In what follows define $k < (1 - H_q(\delta))n$ for large enough n.

1. Argue that C has a distance d if and only if $D(C) < 1$.

2. Argue that $\mathbb{E}[D(\mathscr{G})] < 1$.

3. Argue that for any $1 \leq i < n$ and fixed $k \times n$ matrix G,

$$\min_{\mathbf{v} \in \mathbb{F}_q^k} \mathbb{E}\left[D(\mathscr{G}) | \mathscr{G}^{\leq i} = G^{\leq i}, \mathscr{G}^{i+1} = \mathbf{v}\right] \leq \mathbb{E}\left[D(\mathscr{G}) | \mathscr{G}^{\leq i} = G^{\leq i}\right].$$

4. We are now ready to define the algorithm to compute the final generator matrix G: see Algorithm 6. Prove that Algorithm 6 outputs a matrix G such that the linear code generated by G is an $[n, k, \delta n]_q$ code. Conclude that this code lies on the GV bound.

Algorithm 6 $q^{O(k)}$ time algorithm to compute a code on the GV bound

INPUT: Integer parameters $1 \le k \ne n$ such that $k < (1 - H_q(\delta)n)$
OUTPUT: An $k \times n$ generator matrix G for a code with distance δn

1: Initialize G to be the all 0s matrix ▷ This initialization is arbitrary
2: FOR every $1 \le i \le n$ DO
3: $\quad G^i \leftarrow \arg\min_{\mathbf{v} \in \mathbb{F}_q^k} \mathbb{E}\left[D(\mathscr{G}) | \mathscr{G}^{\le i} = G^{\le i}, \mathscr{G}^{i+1} = \mathbf{v}\right]$
4: RETURN G

5. Finally, we will analyze the run time of Algorithm 6. Argue that Step 2 can be implemented in $\text{poly}\left(n, q^k\right)$ time. Conclude Algorithm 6 can be implemented in time $\text{poly}\left(n, q^k\right)$.

   Hint: It might be useful to maintain a data structure that keeps track of one number for every non-zero $\mathbf{m} \in \mathbb{F}_q^k$ throughout the run of Algorithm 6.

Exercise 4.8. In this problem we will derive the GV bound using a graph-theoretic proof, which is actually equivalent to the greedy proof we saw in Section 4.2.1. Let $1 \le d \le n$ and $q \ge 1$ be integers. Now consider the graph $G_{n,d,q} = (V, E)$, where the vertex set is the set of all vectors in $[q]^n$. Given two vertices $\mathbf{u} \ne \mathbf{v} \in [q]^n$, we have the edge $(u, v) \in E$ if and only if $\Delta(\mathbf{u}, \mathbf{v}) < d$. An *independent set* of a graph $G = (V, E)$ is a subset $I \subseteq V$ such that for every $u \ne v \in I$, we have that (u, v) is *not* an edge. We now consider the following sub-problems:

1. Argue that any independent set C of $G_{n,d,q}$ is a q-ary code of distance d.
2. The *degree* of a vertex in a graph G is the number of edges incident on that vertex. Let Δ be the maximum degree of any vertex in $G = (V, E)$.Then argue that G has an independent set of size at least $\frac{|V|}{\Delta+1}$.
3. Using parts 1 and 2 argue the GV bound.

Exercise 4.9. In this problem we will improve slightly on the GV bound using a more sophisticated graph-theoretic proof. Let $G_{n,d,q}$ and N and Δ be as in the previous exercise (Exercise 4.8). So far we used the fact that $G_{n,d,q}$ has many vertices and small degree to prove it has a large independent set, and thus to prove there is a large code of minimum distance d. In this exercise we will see how a better result can be obtained by counting the number of "triangles" in the graph. A triangle in a graph $G = (V, E)$ is a set $\{u, v, w\} \subset V$ of three vertices such that all three vertices are adjacent, i.e., $(u, v), (v, w), (w, u) \in E$. For simplicity we will focus on the case where $q = 2$ and $d = n/5$, and consider the limit as $n \to \infty$.

1. Prove that a graph on N vertices of maximum degree Δ has at most $O(N\Delta^2)$ triangles.
2. Prove that the number of triangle in graph $G_{n,d,2}$ is at most

$$2^n \cdot \sum_{0 \le e \le 3d/2} \binom{n}{e} \cdot 3^e.$$

Hint: Fix u and let e count the number of coordinates where at least one of v or w disagree with u. Prove that e is at most $3d/2$.

3. Simplify the expression in the case where $d = n/5$ to show that the number of triangles in $G_{n,n/5,2}$ is $O(N \cdot \Delta^{2-\eta})$ for some $\eta > 0$.

4. A famous result in the "probabilistic method" shows (and you don't have to prove this), that if a graph on N vertices of maximum degree Δ has at most $O(N \cdot \Delta^{2-\eta})$ triangles, then it has an independent set of size $\Omega(\frac{N}{\Delta}\log\Delta)$. Use this result to conclude that there is a binary code of block length n and distance $n/5$ of size $\Omega(n2^n/\binom{n}{n/5})$. (Note that this improves over the GV-bound by an $\Omega(n)$ factor.)

Exercise 4.10. Use part 2 from Exercise 1.7 to prove the Singleton bound.

Exercise 4.11. Let C be an $(n,k,d)_q$ code. Then prove that fixing any $n-d+1$ positions uniquely determines the corresponding codeword.

Exercise 4.12. Let C be a binary code of block length n and distance $n/2$. Then $|C| \leq 2n$. (Note that this is a factor 2 better than part 1 in Theorem 4.4.1.)

Exercise 4.13. Prove that the bound in Exercise 4.12 is tight– i.e. there exists binary codes C with block length n and distance $n/2$ such that $|C| = 2n$.

Exercise 4.14. Prove that part 1 of Lemma 4.4.3 is tight.

Exercise 4.15. In this exercise we will prove the Plotkin bound (at least part 2 of Theorem 4.4.1) via a purely combinatorial proof.

Given an $(n,k,d)_q$ code C with $d > \left(1-\frac{1}{q}\right)n$ define

$$S = \sum_{\mathbf{c}_1 \neq \mathbf{c}_2 \in C} \Delta(\mathbf{c}_1, \mathbf{c}_2).$$

For the rest of the problem think of C has an $|C| \times n$ matrix where each row corresponds to a codeword in C. Now consider the following:

1. Looking at the contribution of each column in the matrix above, argue that

$$S \leq \left(1-\frac{1}{q}\right) \cdot n|C|^2.$$

2. Look at the contribution of the rows in the matrix above, argue that

$$S \geq |C|\,(|C|-1) \cdot d.$$

3. Conclude part 2 of Theorem 4.4.1.

Exercise 4.16. In this exercise, we will prove the so called *Griesmer Bound*. For any $[n,k,d]_q$, prove that

$$n \geq \sum_{i=0}^{k-1} \left\lceil \frac{d}{q^i} \right\rceil.$$

Hint: Recall Exercise 2.17.

Exercise 4.17. Use Exercise 4.16 to prove part 2 of Theorem 4.4.1 for linear codes.

Exercise 4.18. Use Exercise 4.16 to prove Theorem 4.3.1 for linear code.

4.6 Bibliographic Notes

Theorem 4.2.1 was proved for general codes by Edgar Gilbert ([24]) and for linear codes by Rom Varshamov ([76]). Hence, the bound is called the Gilbert-Varshamov bound. The Singleton bound (Theorem 4.3.1) is due to Richard C. Singleton [67]. For larger (but still constant) values of q, better lower bounds than the GV bound are known. In particular, for any prime power $q \geq 49$, there exist linear codes, called *algebraic geometric* (or AG) codes that outperform the corresponding GV bound[3]. AG codes out of the scope of this book. One starting point could be the following [42].

The proof method illustrated in Exercise 4.15has a name– *double counting*: in this specific case this follows since we count S in two different ways.

[3]The lower bound of 49 comes about as AG codes are only defined for q being a square (i.e. $q = (q')^2$) and it turns out that $q' = 7$ is the smallest value where AG bound beats the GV bound.

Chapter 5

The Greatest Code of Them All: Reed-Solomon Codes

In this chapter, we will study the Reed-Solomon codes. Reed-Solomon codes have been studied a lot in coding theory. These codes are optimal in the sense that they meet the Singleton bound (Theorem 4.3.1). We would like to emphasize that these codes meet the Singleton bound not just asymptotically in terms of rate and relative distance but also in terms of the dimension, block length and distance. As if this were not enough, Reed-Solomon codes turn out to be more versatile: they have many applications outside of coding theory. (We will see some applications later in the book.)

These codes are defined in terms of univariate polynomials (i.e. polynomials in one unknown/variable) with coefficients from a finite field $\mathbb{F}_q$. It turns out that polynomials over $\mathbb{F}_p$, for prime p, also help us define finite fields $\mathbb{F}_{p^s}$, for $s > 1$. To kill two birds with one stone[1], we first do a quick review of polynomials over finite fields. Then we will define and study some properties of Reed-Solomon codes.

5.1 Polynomials and Finite Fields

We begin with the formal definition of a (univariate) polynomial.

Definition 5.1.1. Let $\mathbb{F}_q$ be a finite field with q elements. Then a function $F(X) = \sum_{i=0}^{\infty} f_i X^i, f_i \in \mathbb{F}_q$ is called a polynomial.

For our purposes, we will only consider the finite case; that is, $F(X) = \sum_{i=0}^{d} f_i X^i$ for some integer $d > 0$, with coefficients $f_i \in \mathbb{F}_q$, and $f_d \neq 0$. For example, $2X^3 + X^2 + 5X + 6$ is a polynomial over $\mathbb{F}_7$.

Next, we define some useful notions related to polynomials. We begin with the notion of degree of a polynomial.

[1] No birds will be harmed in this exercise.

Definition 5.1.2. For $F(X) = \sum_{i=0}^{d} f_i X^i$ ($f_d \neq 0$), we call d the *degree* of $F(X)$. We denote the degree of the polynomial $F(X)$ by $\deg(F)$.

For example, $2X^3 + X^2 + 5X + 6$ has degree 3.

Let $\mathbb{F}_q[X]$ be the set of polynomials over $\mathbb{F}_q$, that is, with coefficients from $\mathbb{F}_q$. Let $F(X), G(X) \in \mathbb{F}_q[X]$ be polynomials. Then $\mathbb{F}_q[X]$ has the following natural operations defined on it:

Addition:

$$F(X) + G(X) = \sum_{i=0}^{\max(\deg(F),\deg(G))} (f_i + g_i) X^i,$$

where the addition on the coefficients is done over $\mathbb{F}_q$. For example, over $\mathbb{F}_2$, $X + (1 + X) = X \cdot (1 + 1) + 1 \cdot (0 + 1)1 = 1$ (recall that over $\mathbb{F}_2$, $1 + 1 = 0$).[2]

Multiplication:

$$F(X) \cdot G(X) = \sum_{i=0}^{\deg(F)+\deg(G)} \left(\sum_{j=0}^{\min(i,\deg(F))} p_j \cdot q_{i-j} \right) X^i,$$

where all the operations on the coefficients are over $\mathbb{F}_q$. For example, over $\mathbb{F}_2$, $X(1 + X) = X + X^2$; $(1 + X)^2 = 1 + 2X + X^2 = 1 + X^2$, where the latter equality follows since $2 \equiv 0 \bmod 2$.

Next, we define the notion of a root of a polynomial.

Definition 5.1.3. $\alpha \in \mathbb{F}_q$ is a root of a polynomial $F(X)$ if $F(\alpha) = 0$.

For instance, 1 is a root of $1 + X^2$ over $\mathbb{F}_2$.

We will also need the notion of a special class of polynomials, which are like prime numbers for polynomials.

Definition 5.1.4. A polynomial $F(X)$ is irreducible if for every $G_1(X), G_2(X)$ such that $F(X) = G_1(X)G_2(X)$, we have $\min(\deg(G_1), \deg(G_2)) = 0$

For example, $1 + X^2$ is not irreducible over $\mathbb{F}_2$, as $(1 + X)(1 + X) = 1 + X^2$. However, $1 + X + X^2$ is irreducible, since its non-trivial factors have to be from the linear terms X or $X + 1$. However, it is easy to check that neither is a factor of $1 + X + X^2$. (In fact, one can show that $1 + X + X^2$ is the only irreducible polynomial of degree 2 over $\mathbb{F}_2$– see Exercise 5.1.) A word of caution: if a polynomial $E(X) \in \mathbb{F}_q[X]$ does not have any root in $\mathbb{F}_q$, it does *not* mean that $E(X)$ is irreducible. For example consider the polynomial $(1 + X + X^2)^2$ over $\mathbb{F}_2$– it does not have any root in $\mathbb{F}_2$ but it obviously is not irreducible.

Just as the set of integers modulo a prime is a field, so is the set of polynomials modulo an irreducible polynomial:

Theorem 5.1.1. *Let $E(X)$ be an irreducible polynomial with degree ≥ 2 over $\mathbb{F}_p$, p prime. Then the set of polynomials in $\mathbb{F}_p[X]$ modulo $E(X)$, denoted by $\mathbb{F}_p[X]/E(X)$, is a field.*

[2] This will be a good time to remember that operations over a finite field are much different from operations over integers/reals. For example, over reals/integers $X + (X + 1) = 2X + 1$.

The proof of the theorem above is similar to the proof of Lemma 2.1.2, so we only sketch the proof here. In particular, we will explicitly state the basic tenets of $\mathbb{F}_p[X]/E(X)$.

- Elements are polynomials in $\mathbb{F}_p[X]$ of degree at most $s-1$. Note that there are p^s such polynomials.
- Addition: $(F(X)+G(X)) \mod E(X) = F(X) \mod E(X) + G(X) \mod E(X) = F(X)+G(X)$. (Since $F(X)$ and $G(X)$ are of degree at most $s-1$, addition modulo $E(X)$ is just plain simple polynomial addition.)
- Multiplication: $(F(X)\cdot G(X)) \mod E(X)$ is the unique polynomial $R(X)$ with degree at most $s-1$ such that for some $A(X)$, $R(X)+A(X)E(X) = F(X)\cdot G(X)$
- The additive identity is the zero polynomial, and the additive inverse of any element $F(X)$ is $-F(X)$.
- The multiplicative identity is the constant polynomial 1. It can be shown that for every element $F(X)$, there exists a unique multiplicative inverse $(F(X))^{-1}$.

For example, for $p=2$ and $E(X) = 1+X+X^2$, $\mathbb{F}_2[X]/(1+X+X^2)$ has as its elements $\{0,1,X,1+X\}$. The additive inverse of any element in $\mathbb{F}_2[X]/(1+X+X^2)$ is the element itself while the multiplicative inverses of $1, X$ and $1+X$ are $1, 1+X$ and X respectively.

A natural question to ask is if irreducible polynomials exist. Indeed, they do for every degree:

Theorem 5.1.2. *For all $s \geq 2$ and $\mathbb{F}_p$, there exists an irreducible polynomial of degree s over $\mathbb{F}_p$. In fact, the number of such irreducible polynomials is $\Theta\left(\frac{p^s}{s}\right)$.*[3]

Given any monic [4] polynomial $E(X)$ of degree s, it can be verified whether it is an irreducible polynomial by checking if $\gcd(E(X), X^{q^s} - X) = E(X)$. This is true as the product of all monic irreducible polynomials in $\mathbb{F}_q[X]$ of degree exactly s is known to be the polynomial $X^{q^s} - X$. Since Euclid's algorithm for computing the $\gcd(F(X), G(X))$ can be implemented in time polynomial in the minimum of $\deg(F)$ and $\deg(G)$ and $\log q$ (see Section **??**), this implies that checking whether a given polynomial of degree s over $\mathbb{F}_q[X]$ is irreducible can be done in time $\text{poly}(s, \log q)$.

This implies an efficient Las Vegas algorithm[5] to generate an irreducible polynomial of degree s over $\mathbb{F}_q$. Note that the algorithm is to keep on generating random polynomials until it comes across an irreducible polynomial (Theorem 5.1.2 implies that the algorithm will check $O(s)$ polynomials in expectation). Algorithm 7 presents the formal algorithm.

The above discussion implies the following:

[3] The result is true even for general finite fields $\mathbb{F}_q$ and not just prime fields but we stated the version over prime fields for simplicity.

[4] I.e. the coefficient of the highest degree term is 1. It is easy to check that if $E(X) = e_s X^s + e_{s-1}X^{s-1} + \cdots + 1$ is irreducible, then $e_s^{-1} \cdot E(X)$ is also an irreducible polynomial.

[5] A Las Vegas algorithm is a randomized algorithm which always succeeds and we consider its time complexity to be its expected worst-case run time.

Algorithm 7 Generating Irreducible Polynomial

INPUT: Prime power q and an integer $s > 1$
OUTPUT: A monic irreducible polynomial of degree s over $\mathbb{F}_q$

1: $b \leftarrow 0$
2: WHILE $b = 0$ DO
3: $F(X) \leftarrow X^s + \sum_{i=0}^{s-1} f_i X^i$, where each $f_i$is chosen uniformly at random from $\mathbb{F}_q$.
4: IF $\gcd(F(X), X^{q^s} - X) = F(X)$ THEN
5: $b \leftarrow 1$.
6: RETURN $F(X)$

Corollary 5.1.3. *There is a Las Vegas algorithm to generate an irreducible polynomial of degree s over any $\mathbb{F}_q$ in expected time* $\text{poly}(s, \log q)$.

Now recall that Theorem 2.1.3 states that for every prime power p^s, there a unique field $\mathbb{F}_{p^s}$. This along with Theorems 5.1.1 and 5.1.2 imply that:

Corollary 5.1.4. *The field $\mathbb{F}_{p^s}$ is $\mathbb{F}_p[X]/E(X)$, where $E(X)$ is an irreducible polynomial of degree s.*

5.2 Reed-Solomon Codes

Recall that the Singleton bound (Theorem 4.3.1) states that for any $(n, k, d)_q$ code, $k \leq n - d + 1$. Next, we will study Reed-Solomon codes, which meet the Singleton bound, i.e. satisfy $k = n - d + 1$ (but have the unfortunate property that $q \geq n$). Note that this implies that the Singleton bound is tight, at least for $q \geq n$.

We begin with the definition of Reed-Solomon codes.

Definition 5.2.1 (Reed-Solomon code)**.** Let $\mathbb{F}_q$ be a finite field. Let $\alpha_1, \alpha_2, ...\alpha_n$ be distinct elements (also called *evaluation points*) from $\mathbb{F}_q$ and choose n and k such that $k \leq n \leq q$. We define an encoding function for Reed-Solomon code as $RS : \mathbb{F}_q^k \to \mathbb{F}_q^n$ as follows. A message $\mathbf{m} = (m_0, m_1, ..., m_{k-1})$ with $m_i \in \mathbb{F}_q$ is mapped to a degree $k-1$ polynomial.

$$\mathbf{m} \mapsto f_{\mathbf{m}}(X),$$

where

$$f_{\mathbf{m}}(X) = \sum_{i=0}^{k-1} m_i X^i. \tag{5.1}$$

Note that $f_{\mathbf{m}}(X) \in \mathbb{F}_q[X]$ is a polynomial of degree at most $k-1$. The encoding of $\mathbf{m}$ is the evaluation of $f_{\mathbf{m}}(X)$ at all the α_i's :

$$RS(\mathbf{m}) = \left(f_{\mathbf{m}}(\alpha_1), f_{\mathbf{m}}(\alpha_2), ..., f_{\mathbf{m}}(\alpha_n)\right).$$

We call this image Reed-Solomon code or RS code. A common special case is $n = q - 1$ with the set of evaluation points being $\mathbb{F}^* \stackrel{\text{def}}{=} \mathbb{F} \setminus \{0\}$.

For example, the first row below are all the codewords in the $[3,2]_3$ Reed-Solomon codes where the evaluation points are $\mathbb{F}_3$ (and the codewords are ordered by the corresponding messages from $\mathbb{F}_3^2$ in lexicographic order where for clarity the second row shows the polynomial $f_{\mathbf{m}}(X)$ for the corresponding $\mathbf{m} \in \mathbb{F}_3^2$):

(0,0,0),	(1,1,1),	(2,2,2),	(0,1,2),	(1,2,0),	(2,0,1),	(0,2,1),	(1,0,2),	(2,1,0)
0,	1,	2,	X,	X+1,	X+2,	2X,	2X+1,	2X+2

Notice that by definition, the entries in $\{\alpha_1, ..., \alpha_n\}$ are distinct and thus, must have $n \leq q$.

We now turn to some properties of Reed-Solomon codes.

Claim 5.2.1. *RS codes are linear codes.*

Proof. The proof follows from the fact that if $a \in \mathbb{F}_q$ and $f(X), g(X) \in \mathbb{F}_q[X]$ are polynomials of degree $\leq k-1$, then $af(X)$ and $f(X)+g(X)$ are also polynomials of degree $\leq k-1$. In particular, let messages $\mathbf{m}_1$ and $\mathbf{m}_2$ be mapped to $f_{\mathbf{m}_1}(X)$ and $f_{\mathbf{m}_2}(X)$ where $f_{\mathbf{m}_1}(X), f_{\mathbf{m}_2}(X) \in \mathbb{F}_q[X]$ are polynomials of degree at most $k-1$ and because of the mapping defined in (5.1), it is easy to verify that:

$$f_{\mathbf{m}_1}(X) + f_{\mathbf{m}_2}(X) = f_{\mathbf{m}_1+\mathbf{m}_2}(X),$$

and

$$a f_{\mathbf{m}_1}(X) = f_{a\mathbf{m}_1}(X).$$

In other words,

$$RS(\mathbf{m}_1) + RS(\mathbf{m}_2) = RS(\mathbf{m}_1 + \mathbf{m}_2)$$

$$aRS(\mathbf{m}_1) = RS(a\mathbf{m}_1).$$

Therefore RS is a $[n,k]_q$ linear code. □

The second and more interesting claim is the following:

Claim 5.2.2. *RS is a $[n,k,n-k+1]_q$ code. That is, it matches the Singleton bound.*

The claim on the distance follows from the fact that every non-zero polynomial of degree $k-1$ over $\mathbb{F}_q[X]$ has at most $k-1$ (not necessarily distinct) roots, and that if two polynomials agree on more than $k-1$ places then they must be the same polynomial.

Proposition 5.2.3 ("Degree Mantra")**.** *A nonzero polynomial $f(X)$ of degree t over a field $\mathbb{F}_q$ has at most t roots in $\mathbb{F}_q$*

Proof. We will prove the theorem by induction on t. If $t = 0$, we are done. Now, consider $f(X)$ of degree $t > 0$. Let $\alpha \in \mathbb{F}_q$ be a root such that $f(\alpha) = 0$. If no such root α exists, we are done. If there is a root α, then we can write

$$f(X) = (X - \alpha)g(X)$$

where $\deg(g) = \deg(f) - 1$ (i.e. $X - \alpha$ divides $f(X)$). Note that $g(X)$ is non-zero since $f(X)$ is non-zero. This is because by the fundamental rule of division of polynomials:

$$f(X) = (X - \alpha)g(X) + R(X)$$

where $\deg(R) \leq 0$ (as the degree cannot be negative this in turn implies that $\deg(R) = 0$) and since $f(\alpha) = 0$,

$$f(\alpha) = 0 + R(\alpha),$$

which implies that $R(\alpha) = 0$. Since $R(X)$ has degree zero (i.e. it is a constant polynomial), this implies that $R(X) \equiv 0$.

Finally, as $g(X)$ is non-zero and has degree $t-1$, by induction, $g(X)$ has at most $t-1$ roots, which implies that $f(X)$ has at most t roots. □

We are now ready to prove Claim 5.2.2

Proof of Claim 5.2.2. We start by proving the claim on the distance. Fix arbitrary $\mathbf{m}_1 \neq \mathbf{m}_2 \in \mathbb{F}_q^k$. Note that $f_{\mathbf{m}_1}(X), f_{\mathbf{m}_2}(X) \in \mathbb{F}_q[X]$ are distinct polynomials of degree at most $k-1$ since $\mathbf{m}_1 \neq \mathbf{m}_2 \in \mathbb{F}_q^k$. Then $f_{\mathbf{m}_1}(X) - f_{\mathbf{m}_2}(X) \neq 0$ also has degree at most $k-1$. Note that $wt(RS(\mathbf{m}_2) - RS(\mathbf{m}_1)) = \Delta(RS(\mathbf{m}_1), RS(\mathbf{m}_2))$. The weight of $RS(\mathbf{m}_2) - RS(\mathbf{m}_1)$ is n minus the number of zeroes in $RS(\mathbf{m}_2) - RS(\mathbf{m}_1)$, which is equal to n minus the number of roots that $f_{\mathbf{m}_1}(X) - f_{\mathbf{m}_2}(X)$ has among $\{\alpha_1, ..., \alpha_n\}$. That is,

$$\Delta(RS(\mathbf{m}_1), RS(\mathbf{m}_2)) = n - |\{\alpha_i \mid f_{\mathbf{m}_1}(\alpha_i) = f_{\mathbf{m}_2}(\alpha_i)\}|.$$

By Proposition 5.2.3, $f_{\mathbf{m}_1}(X) - f_{\mathbf{m}_2}(X)$ has at most $k-1$ roots. Thus, the weight of $RS(\mathbf{m}_2) - RS(\mathbf{m}_1)$ is at least $n-(k-1) = n-k+1$. Therefore $d \geq n-k+1$, and since the Singleton bound (Theorem 4.3.1) implies that $d \leq n-k+1$, we have $d = n-k+1$.[6] The argument above also shows that distinct polynomials $f_{\mathbf{m}_1}(X), f_{\mathbf{m}_2}(X) \in \mathbb{F}_q[X]$ are mapped to distinct codewords. (This is because the Hamming distance between any two codewords is at least $n-k+1 \geq 1$, where the last inequality follows as $k \leq n$.) Therefore, the code contains q^k codewords and has dimension k. The claim on linearity of the code follows from Claim 5.2.1. □

Recall that the Plotkin bound (Corollary 4.4.2) implies that to achieve the Singleton bound, the alphabet size cannot be a constant. Thus, some dependence of q on n in Reed-Solomon codes is unavoidable.

Let us now find a generator matrix for RS codes (which exists by Claim 5.2.1). By Definition 5.2.1, any basis $f_{\mathbf{m}_1}, ..., f_{\mathbf{m}_k}$ of polynomial of degree at most $k-1$ gives rise to a basis $RS(\mathbf{m}_1), ..., RS(\mathbf{m}_k)$ of the code. A particularly nice polynomial basis is the set of monomials $1, X, ..., X^i, ..., X^{k-1}$. The corresponding generator matrix, whose ith row (numbering rows from 0 to $k-1$) is

$$(\alpha_1^i, \alpha_2^i, ..., \alpha_j^i, ..., \alpha_n^i)$$

and this generator matrix is called the *Vandermonde* matrix of size $k \times n$:

[6] See Exercise 5.3 for an alternate direct argument.

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ \alpha_1 & \alpha_2 & \cdots & \alpha_j & \cdots & \alpha_n \\ \alpha_1^2 & \alpha_2^2 & \cdots & \alpha_j^2 & \cdots & \alpha_n^2 \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \alpha_1^i & \alpha_2^i & \cdots & \alpha_j^i & \cdots & \alpha_n^i \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \alpha_1^{k-1} & \alpha_2^{k-1} & \cdots & \alpha_j^{k-1} & \cdots & \alpha_n^{k-1} \end{pmatrix}$$

The class of codes that match the Singleton bound have their own name, which we define and study next.

5.3 A Property of MDS Codes

Definition 5.3.1 (MDS codes)**.** An $(n,k,d)_q$ code is called *Maximum Distance Separable (MDS)* if $d = n-k+1$.

Thus, Reed-Solomon codes are MDS codes.

Next, we prove an interesting property of an MDS code $C \subseteq \Sigma^n$ with integral dimension k. We begin with the following notation.

Definition 5.3.2. For any subset of indices $S \subseteq [n]$ of size exactly k and a code $C \subseteq \Sigma^n$, C_S is the set of all codewords in C projected onto the indices in S.

MDS codes have the following nice property that we shall prove for the special case of Reed-Solomon codes first and subsequently for the general case as well.

Proposition 5.3.1. *Let $C \subseteq \Sigma^n$ of integral dimension k be an MDS code, then for all $S \subseteq [n]$ such that $|S| = k$, we have $|C_S| = \Sigma^k$.*

Before proving Proposition 5.3.1 in its full generality, we present its proof for the special case of Reed-Solomon codes.
Consider any $S \subseteq [n]$ of size k and fix an arbitrary $\mathbf{v} = (v_1, \ldots, v_k) \in \mathbb{F}_q^k$, we need to show that there exists a codeword $\mathbf{c} \in RS$ (assume that the RS code evaluates polynomials of degree at most $k-1$ over $\alpha_1, \ldots, \alpha_n \subseteq \mathbb{F}_q$) such that $\mathbf{c}_S = \mathbf{v}$. Consider a generic degree $k-1$ polynomial $F(X) = \sum_{i=0}^{k-1} f_i X^i$. Thus, we need to show that there exists $F(X)$ such that $F(\alpha_i) = v_i$ for all $i \in S$, where $|S| = k$.

For notational simplicity, assume that $S = [k]$. We think of f_i's as unknowns in the equations that arise out of the relations $F(\alpha_i) = v_i$. Thus, we need to show that there is a solution to the following system of linear equations:

$$\begin{pmatrix} p_0 & p_1 & \cdots & p_{k-1} \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 \\ \alpha_1 & \alpha_i & \alpha_k \\ \alpha_1^2 & \alpha_i^2 & \alpha_k^2 \\ \vdots & \vdots & \vdots \\ \alpha_1^{k-1} & \alpha_i^{k-1} & \alpha_k^{k-1} \end{pmatrix} = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \\ v_k \end{pmatrix}$$

The above constraint matrix is a Vandermonde matrix and is known to have full rank (see Exercise 5.7). Hence, by Exercise 2.6, there always exists a unique solution for $(p_0, \ldots, p_{k-1})$. This completes the proof for Reed-Solomon codes.

Next, we prove the property for the general case which is presented below

Proof of Proposition 5.3.1. Consider a $|C| \times n$ matrix where each row represents a codeword in C. Hence, there are $|C| = |\Sigma|^k$ rows in the matrix. The number of columns is equal to the block length n of the code. Since C is Maximum Distance Separable, its distance $d = n - k + 1$.

Let $S \subseteq [n]$ be of size exactly k. It is easy to see that for any $\mathbf{c}^i \neq \mathbf{c}^j \in C$, the corresponding projections $\mathbf{c}^i_S$ and $\mathbf{c}^j_S \in C_S$ are not the same. As otherwise $\triangle(\mathbf{c}^i, \mathbf{c}^j) \leq d - 1$, which is not possible as the minimum distance of the code C is d. Therefore, every codeword in C gets mapped to a distinct codeword in C_S. As a result, $|C_S| = |C| = |\Sigma|^k$. As $C_S \subseteq \Sigma^k$, this implies that $C_S = \Sigma^k$, as desired. □

Proposition 5.3.1 implies an important property in pseudorandomness: see Exercise 5.8 for more.

5.4 Exercises

Exercise 5.1. Prove that $X^2 + X + 1$ is the unique irreducible polynomial of degree two over $\mathbb{F}_2$.

Exercise 5.2. Argue that any function $f : \mathbb{F}_q \to \mathbb{F}_q$ is equivalent to a polynomial $P(X) \in \mathbb{F}_q[X]$ of degree at most $q - 1$: that is, for every $\alpha \in \mathbb{F}_q$

$$f(\alpha) = P(\alpha).$$

Exercise 5.3. For any $[n, k]_q$ Reed-Solomon code, exhibit two codewords that are at Hamming distance exactly $n - k + 1$.

Exercise 5.4. Let $\mathrm{RS}_{\mathbb{F}_q^*}[n, k]$ denote the Reed-Solomon code over $\mathbb{F}_q$ where the evaluation points is $\mathbb{F}_q$ (i.e. $n = q$). Prove that

$$\left(\mathrm{RS}_{\mathbb{F}_q}[n, k]\right)^{\perp} = \mathrm{RS}_{\mathbb{F}_q}[n, n - k],$$

that is, the dual of these Reed-Solomon codes are Reed-Solomon codes themselves. Conclude that Reed-Solomon codes contain self-dual codes (see Exercise 2.31 for a definition).

Hint: Exercise 2.2 might be useful.

Exercise 5.5. Since Reed-Solomon codes are linear codes, by Proposition 2.3.3, one can do error detection for Reed-Solomon codes in quadratic time. In this problem, we will see that one can design even more efficient error detection algorithm for Reed-Solomon codes. In particular, we will consider *data streaming algorithms* (see Section 17.5 for more motivation on this class of algorithms). A data stream algorithm makes a sequential pass on the input, uses poly-logarithmic space and spend only poly-logarithmic time on each location in the input. In this problem we show that there exists a randomized data stream algorithm to solve the error detection problem for Reed-Solomon codes.

1. Give a randomized data stream algorithm that given as input $\mathbf{y} \in \mathbb{F}_q^m$ decides whether $\mathbf{y} = \mathbf{0}$ with probability at least 2/3. Your algorithm should use $O(\log qm)$ space and polylog(qm) time per position of $\mathbf{y}$. For simplicity, you can assume that given an integer $t \geq 1$ and prime power q, the algorithm has oracle access to an irreducible polynomial of degree t over $\mathbb{F}_q$.
   Hint: Use Reed-Solomon codes.

2. Given $[q,k]_q$ Reed-Solomon code C (i.e. with the evaluation points being $\mathbb{F}_q$), present a data stream algorithm for error detection of C with $O(\log q)$ space and polylog$q$ time per position of the received word. The algorithm should work correctly with probability at least 2/3. You should assume that the data stream algorithm has access to the values of k and q (and knows that C has $\mathbb{F}_q$ as its evaluation points).
   Hint: Part 1 and Exercise 5.4 should be helpful.

Exercise 5.6. We have defined Reed-Solomon in this chapter and Hadamard codes in Section 2.7. In this problem we will prove that certain alternate definitions also suffice.

1. Consider the Reed-Solomon code over a field $\mathbb{F}_q$ and block length $n = q-1$ defined as

$$\mathrm{RS}_{\mathbb{F}_q^*}[n,k,n-k+1] = \{(p(1), p(\alpha), \ldots, p(\alpha^{n-1})) \mid p(X) \in \mathbb{F}[X] \text{ has degree} \leq k-1\}$$

   where α is the generator of the multiplicative group $\mathbb{F}^*$ of $\mathbb{F}$.[7]

   Prove that

$$\begin{aligned} \mathrm{RS}_{\mathbb{F}_q^*}[n,k,n-k+1] \quad = \quad & \{(c_0, c_1, \ldots, c_{n-1}) \in \mathbb{F}^n \mid c(\alpha^\ell) = 0 \text{ for } 1 \leq \ell \leq n-k\,, \\ & \text{where } c(X) = c_0 + c_1 X + \cdots + c_{n-1} X^{n-1}\}\,. \end{aligned} \tag{5.2}$$

   Hint: Exercise 2.2 might be useful.

2. Recall that the $[2^r, r, 2^{r-1}]_2$ Hadamard code is generated by the $r \times 2^r$ matrix whose ith (for $0 \leq i \leq 2^r - 1$) column is the binary representation of i. Briefly argue that the Hadamard codeword for the message $(m_1, m_2, \ldots, m_r) \in \{0,1\}^r$ is the evaluation of the (multivariate)

[7] This means that $\mathbb{F}_q^* = \{1, \alpha, \ldots, \alpha^{n-1}\}$. Further, $\alpha^n = 1$.

polynomial $m_1X_1 + m_2X_2 + \cdots + m_rX_r$ (where $X_1, \ldots, X_r$ are the r variables) over all the possible assignments to the variables $(X_1, \ldots, X_r)$ from $\{0,1\}^r$.

Using the definition of Hadamard codes above (re)prove the fact that the code has distance 2^{r-1}.

Exercise 5.7. Prove that the $k \times k$ Vandermonde matrix (where the (i,j)th entry is α_j^i) has full rank (where $\alpha_1, \ldots, \alpha_k$ are distinct).

Exercise 5.8. A set $S \subseteq \mathbb{F}_q^n$ is said to be a t-wise independent source (for some $1 \le t \le n$) if given a uniformly random sample $(X_1, \ldots, X_n)$ from S, the n random variables are t-wise independent: i.e. any subset of t variables are uniformly independent random variables over $\mathbb{F}_q$. We will explore properties of these objects in this exercise.

1. Argue that the definition of t-wise independent source is equivalent to the definition in Exercise 2.13.

2. Argue that any $[n,k]_q$ code C is an 1-wise independent source.

3. Prove that any $[n,k]_q$ MDS code is a k-wise independent source.

4. Using part 3 or otherwise prove that there exists a k-wise independent source over $\mathbb{F}_2$ of size at most $(2n)^k$. Conclude that $k(\log_2 n + 1)$ uniformly and independent random bits are enough to compute n random bits that are k-wise independent.

5. For $0 < p \le 1/2$, we say the n binary random variables $X_1, \ldots, X_n$ are p-biased and t-wise independent if any of the t random variables are independent and $\Pr[X_i = 1] = p$ for every $i \in [n]$. For the rest of the problem, let p be a power of $1/2$. Then show that any $t \cdot \log_2(1/p)$-wise independent random variables can be converted into t-wise independent p-biased random variables. Conclude that one can construct such sources with $k \log_2(1/p)(1 + \log_2 n)$ uniformly random bits. Then improve this bound to $k(1 + \max(\log_2(1/p), \log_2 n))$ uniformly random bits.

Exercise 5.9. In many applications, errors occur in "bursts"– i.e. all the error locations are contained in a contiguous region (think of a scratch on a DVD or disk). In this problem we will use how one can use Reed-Solomon codes to correct bursty errors.

An error vector $\mathbf{e} \in \{0,1\}^n$ is called a *t-single burst* error pattern if all the non-zero bits in $\mathbf{e}$ occur in the range $[i, i+t-1]$ for some $1 \le i \le n = t+1$. Further, a vector $\mathbf{e} \in \{0,1\}^n$ is called a (s,t)-burst error pattern if it is the union of at most s t-single burst error pattern (i.e. all non-zero bits in $\mathbf{e}$ are contained in one of at most s contiguous ranges in $[n]$).

We call a binary code $C \subseteq \{0,1\}^n$ to be (s,t)-*burst error correcting* if one can uniquely decode from any (s,t)-burst error pattern. More precisely, given an (s,t)-burst error pattern $\mathbf{e}$ and any codeword $\mathbf{c} \in C$, the only codeword $\mathbf{c}' \in C$ such that $(\mathbf{c} + \mathbf{e}) - \mathbf{c}'$ is an (s,t)-burst error pattern satisfies $\mathbf{c}' = \mathbf{c}$.

1. Argue that if C is (st)-error correcting (in the sense of Definition 1.3.3), then it is also (s,t)-burst error correcting. Conclude that for any $\varepsilon > 0$, there exists code with rate $\Omega(\varepsilon^2)$ and block length n that is (s,t)-burst error correcting for any s, t such that $s \cdot t \le \left(\frac{1}{4} - \varepsilon\right) \cdot n$.

2. Argue that for any rate $R > 0$ and for large enough n, there exist (s,t)-burst error correcting as long as $s \cdot t \leq \left(\frac{1-R-\varepsilon}{2}\right) \cdot n$ and $t \geq \Omega\left(\frac{\log n}{\varepsilon}\right)$. In particular, one can correct from $\frac{1}{2} - \varepsilon$ fraction of burst-errors (as long as each burst is "long enough") with rate $\Omega(\varepsilon)$ (compare this with item 1).

   Hint: Use Reed-Solomon codes.

Exercise 5.10. In this problem we will look at a very important class of codes called BCH codes[8].

Let $\mathbb{F} = \mathbb{F}_{2^m}$. Consider the binary code C_{BCH} defined as $\mathrm{RS}_{\mathbb{F}}[n, k, n-k+1] \cap \mathbb{F}_2^n$.

1. Prove that C_{BCH} is a binary linear code of distance at least $d = n-k+1$ and dimension at least $n - (d-1)\log_2(n+1)$.

   Hint: Use the characterization (5.2) of the Reed-Solomon code from Exercise 5.6.

2. Prove a better lower bound of $n - \left\lceil \frac{d-1}{2} \right\rceil \log_2(n+1)$ on the dimension of C_{BCH}.

   Hint: Try to find redundant checks amongst the "natural" parity checks defining C_{BCH}).

3. For $d = 3$, C_{BCH} is the same as another code we have seen. What is that code?

4. For constant d (and growing n), prove that C_{BCH} have nearly optimal dimension for distance d, in that the dimension cannot be $n - t\log_2(n+1)$ for $t < \frac{d-1}{2}$.

Exercise 5.11. In this exercise, we continue in the theme of Exercise 5.10 and look at the intersection of a Reed-Solomon code with $\mathbb{F}_2^n$ to get a binary code. Let $\mathbb{F} = \mathbb{F}_{2^m}$. Fix positive integers d, n with $(d-1)m < n < 2^m$, and a set $S = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$ of n distinct nonzero elements of $\mathbb{F}$. For a vector $\mathbf{v} = (v_1, \ldots, v_n) \in (\mathbb{F}^*)^n$ of n not necessarily distinct nonzero elements from $\mathbb{F}$, define the *Generalized Reed-Solomon code* $\mathrm{GRS}_{S,\mathbf{v},d}$ as follows:

$$\mathrm{GRS}_{S,\mathbf{v},d} = \{(v_1 p(\alpha_1), v_2 p(\alpha_2), \ldots, v_n p(\alpha_n)) \mid p(X) \in \mathbb{F}[X] \text{ has degree} \leq n-d\} .$$

1. Prove that $\mathrm{GRS}_{S,\mathbf{v},d}$ is an $[n, n-d+1, d]_{\mathbb{F}}$ linear code.

2. Argue that $\mathrm{GRS}_{S,\mathbf{v},d} \cap \mathbb{F}_2^n$ is a binary linear code of rate at least $1 - \frac{(d-1)m}{n}$.

3. Let $\mathbf{c} \in \mathbb{F}_2^n$ be a nonzero binary vector. Prove that (for every choice of d, S) there are at most $(2^m - 1)^{n-d+1}$ choices of the vector $\mathbf{v}$ for which $\mathbf{c} \in \mathrm{GRS}_{S,\mathbf{v},d}$.

4. Using the above, prove that if the integer D satisfies $\mathrm{Vol}_2(n, D-1) < (2^m - 1)^{d-1}$ (where $\mathrm{Vol}_2(n, D-1) = \sum_{i=0}^{D-1} \binom{n}{i}$), then there exists a vector $\mathbf{v} \in (\mathbb{F}^*)^n$ such that the minimum distance of the binary code $\mathrm{GRS}_{S,\mathbf{v},d} \cap \mathbb{F}_2^n$ is at least D.

5. Using parts 2 and 4 above (or otherwise), argue that the family of codes $\mathrm{GRS}_{S,\mathbf{v},d} \cap \mathbb{F}_2^n$ contains binary linear codes that meet the Gilbert-Varshamov bound.

[8] The acronym BCH stands for Bose-Chaudhuri-Hocquenghem, the discoverers of this family of codes.

Exercise 5.12. In this exercise we will show that the dual of a GRS code is a GRS itself with different parameters. First, we state the obvious definition of GRS codes over a general finite field $\mathbb{F}_q$ (as opposed to the definition over fields of characteristic two in Exercise 5.11). In particular, define the code $\mathrm{GRS}_{S,\mathbf{v},d,q}$ as follows:

$$\mathrm{GRS}_{S,\mathbf{v},d,q} = \{(v_1 p(\alpha_1), v_2 p(\alpha_2), \ldots, v_n p(\alpha_n)) \mid p(X) \in \mathbb{F}_q[X] \text{ has degree} \leq n-d\} .$$

Then show that

$$\left(\mathrm{GRS}_{S,\mathbf{v},d,q}\right)^{\perp} = \mathrm{GRS}_{S,\mathbf{v}',n-d+2,q},$$

where $\mathbf{v}' \in \mathbb{F}_q^n$ is a vector with all non-zero components.

Exercise 5.13. In Exercise 2.16, we saw that any linear code can be converted in to a systematic code. In other words, there is a map to convert Reed-Solomon codes into a systematic one. In this exercise the goal is to come up with an explicit encoding function that results in a systematic Reed-Solomon code.

In particular, given the set of evaluation points $\alpha_1, \ldots, \alpha_n$, design an explicit map f from $\mathbb{F}_q^k$ to a polynomial of degree at most $k-1$ such that the following holds. For every message $\mathbf{m} \in \mathbb{F}_q^k$, if the corresponding polynomial is $f_{\mathbf{m}}(X)$, then the vector $\left(f_{\mathbf{m}}(\alpha_i)\right)_{i\in[n]}$ has the message $\mathbf{m}$ appear in the corresponding codeword (say in its first k positions). Further, argue that this map results in an $[n, k, n-k+1]_q$ code.

Exercise 5.14. In this problem, we will consider the number-theoretic counterpart of Reed-Solomon codes. Let $1 \leq k < n$ be integers and let $p_1 < p_2 < \cdots < p_n$ be n distinct primes. Denote $K = \prod_{i=1}^{k} p_i$ and $N = \prod_{i=1}^{n} p_i$. The notation $\mathbb{Z}_M$ stands for integers modulo M, i.e., the set $\{0, 1, \ldots, M-1\}$. Consider the *Chinese Remainder code* defined by the encoding map $E : \mathbb{Z}_K \to \mathbb{Z}_{p_1} \times \mathbb{Z}_{p_2} \times \cdots \times \mathbb{Z}_{p_n}$ defined by:

$$E(m) = (m \mod p_1,\ m \mod p_2,\ \cdots,\ m \mod p_n) .$$

(Note that this is not a code in the usual sense we have been studying since the symbols at different positions belong to different alphabets. Still notions such as distance of this code make sense and are studied in the question below.)

Suppose that $m_1 \neq m_2$. For $1 \leq i \leq n$, define the indicator variable $b_i = 1$ if $E(m_1)_i \neq E(m_2)_i$ and $b_i = 0$ otherwise. Prove that $\prod_{i=1}^{n} p_i^{b_i} > N/K$.

Use the above to deduce that when $m_1 \neq m_2$, the encodings $E(m_1)$ and $E(m_2)$ differ in at least $n-k+1$ locations.

Exercise 5.15. In this problem, we will consider derivatives over a finite field $\mathbb{F}_q$. Unlike the case of derivatives over reals, derivatives over finite fields do not have any physical interpretation but as we shall see shortly, the notion of derivatives over finite fields is still a useful concept. In particular, given a polynomial $f(X) = \sum_{i=0}^{t} f_i X^i$ over $\mathbb{F}_q$, we define its derivative as

$$f'(X) = \sum_{i=0}^{t-1} (i+1) \cdot f_{i+1} \cdot X^i.$$

Further, we will denote by $f^{(i)}(X)$, the result of applying the derivative on f i times. In this problem, we record some useful facts about derivatives.

1. Define $R(X,Z) = f(X+Z) = \sum_{i=0}^{t} r_i(X) \cdot Z^i$. Then for any $j \geq 1$,

$$f^{(j)}(X) = j! \cdot r_j(X).$$

2. Using part 1 or otherwise, show that for any $j \geq \text{char}(\mathbb{F}_q)$,[9] $f^{(j)}(X) \equiv 0$.

3. Let $j \leq \text{char}(\mathbb{F}_q)$. Further, assume that for every $0 \leq i < j$, $f^{(i)}(\alpha) = 0$ for some $\alpha \in \mathbb{F}_q$. Then prove that $(X-\alpha)^j$ divides $f(X)$.

4. Finally, we will prove the following generalization of the degree mantra (Proposition 5.2.3). Let $f(X)$ be a non-zero polynomial of degree t and $m \leq \text{char}(\mathbb{F}_q)$. Then there exists at most $\left\lfloor \frac{t}{m} \right\rfloor$ distinct elements $\alpha \in \mathbb{F}_q$ such that $f^{(j)}(\alpha) = 0$ for every $0 \leq j < m$.

Exercise 5.16. In this exercise, we will consider a code that is related to Reed-Solomon codes and uses derivatives from Exercise 5.15. These codes are called *derivative codes*.

Let $m \geq 1$ be an integer parameter and consider parameters $k > \text{char}(\mathbb{F}_q)$ and n such that $m < k < nm$. Then the derivative code with parameters (n,k,m) is defined as follow. Consider any message $\mathbf{m} \in \mathbb{F}_q^k$ and let $f_{\mathbf{m}}(X)$ be the message polynomial as defined for the Reed-Solomon code. Let $\alpha_1, \ldots, \alpha_n \in \mathbb{F}_q$ be distinct elements. Then the codeword for $\mathbf{m}$ is given by

$$\begin{pmatrix} f_{\mathbf{m}}(\alpha_1) & f_{\mathbf{m}}(\alpha_2) & \cdots & f_{\mathbf{m}}(\alpha_n) \\ f^{(1)}_{\mathbf{m}}(\alpha_1) & f^{(1)}_{\mathbf{m}}(\alpha_2) & \cdots & f^{(1)}_{\mathbf{m}}(\alpha_n) \\ \vdots & \vdots & \vdots & \vdots \\ f^{(m-1)}_{\mathbf{m}}(\alpha_1) & f^{(m-1)}_{\mathbf{m}}(\alpha_2) & \cdots & f^{(m-1)}_{\mathbf{m}}(\alpha_n) \end{pmatrix}.$$

Prove that the above code is an $\left[n, \frac{k}{m}, n - \left\lfloor \frac{k-1}{m} \right\rfloor\right]_{q^m}$-code (and is thus MDS).

Exercise 5.17. In this exercise, we will consider another code related to Reed-Solomon codes that are called *Folded Reed-Solomon* codes. We will see a lot more of these codes in Chapter 14.

Let $m \geq 1$ be an integer parameter and let $\alpha_1, \ldots, \alpha_n \in \mathbb{F}_q$ are distinct elements such that for some element $\gamma \in \mathbb{F}_q^*$, the sets

$$\{\alpha_i, \alpha_i\gamma, \alpha_i\gamma^2, \ldots, \alpha_i\gamma^{m-1}\}, \tag{5.3}$$

are pair-wise disjoint for different $i \in [n]$. Then the folded Reed-Solomon code with parameters $(m,k,n,\gamma,\alpha_1,\ldots,\alpha_n)$ is defined as follows. Consider any message $\mathbf{m} \in \mathbb{F}_q^k$ and let $f_{\mathbf{m}}(X)$ be the message polynomial as defined for the Reed-Solomon code. Then the codeword for $\mathbf{m}$ is given by:

$$\begin{pmatrix} f_{\mathbf{m}}(\alpha_1) & f_{\mathbf{m}}(\alpha_2) & \cdots & f_{\mathbf{m}}(\alpha_n) \\ f_{\mathbf{m}}(\alpha_1 \cdot \gamma) & f_{\mathbf{m}}(\alpha_2 \cdot \gamma) & \cdots & f_{\mathbf{m}}(\alpha_n \cdot \gamma) \\ \vdots & \vdots & \vdots & \vdots \\ f_{\mathbf{m}}(\alpha_1 \cdot \gamma^{m-1}) & f_{\mathbf{m}}(\alpha_2 \cdot \gamma^{m-1}) & \cdots & f_{\mathbf{m}}(\alpha_n \cdot \gamma^{m-1}) \end{pmatrix}.$$

Prove that the above code is an $\left[n, \frac{k}{m}, n - \left\lfloor \frac{k-1}{m} \right\rfloor\right]_{q^m}$-code (and is thus, MDS).

[9] $\text{char}(\mathbb{F}_q)$ denotes the *characteristic* of $\mathbb{F}_q$. That is, if $q = p^s$ for some prime p, then $\text{char}(\mathbb{F}_q) = p$. Any natural number i in $\mathbb{F}_q$ is equivalent to $i \mod \text{char}(\mathbb{F}_q)$.

Exercise 5.18. In this problem we will see that Reed-Solomon codes, derivative codes (Exercise 5.16) and folded Reed-Solomon codes (Exercise 5.17) are all essentially special cases of a large family of codes that are based on polynomials. We begin with the definition of these codes.

Let $m \geq 1$ be an integer parameter and define $m < k \leq n$. Further, let $E_1(X), \ldots, E_n(X)$ be n polynomials over $\mathbb{F}_q$, each of degree m. Further, these polynomials pair-wise do not have any non-trivial factors (i.e. $\gcd(E_i(X), E_j(X))$ has degree 0 for every $i \neq j \in [n]$.) Consider any message $\mathbf{m} \in \mathbb{F}_q^k$ and let $f_{\mathbf{m}}(X)$ be the message polynomial as defined for the Reed-Solomon code. Then the codeword for $\mathbf{m}$ is given by:

$$\left(f_{\mathbf{m}}(X) \mod E_1(X), f_{\mathbf{m}}(X) \mod E_2(X), \ldots, f_{\mathbf{m}}(X) \mod E_n(X)\right).$$

In the above we think of $f_{\mathbf{m}}(X) \mod E_i(X)$ as an element of $\mathbb{F}_{q^m}$. In particular, given given a polynomial of degree at most $m-1$, we will consider any bijection between the q^m such polynomials and $\mathbb{F}_{q^m}$. We will first see that this code is MDS and then we will see why it contains Reed-Solomon and related codes as special cases.

1. Prove that the above code is an $\left[n, \frac{k}{m}, n - \left\lfloor \frac{k-1}{m} \right\rfloor\right]_{q^m}$-code (and is thus, MDS).

2. Let $\alpha_1, \ldots, \alpha_n \in \mathbb{F}_q$ be distinct elements. Define $E_i(X) = X - \alpha_i$. Argue that for this special case the above code (with $m = 1$) is the Reed-Solomon code.

3. Let $\alpha_1, \ldots, \alpha_n \in \mathbb{F}_q$ be distinct elements. Define $E_i(X) = (X - \alpha_i)^m$. Argue that for this special case the above code is the derivative code (with an appropriate mapping from polynomials of degree at most $m-1$ and $\mathbb{F}_q^m$, where the mapping could be different for each $i \in [n]$ and can depend on $E_i(X)$).

4. Let $\alpha_1, \ldots, \alpha_n \in \mathbb{F}_q$ be distinct elements and $\gamma \in \mathbb{F}_q^*$ such that (5.3) is satisfied. Define $E_i(X) = \prod_{j=0}^{m-1}(X - \alpha_i \cdot \gamma^j)$. Argue that for this special case the above code is the folded Reed-Solomon code (with an appropriate mapping from polynomials of degree at most $m-1$ and $\mathbb{F}_q^m$, where the mapping could be different for each $i \in [n]$ and can depend on $E_i(X)$).

Exercise 5.19. In this exercise we will develop a sufficient condition to determine the irreducibility of certain polynomials called the *Eisenstein's criterion*.

Let $F(X, Y)$ be a polynomial of $\mathbb{F}_q$. Think of this polynomial as over X with coefficients as polynomials in Y over $\mathbb{F}_q$. Technically, we think of the coefficients as coming from the ring of polynomials in Y over $\mathbb{F}_q$. We will denote the ring of polynomials in Y over $\mathbb{F}_q$ as $\mathbb{F}_q(Y)$ and we will denote the polynomials in X with coefficients from $\mathbb{F}_q(Y)$ as $\mathbb{F}_q(Y)[X]$.

In particular, let

$$F(X, Y) = X^t + f_{t-1}(Y) \cdot X^{t-1} + \cdots + f_0(Y),$$

where each $f_i(Y) \in \mathbb{F}_q(Y)$. Let $P(Y)$ be a *prime* for $\mathbb{F}_q(Y)$ (i.e. $P(Y)$ has degree at least one and if $P(Y)$ divides $A(Y) \cdot B(Y)$ then $P(Y)$ divides at least one of $A(Y)$ or $B(Y)$). If the following conditions hold:

(i) $P(Y)$ divides $f_i(Y)$ for every $0 \le i < t$; but

(ii) $P^2(Y)$ does not divide $f_0(Y)$

then $F(X,Y)$ does not have any non-trivial factors over $\mathbb{F}_q(Y)[X]$ (i.e. all factors have either degree t or 0 in X).

In the rest of the problem, we will prove this result in a sequence of steps:

1. For the sake of contradiction assume that $F(X,Y) = G(X,Y) \cdot H(X,Y)$ where

$$G(X,Y) = \sum_{i=0}^{t_1} g_i(Y) \cdot X^I \text{ and } H(X,Y) = \sum_{i=0}^{t_2} h_i(Y) \cdot X^i,$$

where $0 < t_1, t_2 < t$. Then argue that $P(Y)$ does not divide both of $g_0(Y)$ and $h_0(Y)$.

For the rest of the problem WLOG assume that $P(Y)$ divides $g_0(Y)$ (and hence does not divide $h_0(Y)$).

2. Argue that there exists an i^* such that $P(Y)$ divide $g_i(Y)$ for every $0 \le i < i^*$ but $P(Y)$ does not divide $g_{i^*}(Y)$ (define $g_t(Y) = 1$).

3. Argue that $P(Y)$ does not divide $f_i(Y)$. Conclude that $F(X,Y)$ does not have any non-trivial factors, as desired.

Exercise 5.20. We have mentioned objects called algebraic-geometric (AG) codes, that generalize Reed-Solomon codes and have some amazing properties: see for example, Section 4.6. The objective of this exercise is to construct one such AG code, and establish its rate vs distance trade-off.

Let p be a prime and $q = p^2$. Consider the equation

$$Y^p + Y = X^{p+1} \tag{5.4}$$

over $\mathbb{F}_q$.

1. Prove that there are exactly p^3 solutions in $\mathbb{F}_q \times \mathbb{F}_q$ to (5.4). That is, if $S \subseteq \mathbb{F}_q^2$ is defined as

$$S = \left\{(\alpha,\beta) \in \mathbb{F}_q^2 \mid \beta^p + \beta = \alpha^{p+1}\right\}$$

then $|S| = p^3$.

2. Prove that the polynomial $F(X,Y) = Y^p + Y - X^{p+1}$ is irreducible over $\mathbb{F}_q$.
Hint: Exercise 5.19 could be useful.

3. Let $n = p^3$. Consider the evaluation map $\mathrm{ev} : \mathbb{F}_q[X,Y] \to \mathbb{F}_q^n$ defined by

$$\mathrm{ev}(f) = (f(\alpha,\beta) : (\alpha,\beta) \in S)\ .$$

Argue that if $f \neq 0$ and is not divisible by $Y^p + Y - X^{p+1}$, then $\mathrm{ev}(f)$ has Hamming weight at least $n - \deg(f)(p+1)$, where $\deg(f)$ denotes the *total* degree of f.
Hint: You are allowed to make use of *Bézout's theorem*, which states that if $f, g \in \mathbb{F}_q[X,Y]$ are nonzero polynomials *with no common factors*, then they have at most $\deg(f)\deg(g)$ common zeroes.

4. For an integer parameter $\ell \geq 1$, consider the set $\mathscr{F}_\ell$ of bivariate polynomials

$$\mathscr{F}_\ell = \left\{ f \in \mathbb{F}_q[X, Y] \mid \deg(f) \leq \ell, \deg_X(f) \leq p \right\}$$

where $\deg_X(f)$ denotes the degree of f in X.

Argue that $\mathscr{F}_\ell$ is an $\mathbb{F}_q$-linear space of dimension $(\ell+1)(p+1) - \frac{p(p+1)}{2}$.

5. Consider the code $C \subseteq \mathbb{F}_q^n$ for $n = p^3$ defined by

$$C = \left\{ \mathrm{ev}(f) \mid f \in \mathscr{F}_\ell \right\} .$$

Prove that C is a linear code with minimum distance at least $n - \ell(p+1)$.

6. Deduce a construction of an $[n, k]_q$ code with distance $d \geq n - k + 1 - p(p-1)/2$.

(Note that Reed-Solomon codes have $d = n - k + 1$, whereas these codes are off by $p(p-1)/2$ from the Singleton bound. However they are much longer than Reed-Solomon codes, with a block length of $n = q^{3/2}$, and the deficiency from the Singleton bound is only $o(n)$.)

5.5 Bibliographic Notes

Reed-Solomon codes were invented by Irving Reed and Gus Solomon [62]. Even though Reed-Solomon codes need $q \geq n$, they are used widely in practice. For example, Reed-Solomon codes are used in storage of information in CDs and DVDs. This is because they are robust against burst-errors that come in contiguous manner. In this scenario, a large alphabet is then a good thing since bursty errors will tend to corrupt the entire symbol in $\mathbb{F}_q$ unlike partial errors, e.g. errors over bits. (See Exercise 5.9.)

It is a big open question to present a deterministic algorithm to compute an irreducible polynomial of a given degree with the same time complexity as in Corollary 5.1.3. Such results are known in general if one is happy with polynomial dependence on q instead of $\log q$. See the book by Shoup [66] for more details.

Chapter 6

What Happens When the Noise is Stochastic: Shannon's Theorem

Shannon was the first to present a rigorous mathematical framework for communication, which (as we have already seen) is the problem of reproducing at one point (typically called the "receiver" of the channel) a message selected at another point (called the "sender" to the channel). Unlike Hamming, Shannon modeled the noise stochastically, i.e. as a well defined random process. He proved a result that pin-pointed the best possible rate of transmission of information over a very wide range of stochastic channels. In fact, Shannon looked at the communication problem at a higher level, where he allowed for compressing the data first (before applying any error-correcting code), so as to minimize the amount of symbols transmitted over the channel.

In this chapter, we will study some stochastic noise models (most of) which were proposed by Shannon. We then prove an optimal tradeoff between the rate and fraction of errors that are correctable for a specific stochastic noise model called the Binary Symmetric Channel.

6.1 Overview of Shannon's Result

Shannon introduced the notion of reliable communication[1] over noisy channels. Broadly, there are two types of channels that were studied by Shannon:

- (Noisy Channel) This type of channel introduces errors during transmission, which result in an incorrect reception of the transmitted signal by the receiver. Redundancy is added at the transmitter to increase reliability of the transmitted data. The redundancy is taken off at the receiver. This process is termed as *Channel Coding*.

- (Noise-free Channel) As the name suggests, this channel does not introduce any type of error in transmission. Redundancy in source data is used to compress the source data at the transmitter. The data is decompressed at the receiver. The process is popularly known as *Source Coding*.

[1] That is, the ability to successfully send the required information over a channel that can lose or corrupt data.

Figure 6.1 presents a generic model of a communication system, which combines the two concepts we discussed above.

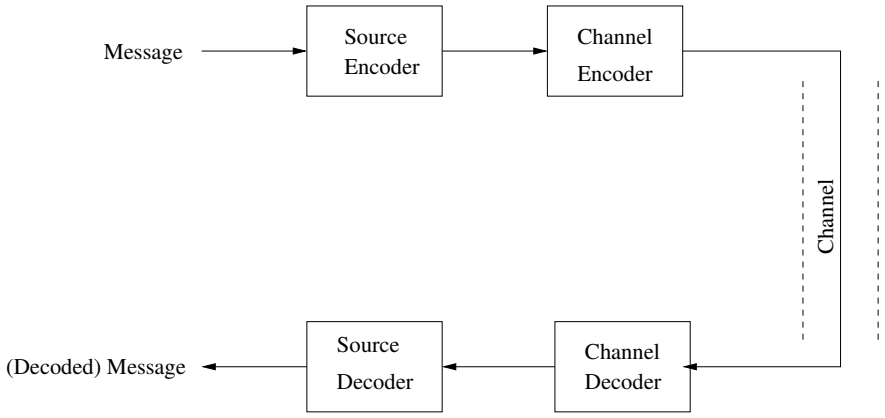


Figure 6.1: The communication process

In Figure 6.1, source coding and channel coding are coupled. In general, to get the optimal performance, it makes sense to design both the source and channel coding schemes simultaneously. However, Shannon's source coding theorem allows us to decouple both these parts of the communication setup and study each of these parts separately. Intuitively, this makes sense: if one can have reliable communication over the channel using channel coding, then for the source coding the channel effectively has no noise.

For source coding, Shannon proved a theorem that precisely identifies the amount by which the message can be compressed: this amount is related to the *entropy* of the message. We will however, not talk much more about source coding in in this book. (However, see Exercises 6.10, 6.11 and 6.12.) From now on, we will exclusively focus on the channel coding part of the communication setup. Note that one aspect of channel coding is how we model the channel noise. So far we have seen Hamming's worst case noise model in some detail. Next, we will study some specific stochastic channels.

6.2 Shannon's Noise Model

Shannon proposed a stochastic way of modeling noise. The input symbols to the channel are assumed to belong to some *input alphabet* $\mathscr{X}$, while the channel selects symbols from its *output alphabet* $\mathscr{Y}$. The following diagram shows this relationship:

$$\mathscr{X} \ni x \rightarrow \boxed{\text{channel}} \rightarrow y \in \mathscr{Y}$$

The channels considered by Shannon are also *memoryless*, that is, noise acts independently on each transmitted symbol. In this book, we will only study *discrete* channels where both the alphabets $\mathscr{X}$ and $\mathscr{Y}$ are finite. For the sake of variety, we will define one channel that is continuous, though we will not study it in any detail later on.

The final piece in specification of a channel is the *transition matrix* $\mathbf{M}$ that governs the process of how the channel introduces error. In particular, the channel is described in form of a matrix with entries as cross over probability over all combination of the input and output alphabets. For any pair $(x, y) \in \mathcal{X} \times \mathcal{Y}$, let $\Pr(y|x)$ denote the probability that y is output by the channel when x is input to the channel. Then the transition matrix is given by $\mathbf{M}(x, y) = \Pr(y|x)$. Specific structure of the matrix is shown below.

$$\mathbf{M} = \begin{pmatrix} & \vdots & \\ \cdots & \Pr(y|x) & \cdots \\ & \vdots & \end{pmatrix}$$

Next, we look at some specific instances of channels.

Binary Symmetric Channel (BSC)**.** Let $0 \leq p \leq 1$. The Binary Symmetric Channel with *crossover probability* p or BSC_p is defined as follows. $\mathcal{X} = \mathcal{Y} = \{0, 1\}$. The 2×2 transition matrix can naturally be represented as a bipartite graph where the left vertices correspond to the rows and the right vertices correspond to the columns of the matrix, where $\mathbf{M}(x, y)$ is represented as the weight of the corresponding (x, y) edge. For BSC_p, the graph is illustrated in Figure 6.2.

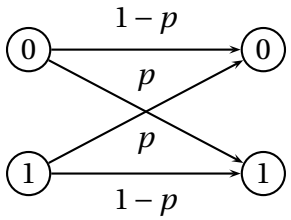


Figure 6.2: Binary Symmetric Channel BSC_p

In other words, every bit is flipped with probability p. We claim that we need to only consider the case when $p \leq \frac{1}{2}$, i.e. if we know how to ensure reliable communication over BSC_p for $p \leq \frac{1}{2}$, then we can also handle the case of $p > \frac{1}{2}$. (See Exercise 6.1.)

q-ary Symmetric Channel (qSC)**.** We now look at the generalization of BSC_p to alphabets of size $q \geq 2$. Let $0 \leq p \leq 1 - \frac{1}{q}$. (As with the case of BSC_p, we can assume that $p \leq 1 - 1/q$– see Exercise 6.2.) The q-ary Symmetric Channel with crossover probability p, or $q\mathrm{SC}_p$ is defined as follows. $\mathcal{X} = \mathcal{Y} = [q]$. The transition matrix $\mathbf{M}$ for $q\mathrm{SC}_p$ is defined as follows.

$$M(x, y) = \begin{cases} 1 - p & \text{if } y = x \\ \frac{p}{q-1} & \text{if } y \neq x \end{cases}$$

In other words, every symbol is retained as it at the output with probability $1 - p$ and is distorted to each of the $q - 1$ possible different symbols with equal probability of $\frac{p}{q-1}$.

Binary Erasure Channel (BEC) In the previous two examples that we saw, $\mathscr{X} = \mathscr{Y}$. However this might not always be the case.

Let $0 \leq \alpha \leq 1$. The Binary Erasure Channel with *erasure probability* α (denoted by BEC_α) is defined as follows. $\mathscr{X} = \{0,1\}$ and $\mathscr{Y} = \{0,1,?\}$, where ? denotes an *erasure*. The transition matrix is as follows:

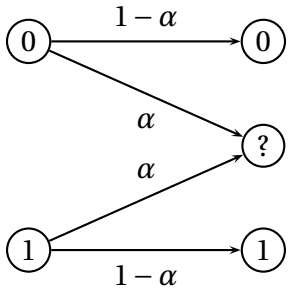


Figure 6.3: Binary Erasure Channel BEC_α

In Figure 6.3 any missing edge represents a transition that occurs with 0 probability. In other words, every bit in BEC_α is erased with probability α (and is left unchanged with probability $1-\alpha$).

Binary Input Additive Gaussian White Noise Channel (BIAGWN)**.** We now look at a channel that is continuous. Let $\sigma \geq 0$. The Binary Input Additive Gaussian White Noise Channel with standard deviation σ or BIAGWN_σ is defined as follows. $\mathscr{X} = \{-1,1\}$ and $\mathscr{Y} = \mathbb{R}$. The noise is modeled by continuous Gaussian probability distribution function. The Gaussian distribution has lots of nice properties and is a popular choice for modeling noise continuous in nature. Given $(x,y) \in \{-1,1\} \times \mathbb{R}$, the noise $y-x$ is distributed according to the Gaussian distribution of mean of zero and standard deviation of σ. In other words,

$$\Pr(y \mid x) = \frac{1}{\sigma\sqrt{2\pi}} \cdot \exp\left(-\left(\frac{(y-x)^2}{2\sigma^2}\right)\right)$$

6.2.1 Error Correction in Stochastic Noise Models

We now need to revisit the notion of error correction from Section 1.3. Note that unlike Hamming's noise model, we cannot hope to *always* recover the transmitted codeword. As an example, in BSC_p there is always some positive probability that a codeword can be distorted into another codeword during transmission. In such a scenario no decoding algorithm can hope to recover the transmitted codeword. Thus, in some stochastic channels there is always some *decoding error probability* (where the randomness is from the channel noise): see Exercise 6.14 for example channels where one can have zero decoding error probability. However, we would like this error probability to be small for every possible transmitted codeword. More precisely, for every message, we would like the decoding algorithm to recover the transmitted message with probability $1-f(n)$, where $\lim_{n\to\infty} f(n) \to 0$, that is $f(n)$ is $o(1)$. Ideally, we would like to have $f(n) = 2^{-\Omega(n)}$. We will refer to $f(n)$ as the decoding error probability.

6.2.2 Shannon's General Theorem

Recall that the big question that we are interested in this book is the tradeoff between the rate of the code and the fraction of errors that can be corrected. For stochastic noise models that we have seen, it is natural to think of the fraction of errors to be the parameter that governs the amount of error that is introduced by the channel. For example, for BSC_p, we will think of p as the fraction of errors.

Shannon's remarkable theorem on channel coding was to *precisely* identify when reliable transmission is possible over the stochastic noise models that he considered. In particular, for the general framework of noise models, Shannon defined the notion of *capacity*, which is a real number such that reliable communication is possible if and only if the rate is less than the capacity of the channel. In other words, given a noisy channel with capacity C, if information is transmitted at rate R for any $R < C$, then there exists a coding scheme that guarantees negligible probability of miscommunication. On the other hand if $R > C$, then regardless of the chosen coding scheme there will be some message for which the decoding error probability is bounded from below by some constant.

In this chapter, we are going to state (and prove) Shannon's general result for the special case of BSC_p.

6.3 Shannon's Result for BSC_p

We begin with a notation. For the rest of the chapter, we will use the notation $\mathbf{e} \sim \mathrm{BSC}_p$ to denote an error pattern $\mathbf{e}$ that is drawn according to the error distribution induced by BSC_p. We are now ready to state the theorem.

Theorem 6.3.1 (Shannon's Capacity Theorem for BSC)**.** *For real numbers p, ε such that $0 \leq p < \frac{1}{2}$ and $0 \leq \varepsilon \leq \frac{1}{2} - p$, the following statements are true for large enough n:*

1. *There exists a real $\delta > 0$, an encoding function $E : \{0,1\}^k \to \{0,1\}^n$ and a decoding function $D : \{0,1\}^n \to \{0,1\}^k$ where $k \leq \lfloor (1 - H(p+\varepsilon))\, n \rfloor$, such that the following holds for every $\mathbf{m} \in \{0,1\}^k$:*
$$\Pr_{\mathbf{e} \sim \mathrm{BSC}_p} [D(E(\mathbf{m}) + \mathbf{e})) \neq \mathbf{m}] \leq 2^{-\delta n}.$$

2. *If $k \geq \lceil (1 - H(p) + \varepsilon)\, n \rceil$ then for every pair of encoding and decoding functions, $E : \{0,1\}^k \to \{0,1\}^n$ and $D : \{0,1\}^n \to \{0,1\}^k$, there exists $\mathbf{m} \in \{0,1\}^k$ such that*
$$\Pr_{\mathbf{e} \sim \mathrm{BSC}_p} [D(E(\mathbf{m}) + \mathbf{e})) \neq \mathbf{m}] \geq \frac{1}{2}.$$

Note that Theorem 6.3.1 implies that the capacity of BSC_p is $1 - H(p)$. It can also be shown that the capacity of $q\mathrm{SC}_p$ and BEC_α are $1 - H_q(p)$ and $1 - \alpha$ respectively. (See Exercises 6.6 and 6.7.)

Entropy function appears in Theorem 6.3.1 due to the same technical reason that it appears in the GV bound: the entropy function allows us to use sufficiently tight bounds on the volume of a Hamming ball (Proposition 3.3.1).

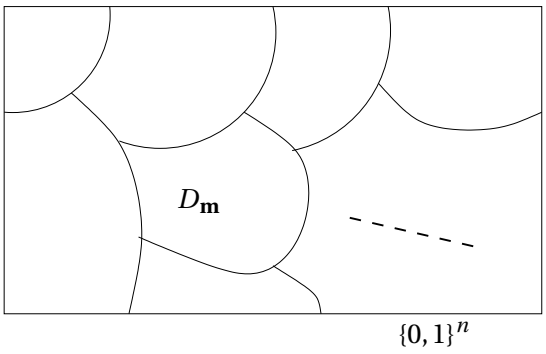


Figure 6.4: The sets $D_{\mathbf{m}}$ partition the ambient space $\{0,1\}^n$.

6.3.1 Proof of Converse of Shannon's Capacity Theorem for BSC

We start with the proof of part (2) of Theorem 6.3.1. (Proof of part (1) follows in the next section.)

For the proof we will assume that $p > 0$ (since when $p = 0$, $1 - H(p) + \varepsilon > 1$ and so we have nothing to prove). For the sake of contradiction, assume that the following holds for every $\mathbf{m} \in \{0,1\}^k$:

$$\Pr_{\mathbf{e} \sim \mathrm{BSC}_p} [D(E(\mathbf{m}) + \mathbf{e}) \neq \mathbf{m}] \leq 1/2.$$

Define $D_{\mathbf{m}}$ to be the set of received words $\mathbf{y}$ that are decoded to $\mathbf{m}$ by D, that is,

$$D_{\mathbf{m}} = \left\{\mathbf{y} | D(\mathbf{y}) = \mathbf{m}\right\}.$$

The main idea behind the proof is the following: first note that the sets $D_{\mathbf{m}}$ partition the entire space of received words $\{0,1\}^n$ (see Figure 6.3.1 for an illustration). (This is because D is a function.) Next we will argue that since the decoding error probability is at most a $1/2$, then $D_{\mathbf{m}}$ for every $\mathbf{m} \in \{0,1\}^k$ is "large." Then by a simple packing argument, it follows that we cannot have too many distinct $\mathbf{m}$, which we will show implies that $k < (1 - H(p) + \varepsilon)n$: a contradiction. Before we present the details, we outline how we will argue that $D_{\mathbf{m}}$ is large. Let $S_{\mathbf{m}}$ be the shell of radius $[(1-\gamma)pn, (1+\gamma)pn]$ around $E(\mathbf{m})$, that is,

$$S_{\mathbf{m}} = B\left(E(\mathbf{m}), (1+\gamma)pn\right) \setminus B\left(E(\mathbf{m}), (1-\gamma)pn\right).$$

(We will set $\gamma > 0$ in terms of ε and p at the end of the proof.) See Figure 6.3.1 for an illustration. Then we argue that because the decoding error probability is bounded by $1/2$, most of the received words in the shell $S_{\mathbf{m}}$ are decoded correctly, i.e. they fall in $D_{\mathbf{m}}$. To complete the argument, we show that number of such received words is indeed large enough.

Fix an arbitrary message $\mathbf{m} \in \{0,1\}^k$. Note that by our assumption, the following is true (where from now on we omit the explicit dependence of the probability on the BSC_p noise for clarity):

$$\Pr[E(\mathbf{m}) + \mathbf{e} \notin D_{\mathbf{m}}] \leq 1/2. \tag{6.1}$$

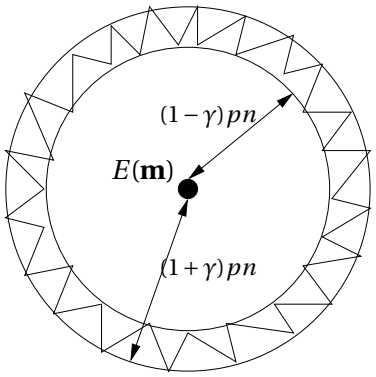


Figure 6.5: The shell $S_{\mathbf{m}}$ of inner radius $(1-\gamma)pn$ and outer radius $(1+\gamma)pn$.

Further, by the (multiplicative) Chernoff bound (Theorem 3.1.6),

$$\Pr\left[E(\mathbf{m})+\mathbf{e}\notin S_{\mathbf{m}}\right]\leq 2^{-\Omega(\gamma^2 n)}. \tag{6.2}$$

(6.1) and (6.2) along with the union bound (Proposition 3.1.3) imply the following:

$$\Pr\left[E(\mathbf{m})+\mathbf{e}\notin D_{\mathbf{m}}\cap S_{\mathbf{m}}\right]\leq \frac{1}{2}+2^{-\Omega(\gamma^2 n)}.$$

The above in turn implies that

$$\Pr\left[E(\mathbf{m})+\mathbf{e}\in D_{\mathbf{m}}\cap S_{\mathbf{m}}\right]\geq \frac{1}{2}-2^{-\Omega(\gamma^2 n)}\geq \frac{1}{4}, \tag{6.3}$$

where the last inequality holds for large enough n. Next we upper bound the probability above to obtain a lower bound on $|D_{\mathbf{m}}\cap S_{\mathbf{m}}|$.

It is easy to see that

$$\Pr\left[E(\mathbf{m})+\mathbf{e}\in D_{\mathbf{m}}\cap S_{\mathbf{m}}\right]\leq |D_{\mathbf{m}}\cap S_{\mathbf{m}}|\cdot p_{\max},$$

where

$$p_{\max}=\max_{\mathbf{y}\in S_{\mathbf{m}}}\Pr\left[E(\mathbf{m})+\mathbf{e}=\mathbf{y}\right]=\max_{d\in[(1-\gamma)pn,(1+\gamma)pn]}p^d(1-p)^{n-d}.$$

In the above, the second equality follows from the fact that all error patterns with the same Hamming weight appear with the same probability when chosen according to BSC_p. Next, note that $p^d(1-p)^{n-d}$ is decreasing in d for $p\leq 1/2$.[2] Thus, we have

$$p_{\max}=p^{(1-\gamma)pn}(1-p)^{n-(1-\gamma)pn}=\left(\frac{1-p}{p}\right)^{\gamma pn}\cdot p^{pn}(1-p)^{(1-p)n}=\left(\frac{1-p}{p}\right)^{\gamma pn}2^{-nH(p)}.$$

[2]Indeed $p^d(1-p)^{n-d}=(p/(1-p))^d(1-p)^n$ and the bound $p\leq 1/2$ implies that the first exponent is at most 1, which implies that the expression is decreasing in d.

Thus, we have shown that

$$\Pr\left[E(\mathbf{m})+\mathbf{e}\in D_{\mathbf{m}}\cap S_{\mathbf{m}}\right]\leq |D_{\mathbf{m}}\cap S_{\mathbf{m}}|\cdot\left(\frac{1-p}{p}\right)^{\gamma pn}2^{-nH(p)},$$

which, by (6.3), implies that

$$|D_{\mathbf{m}}\cap S|\geq\frac{1}{4}\cdot\left(\frac{1-p}{p}\right)^{-\gamma pn}2^{nH(p)}. \tag{6.4}$$

Next, we consider the following sequence of relations:

$$\begin{aligned}
2^n &= \sum_{\mathbf{m}\in\{0,1\}^k}|D_{\mathbf{m}}| && (6.5)\\
&\geq \sum_{\mathbf{m}\in\{0,1\}^k}|D_{\mathbf{m}}\cap S_{\mathbf{m}}| \\
&\geq \frac{1}{4}\left(\frac{1}{p}-1\right)^{-\gamma pn}\sum_{\mathbf{m}\in\{0,1\}^k}2^{H(p)n} && (6.6)\\
&= 2^{k-2}\cdot 2^{H(p)n-\gamma p\log(1/p-1)n} \\
&> 2^{k+H(p)n-\varepsilon n}. && (6.7)
\end{aligned}$$

In the above, (6.5) follows from the fact that for $\mathbf{m}_1\neq\mathbf{m}_2$, $D_{\mathbf{m}_1}$ and $D_{\mathbf{m}_2}$ are disjoint. (6.6) follows from (6.4). (6.7) follows for large enough n and if we pick $\gamma=\frac{\varepsilon}{2p\log\left(\frac{1}{p}-1\right)}$. (Note that as $0<p<1/2$, $\gamma=\Theta(\varepsilon)$.)

(6.7) implies that $k<(1-H(p)+\varepsilon)n$, which is a contradiction. The proof of part (2) of Theorem 6.3.1 is complete.

Remark 6.3.1. It can be verified that the proof above can also work if the decoding error probability is bounded by $1-2^{-\beta n}$ (instead of the $1/2$ in part (2) of Theorem 6.3.1) for small enough $\beta=\beta(\varepsilon)>0$.

Next, we will prove part (1) of Theorem 6.3.1.

6.3.2 Proof of Positive Part of Shannon's Theorem

Proof Overview. The proof of part (1) of Theorem 6.3.1 will be done by the probabilistic method (Section 3.2). In particular, we randomly select an encoding function $E:\{0,1\}^k\rightarrow\{0,1\}^n$. That is, for every $\mathbf{m}\in\{0,1\}^k$ pick $E(\mathbf{m})$ uniformly and independently at random from $\{0,1\}^n$. D will be the maximum likelihood decoding (MLD) function. The proof will have the following two steps:

- (Step 1) For any arbitrary $\mathbf{m}\in\{0,1\}^k$, we will show that for a random choice of E, the probability of failure, over BSC_p noise, is small. This implies the existence of a good encoding function for any arbitrary message.

- (Step 2) We will show a similar result for *all* $\mathbf{m}$. This involves dropping half of the code words.

Note that there are two sources of randomness in the proof:

1. Randomness in the choice of encoding function E and
2. Randomness in the noise.

We stress that the first kind of randomness is for the probabilistic method while the second kind of randomness will contribute to the decoding error probability.

"Proof by picture" of Step 1**.** Before proving part (1) of Theorem 6.3.1, we will provide a pictorial proof of Step 1. We begin by fixing $\mathbf{m} \in \{0,1\}^k$. In Step 1, we need to estimate the following quantity:

$$\mathbb{E}_E \left[\Pr_{\mathbf{e} \sim \mathrm{BSC}_p} [D(E(\mathbf{m}) + \mathbf{e}) \neq \mathbf{m}] \right].$$

By the additive Chernoff bound (Theorem 3.1.6), with all but an exponentially small probability, the received word will be contained in a Hamming ball of radius $\left(p + \varepsilon'\right) n$ (for some $\varepsilon' > 0$ that we will choose appropriately). So one can assume that the received word $\mathbf{y}$ with high probability satisfies $\Delta(E(\mathbf{m}), \mathbf{y}) \leq (p + \varepsilon') n$. Given this, pretty much the only thing to do is to estimate the decoding error probability for such a $\mathbf{y}$. Note that by the fact that D is MLD, an error can happen only if there exists another message $\mathbf{m}'$ such that $\Delta(E(\mathbf{m}'), \mathbf{y}) \leq \Delta(E(\mathbf{m}), \mathbf{y})$. The latter event implies that $\Delta(E(\mathbf{m}'), \mathbf{y}) \leq (p + \varepsilon') n$ (see Figure 6.6). Thus, the decoding error probability is upper bounded by

$$\Pr_{\mathbf{e} \sim \mathrm{BSC}_p} \left[E\left(\mathbf{m}'\right) \in B\left(\mathbf{y}, \left(p + \varepsilon'\right) n\right) \right] = \frac{Vol_2\left(\left(p + \varepsilon'\right) n, n\right)}{2^n} \approx \frac{2^{H(p) n}}{2^n},$$

where the last step follows from Proposition 3.3.1. Finally, by the union bound (Proposition 3.1.3), the existence of such a "bad" $\mathbf{m}'$ is upper bounded by $\approx \frac{2^k 2^{nH(p)}}{2^n}$, which by our choice of k is $2^{-\Omega(n)}$, as desired.

The Details. For notational convenience, we will use $\mathbf{y}$ and $E(\mathbf{m}) + \mathbf{e}$ interchangeably:

$$\mathbf{y} = E(\mathbf{m}) + \mathbf{e}.$$

That is, $\mathbf{y}$ is the received word when $E(\mathbf{m})$ is transmitted and $\mathbf{e}$ is the error pattern.

We start the proof by restating the decoding error probability in part (1) of Shannon's capacity theorem for BSC_p (Theorem 6.3.1) by breaking up the quantity into two sums:

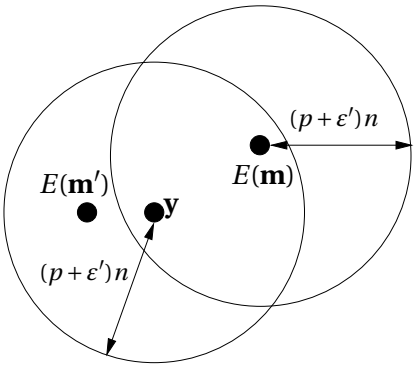


Figure 6.6: Hamming balls of radius $\left(p+\varepsilon'\right)n$ and centers $E(\mathbf{m})$ and $E(\mathbf{m}')$ illustrates Step 1 in the proof of part (1) of Shannon's capacity theorem for the BSC.

$$\begin{aligned}\Pr_{\mathbf{e}\sim \mathrm{BSC}_p}\left[D\left(E\left(\mathbf{m}\right)+\mathbf{e}\right)\neq \mathbf{m}\right] = &\sum_{\mathbf{y}\in B\left(E(\mathbf{m}),(p+\varepsilon')n\right)} \Pr\left[\mathbf{y}|E(\mathbf{m})\right]\cdot \mathbb{1}_{D(\mathbf{y})\neq \mathbf{m}} \\ &+ \sum_{\mathbf{y}\notin B\left(E(\mathbf{m}),(p+\varepsilon')n\right)} \Pr\left[\mathbf{y}|E(\mathbf{m})\right]\cdot \mathbb{1}_{D(\mathbf{y})\neq \mathbf{m}},\end{aligned}$$

where $\mathbb{1}_{D(\mathbf{y})\neq \mathbf{m}}$ is the indicator function for the event that $D(\mathbf{y})\neq \mathbf{m}$ *given* that $E(\mathbf{m})$ was the transmitted codeword and we use $\mathbf{y}|E(\mathbf{m})$ as a shorthand for "$\mathbf{y}$ is the received word given that $E(\mathbf{m})$ was the transmitted codeword." As $\mathbb{1}_{D(\mathbf{y})\neq \mathbf{m}}\leq 1$ (since it takes a value in $\{0,1\}$) and by the (additive) Chernoff bound (Theorem 3.1.6) we have

$$\Pr_{\mathbf{e}\sim \mathrm{BSC}_p}\left[D\left(E\left(\mathbf{m}\right)+\mathbf{e}\right)\neq \mathbf{m}\right] \leq \sum_{\mathbf{y}\in B\left(E(\mathbf{m}),(p+\varepsilon')n\right)} \Pr\left[\mathbf{y}|E(\mathbf{m})\right]\cdot \mathbb{1}_{D(\mathbf{y})\neq \mathbf{m}} + e^{-(\varepsilon')^2 n/2}.$$

In order to apply the probabilistic method (Section 3.2), we will analyze the expectation (over the random choice of E) of the decoding error probability, which by the upper bound above satisfies

$$\mathbb{E}_E\left[\Pr_{\mathbf{e}\sim \mathrm{BSC}_p}\left[D\left(E\left(\mathbf{m}\right)+\mathbf{e}\right)\neq \mathbf{m}\right]\right] \leq e^{-\varepsilon'^2 n/2} + \sum_{\mathbf{y}\in B\left(E(\mathbf{m}),(p+\varepsilon')n\right)} \Pr_{\mathbf{e}\sim \mathrm{BSC}_p}\left[\mathbf{y}|E(\mathbf{m})\right]\cdot \mathbb{E}_E\left[\mathbb{1}_{D(\mathbf{y})\neq \mathbf{m}}\right]. \quad (6.8)$$

In the above we used linearity of expectation (Proposition 3.1.2) and the fact that the distributions on $\mathbf{e}$ and E are independent.

Next, for a fixed received word $\mathbf{y}$ and the transmitted codeword $E(\mathbf{m})$ such that $\Delta(\mathbf{y},E(\mathbf{m}))\leq (p+\varepsilon')n$ we estimate $\mathbb{E}_E\left[\mathbb{1}_{D(\mathbf{y})\neq \mathbf{m}}\right]$. Since D is MLD, we have

$$\mathbb{E}_E\left[\mathbb{1}_{D(\mathbf{y})\neq \mathbf{m}}\right] = \Pr_E\left[\mathbb{1}_{D(\mathbf{y})\neq \mathbf{m}}|E(\mathbf{m})\right] \leq \sum_{\mathbf{m}'\neq \mathbf{m}} \Pr\left[\Delta\left(E\left(\mathbf{m}'\right),\mathbf{y}\right)\leq \Delta\left(E\left(\mathbf{m}\right),\mathbf{y}\right)|E(\mathbf{m})\right], \quad (6.9)$$

where in the above "$|E(\mathbf{m})$" is short for "being conditioned on $E(\mathbf{m})$ being transmitted" and the inequality follows from the union bound (Proposition 3.1.3) and the fact that D is MLD.

Noting that $\Delta(E(\mathbf{m}'),\mathbf{y}) \leq \Delta(E(\mathbf{m}),\mathbf{y}) \leq (p+\varepsilon')n$ (see Figure 6.6), by (6.9) we have

$$\begin{aligned}
\mathbb{E}_E\left[\mathbb{1}_{D(\mathbf{y})\neq\mathbf{m}}\right] &\leq \sum_{\mathbf{m}'\neq\mathbf{m}} \Pr\left[E\left(\mathbf{m}'\right) \in B\left(\mathbf{y},\left(p+\varepsilon'\right)n\right) | E(\mathbf{m})\right] \\
&= \sum_{\mathbf{m}'\neq\mathbf{m}} \frac{\left|B\left(\mathbf{y},\left(p+\varepsilon'\right)n\right)\right|}{2^n} && (6.10)\\
&\leq \sum_{\mathbf{m}'\neq\mathbf{m}} \frac{2^{H(p+\varepsilon')n}}{2^n} && (6.11)\\
&< 2^k \cdot 2^{-n(1-H(p+\varepsilon'))} \\
&\leq 2^{n(1-H(p+\varepsilon))-n(1-H(p+\varepsilon'))} && (6.12)\\
&= 2^{-n(H(p+\varepsilon)-H(p+\varepsilon'))}. && (6.13)
\end{aligned}$$

In the above (6.10) follows from the fact that the choice for $E(\mathbf{m}')$ is independent of $E(\mathbf{m})$. (6.11) follows from the upper bound on the volume of a Hamming ball (Proposition 3.3.1) while (6.12) follows from our choice of k.

Using (6.13) in (6.8), we get

$$\begin{aligned}
\mathbb{E}_E\left[\Pr_{\mathbf{e}\sim\mathrm{BSC}_p}[D(E(\mathbf{m})+\mathbf{e})\neq\mathbf{m}]\right] &\leq e^{-\varepsilon'^2 n/2} + 2^{-n(H(p+\varepsilon)-H(p+\varepsilon'))} \sum_{\mathbf{y}\in B(E(\mathbf{m}),(p+\varepsilon')n)} \Pr\left[\mathbf{y}|E(\mathbf{m})\right] \\
&\leq e^{-\varepsilon'^2 n/2} + 2^{-n(H(p+\varepsilon)-H(p+\varepsilon'))} \leq 2^{-\delta' n}, && (6.14)
\end{aligned}$$

where the second inequality follows from the fact that

$$\sum_{\mathbf{y}\in B(E(\mathbf{m}),(p+\varepsilon')n)} \Pr\left[\mathbf{y}|E(\mathbf{m})\right] \leq \sum_{\mathbf{y}\in\{0,1\}^n} \Pr\left[\mathbf{y}|E(\mathbf{m})\right] = 1$$

and the last inequality follows for large enough n, say $\varepsilon' = \varepsilon/2$ and by picking $\delta' > 0$ to be small enough. (See Exercise 6.3.)

Thus, we have shown that for any arbitrary $\mathbf{m}$ the average (over the choices of E) decoding error probability is small. However, we still need to show that the decoding error probability is exponentially small for *all* messages *simultaneously*. Towards this end, as the bound holds for each $\mathbf{m}$ we have

$$\mathbb{E}_{\mathbf{m}}\left[\mathbb{E}_E\left[\Pr_{\mathbf{e}\sim\mathrm{BSC}_p}[D\left(E\left(\mathbf{m}\right)+\mathbf{e}\right)\neq\mathbf{m}]\right]\right] \leq 2^{-\delta' n}.$$

The order of the summation in the expectation with respect to $\mathbf{m}$ and the summation in the expectation with respect to the choice of E can be switched (as the probability distributions are defined over different domains), resulting in the following expression:

$$\mathbb{E}_E\left[\mathbb{E}_{\mathbf{m}}\left[\Pr_{\mathbf{e}\sim\mathrm{BSC}_p}[D\left(E\left(\mathbf{m}\right)+\mathbf{e}\right)\neq\mathbf{m}]\right]\right] \leq 2^{-\delta' n}.$$

By the probabilistic method, there exists an encoding function E^* (and a corresponding decoding function D^*) such that

$$\mathbb{E}_{\mathbf{m}}\left[\Pr_{\mathbf{e}\sim\mathrm{BSC}_p}\left[D^*\left(E^*\left(\mathbf{m}\right)+\mathbf{e}\right)\neq\mathbf{m}\right]\right]\leq 2^{-\delta' n}. \tag{6.15}$$

(6.15) implies that the *average* decoding error probability is exponentially small. However, recall we need to show that the *maximum* decoding error probability is small. To achieve such a result, we will throw away half of the messages, i.e. *expurgate* the code. In particular, we will order the messages in decreasing order of their decoding error probability and then drop the top half. We claim that the maximum decoding error probability for the remaining messages is $2\cdot 2^{-\delta' n}$. Next, we present the details.

From Average to Worst-Case Decoding Error Probability. We begin with the following "averaging" argument.

Claim 6.3.2. *Let the messages be ordered as* $\mathbf{m}_1,\mathbf{m}_2,\ldots,\mathbf{m}_{2^k}$ *and define*

$$P_i=\Pr_{\mathbf{e}\sim\mathrm{BSC}_p}\left[D(E(\mathbf{m}_i)+\mathbf{e})\neq\mathbf{m}_i\right].$$

Assume that $P_1\leq P_2\leq\ldots\leq P_{2^k}$ *and (6.15) holds, then* $P_{2^{k-1}}\leq 2\cdot 2^{-\delta' n}$

Proof. By the definition of P_i,

$$\begin{aligned}\frac{1}{2^k}\sum_{i=1}^{2^k}P_i&=\mathbb{E}_{\mathbf{m}}\Pr_{\mathbf{e}\sim\mathrm{BSC}_p}\left[D(E(\mathbf{m})+\mathbf{e})\neq\mathbf{m}\right]\\&\leq 2^{-\delta' n},\end{aligned} \tag{6.16}$$

where (6.16) follows from (6.15). For the sake of contradiction assume that

$$P_{2^{k-1}}>2\cdot 2^{-\delta' n}. \tag{6.17}$$

So,

$$\begin{aligned}\frac{1}{2^k}\sum_{i=1}^{2^k}P_i&\geq\frac{1}{2^k}\sum_{i=2^{k-1}+1}^{2^k}P_i &(6.18)\\&>\frac{2\cdot 2^{-\delta' n}\cdot 2^{k-1}}{2^k} &(6.19)\\&>2^{-\delta' n}, &(6.20)\end{aligned}$$

where (6.18) follows by dropping half the summands from the sum. (6.19) follows from (6.17) and the assumption on the sortedness of P_i. The proof is now complete by noting that (6.20) contradicts (6.16). □

Thus, our final code will have $\mathbf{m}_1, \ldots, \mathbf{m}_{2^{k-1}}$ as its messages and hence, has dimension $k' = k-1$. Define $\delta = \delta' + \frac{1}{n}$. In the new code, maximum error probability is at most $2^{-\delta n}$. Also if we picked $k \leq \lfloor \big(1 - H(p+\varepsilon)\big) n \rfloor + 1$, then $k' \leq \lfloor \big(1 - H(p+\varepsilon)\big) n \rfloor$, as required. This completes the proof of Theorem 6.3.1.

We have shown that a random code can achieve capacity. However, we do not know of even a succinct representation of general codes. A natural question to ask is if random linear codes can achieve the capacity of BSC_p. The answer is yes: see Exercise 6.4.

For linear code, representation and encoding are efficient. But the proof does not give an explicit construction. Intuitively, it is clear that since Shannon's proof uses a random code it does not present an "explicit" construction. However, in this book, we will formally define what we mean by an explicit construction.

Definition 6.3.1. A code C of block length n is called *explicit* if there exists a poly(n)-time algorithm that computes a succinct description of C given n. For linear codes, such a succinct description could be a generator matrix or a parity check matrix.

We will also need the following stronger notion of an explicitness:

Definition 6.3.2. A linear $[n,k]$ code C is called *strongly explicit*, if given any index pair $(i,j) \in [k] \times [n]$, there is a poly($\log n$) time algorithm that outputs $G_{i,j}$, where G is a generator matrix of C.

Further, Shannon's proof uses MLD for which only exponential time implementations are known. Thus, the biggest question left unsolved by Shannon's work is the following.

Question 6.3.1. *Can we come up with an explicit construction of a code of rate* $1 - H(p+\varepsilon)$ *with efficient decoding and encoding algorithms that achieves reliable communication over* BSC_p*?*

As a baby step towards the resolution of the above question, one can ask the following question:

Question 6.3.2. *Can we come up with an explicit construction with* $R > 0$ *and* $p > 0$*?*

Note that the question above is similar to Question 2.7.1 in Hamming's world. See Exercise 6.13 for an affirmative answer.

6.4 Hamming vs. Shannon

As a brief interlude, let us compare the salient features of the works of Hamming and Shannon that we have seen so far:

HAMMING	SHANNON
Focus on codewords itself	Directly deals with encoding and decoding functions
Looked at explicit codes	Not explicit at all
Fundamental trade off: rate vs. distance (easier to get a handle on this)	Fundamental trade off: rate vs. error
Worst case errors	Stochastic errors

Intuitively achieving positive results in the Hamming world is harder than achieving positive results in Shannon's world. The reason is that the adversary in Shannon's world (e.g. BSC_p) is much weaker than the worst-case adversary in Hamming's world (say for bits). We make this intuition (somewhat) precise as follows:

Proposition 6.4.1. *Let $0 \leq p < \frac{1}{2}$ and $0 < \varepsilon \leq \frac{1}{2} - p$. If an algorithm A can handle $p + \varepsilon$ fraction of worst case errors, then it can be used for reliable communication over* BSC_p

Proof. By the additive Chernoff bound (Theorem 3.1.6), with probability $\geq 1 - e^{\frac{-\varepsilon^2 n}{2}}$, the fraction of errors in BSC_p is $\leq p + \varepsilon$. Then by assumption on A, it can be used to recover the transmitted message. □

Note that the above result implies that one can have reliable transmission over BSC_p with any code of relative distance $2p + \varepsilon$ (for any $\varepsilon > 0$).

A much weaker converse of Proposition 6.4.1 is also true. More precisely, if the decoding error probability is exponentially small for the BSC, then the corresponding code must have constant relative distance (though this distance does not come even close to achieving say the Gilbert-Varshamov bound). For more see Exercise 6.5.

6.5 Exercises

Exercise 6.1. Let (E, D) be a pair of encoder and decoder that allows for successful transmission over BSC_p for every $p \leq \frac{1}{2}$. Then there exists a pair (E', D') that allows for successful transmission over $\mathrm{BSC}_{p'}$ for any $p' > 1/2$. If D is (deterministic) polynomial time algorithm, then D' also has to be a (deterministic) polynomial time algorithm.

Exercise 6.2. Let (E, D) be a pair of encoder and decoder that allows for successful transmission over $q\mathrm{SC}_p$ for every $p \leq 1 - \frac{1}{q}$. Then there exists a pair (E', D') that allows for successful transmission over $q\mathrm{SC}_{p'}$ for any $p' > 1 - \frac{1}{2}$. If D is polynomial time algorithm, then D' also has to be a polynomial time algorithm though D' can be a randomized algorithm even if D is deterministic.[3]

Exercise 6.3. Argue that in the positive part of Theorem 6.3.1, one can pick $\delta = \Theta(\varepsilon^2)$. That is, for $0 \leq p < 1/2$ and small enough ε, there exist codes of rate $1 - H(p) - \varepsilon$ and block length n that can be decoded with error probability at most $2^{-\Theta(\varepsilon^2)n}$ over BSC_p.

[3] A randomized D' means that given a received word $\mathbf{y}$ the algorithm can use random coins and the decoding error probability is over both the randomness from its internal coin tosses as well as the randomness from the channel.

Exercise 6.4. Prove that there exists linear codes that achieve the BSC_p capacity. (Note that in Section 6.3 we argued that there exists not necessarily a linear code that achieves the capacity.)

Hint: Modify the argument in Section 6.3: in some sense the proof is easier.

Exercise 6.5. Prove that for communication on BSC_p, if an encoding function E achieves a maximum decoding error probability (taken over all messages) that is exponentially small, i.e., at most $2^{-\gamma n}$ for some $\gamma > 0$, then there exists a $\delta = \delta(\gamma, p) > 0$ such that the code defined by E has relative distance at least δ. In other words, good distance is *necessary* for exponentially small maximum decoding error probability.

Exercise 6.6. Prove that the capacity of the $q\mathrm{SC}_p$ is $1 - H_q(p)$.

Exercise 6.7. The binary erasure channel with erasure probability α has capacity $1-\alpha$. In this problem, you will prove this result (and its generalization to larger alphabets) via a sequence of smaller results.

1. For positive integers $k \le n$, show that less than a fraction q^{k-n} of the $k \times n$ matrices G over $\mathbb{F}_q$ fail to generate a linear code of block length n and dimension k. (Or equivalently, except with probability less than q^{k-n}, the rank of a random $k \times n$ matrix G over $\mathbb{F}_q$ is k.)

   Hint: Try out the obvious greedy algorithm to construct a $k \times n$ matrix of rank k. You will see that you will have many choices every step: from this compute (a lower bound on) the number of full rank matrices that can be generated by this algorithm.

2. Consider the q-ary erasure channel with erasure probability α ($q\mathrm{EC}_\alpha$, for some α, $0 \le \alpha \le 1$): the input to this channel is a field element $x \in \mathbb{F}_q$, and the output is x with probability $1-\alpha$, and an erasure '?' with probability α. For a linear code C generated by an $k \times n$ matrix G over $\mathbb{F}_q$, let $D : (\mathbb{F}_q \cup \{?\})^n \to C \cup \{\mathsf{fail}\}$ be the following decoder:

   $$D(\mathbf{y}) = \begin{cases} \mathbf{c} & \text{if } \mathbf{y} \text{ agrees with exactly one } \mathbf{c} \in C \text{ on the unerased entries in } \mathbb{F}_q \\ \mathsf{fail} & \text{otherwise} \end{cases}$$

   For a set $J \subseteq \{1, 2, \ldots, n\}$, let $P_{\mathrm{err}}(G|J)$ be the probability (over the channel noise and choice of a random message) that D outputs fail conditioned on the erasures being indexed by J. Prove that the average value of $P_{\mathrm{err}}(G|J)$ taken over all $G \in \mathbb{F}_q^{k \times n}$ is less than $q^{k-n+|J|}$.

3. Let $P_{\mathrm{err}}(G)$ be the decoding error probability of the decoder D for communication using the code generated by G on the $q\mathrm{EC}_\alpha$. Show that when $k = Rn$ for $R < 1-\alpha$, the average value of $P_{\mathrm{err}}(G)$ over all $k \times n$ matrices G over $\mathbb{F}_q$ is exponentially small in n.

4. Conclude that one can reliably communicate on the $q\mathrm{EC}_\alpha$ at any rate less than $1-\alpha$ using a linear code.

Exercise 6.8. Consider a binary channel whose input/output alphabet is $\{0,1\}$, where a 0 is transmitted faithfully as a 0 (with probability 1), but a 1 is transmitted as a 0 with probability $\frac{1}{2}$ and a 1 with probability 1/2. Compute the capacity of this channel.

Hint: This can be proved from scratch using only simple probabilistic facts already stated/used in the book.

Exercise 6.9. Argue that Reed-Solomon codes from Chapter 5 are strongly explicit codes (as in Definition 6.3.2).

Exercise 6.10. In this problem we will prove a special case of the source coding theorem. For any $0 \leq p \leq 1/2$, let $\mathscr{D}(p)$ be the distribution on $\{0,1\}^n$, where each of the n bits are picked independently to be 1 with probability p and 0 otherwise. Argue that for every $\varepsilon > 0$, strings from $\mathscr{D}(p)$ can be compressed with $H(p+\varepsilon) \cdot n$ bits for large enough n.

More precisely show that for any constant $0 \leq p \leq 1/2$ and every $\varepsilon > 0$, for large enough n there exists an encoding (or compression) function $E : \{0,1\}^n \to \{0,1\}^*$ and a decoding (or decompression) function $D : \{0,1\}^* \to \{0,1\}^n$ such that[4]

1. For every $\mathbf{x} \in \{0,1\}^n$, $D(E(\mathbf{x})) = \mathbf{x}$, and

2. $\mathbb{E}_{\mathbf{x} \leftarrow \mathscr{D}(p)} [|E(\mathbf{x})|] \leq H(p+\varepsilon) \cdot n$, where we use $|E(\mathbf{x})|$ to denote the length of the string $E(\mathbf{x})$. In other words, the *compression rate* is $H(p+\varepsilon)$.

Hint: Handle the "typical" strings from $\mathscr{D}$ and non-typical strings separately.

Exercise 6.11. Show that if there is a constructive solution to Shannon's channel coding theorem with E being a linear map, then there is a constructive solution to Shannon's source coding theorem in the case where the source produces a sequence of independent bits of bias p.

More precisely, let (E,D) be an encoding and decoding pairs that allows for reliable communication over BSC_p with exponentially small decoding error and E is a linear map with rate $1 - H(p) - \varepsilon$. Then there exists a compressing and decompressing pair (E',D') that allows for compression rate $H(p)+\varepsilon$ (where compression rate is as defined in part 2 in Exercise 6.10). The decompression algorithm D' can be randomized and is allowed exponentially small error probability (where the probability can be taken over both the internal randomness of D' and $\mathscr{D}(p)$). Finally if (E,D) are both polynomial time algorithms, then (E',D') have to be polynomial time algorithms too.

Exercise 6.12. Consider a Markovian source of bits, where the source consists of a 6-cycle with three successive vertices outputting 0, and three successive vertices outputting 1, with the probability of either going left (or right) from any vertex is exactly 1/2. More precisely, consider a graph with six vertices $v_0, v_1, \ldots, v_5$ such that there exists an edge $(v_i, v_{(i+1) \mod 6})$ for every $0 \leq i \leq 5$. Further the vertices v_i for $0 \leq i < 3$ are labeled $\ell(v_i) = 0$ and vertices v_j for $3 \leq j < 6$ are labeled $\ell(v_j) = 1$. Strings are generated from this source as follows: one starts with some *start* vertex u_0 (which is one of the v_i's): i.e. the start *state* is u_0. Any any point of time if the current state if u, then the source outputs $\ell(u)$. Then with probability 1/2 the states moves to each of the two neighbors of u.

Compute the optimal compression rate of this source.

[4] We use $\{0,1\}^*$ to denote the set of all binary strings.

Hint: Compress "state diagram" to a minimum and then make some basic observations to compress the source information.

Exercise 6.13. Given codes C_1 and C_2 with encoding functions $E_1 : \{0,1\}^{k_1} \to \{0,1\}^{n_1}$ and $E_2 : \{0,1\}^{k_2} \to \{0,1\}^{n_2}$ let $E_1 \otimes E_2 : \{0,1\}^{k_1 \times k_2} \to \{0,1\}^{n_1 \times n_2}$ be the encoding function obtained as follows: view a message $\mathbf{m}$ as a $k_1 \times k_2$ matrix. Encode the columns of $\mathbf{m}$ individually using the function E_1 to get an $n_1 \times k_2$ matrix $\mathbf{m}'$. Now encode the rows of $\mathbf{m}'$ individually using E_2 to get an $n_1 \times n_2$ matrix that is the final encoding under $E_1 \otimes E_2$ of $\mathbf{m}$. Let $C_1 \otimes C_2$ be the code associated with $E_1 \otimes E_2$ (recall Exercise 2.19).

For $i \geq 3$, let H_i denote the $[2^i - 1, 2^i - i - 1, 3]_2$-Hamming code. Let $C_i = H_i \otimes C_{i-1}$ with $C_3 = H_3$ be a new family of codes.

1. Give a lower bound on the relative minimum distance of C_i. Does it go to zero as $i \to \infty$?

2. Give a lower bound on the rate of C_i. Does it go to zero as $i \to \infty$?

3. Consider the following simple decoding algorithm for C_i: Decode the rows of the rec'd vector recursively using the decoding algorithm for C_{i-1}. Then decode each column according to the Hamming decoding algorithm (e.g. Algorithm 4). Let δ_i denote the probability of decoding error of this algorithm on the BSC_p. Show that there exists a $p > 0$ such that $\delta_i \to 0$ as $i \to \infty$.

   Hint: First show that $\delta_i \leq 4^i \delta_{i-1}^2$.

Exercise 6.14. We consider the problem of determining the best possible rate of transmission on a stochastic memoryless channel with *zero decoding error probability*. Recall that a memoryless stochastic channel is specified by a transition matrix $\mathbf{M}$ s.t. $\mathbf{M}(x, y)$ denotes the probability of y being received if x was transmitted over the channel. Further, the noise acts independently on each transmitted symbol. Let $\mathbb{D}$ denote the input alphabet. Let $R(\mathbf{M})$ denote the best possible rate for a code C such that there exists a decoder D such that for every $\mathbf{c} \in C$, $\Pr[D(\mathbf{y}) \neq \mathbf{c}] = 0$, where $\mathbf{y}$ is picked according to the distribution induced by $\mathbf{M}$ when $\mathbf{c}$ is transmitted over the channel (i.e. the probability that $\mathbf{y}$ is a received word is exactly $\prod_{i=1}^{n} \mathbf{M}(c_i, y_i)$ where C has block length n). In this exercise we will derive an alternate characterization of $R(\mathbf{M})$.

We begin with some definitions related to graphs $\mathcal{G} = (V, E)$. An *independent set* S of $\mathcal{G}$ is a subset $S \subseteq V$ such that there is no edge contained in S, i.e. for every $u \neq v \in S$, $(u, v) \notin E$. For a given graph $\mathcal{G}$, we use $\alpha(\mathcal{G})$ to denote the size of largest independent set in $\mathcal{G}$. Further, given an integer $n \geq 1$, the *n-fold product* of $\mathcal{G}$, which we will denote by $\mathcal{G}^n$, is defined as follows: $\mathcal{G}^n = (V^n, E')$, where $((u_1, \ldots, u_n), (v_1, \ldots, v_n)) \in E'$ if and only if for every $i \in [n]$ either $u_i = v_i$ or $(u_i, v_i) \in E$.

Finally, define a *confusion graph* $\mathcal{G}_{\mathbf{M}} = (V, E)$ as follows. The set of vertices $V = \mathbb{D}$ and for every $x_1 \neq x_2 \in \mathbb{D}$, $(x, y) \in E$ if and only if there exists a y such that $\mathbf{M}(x_1, y) \neq 0$ and $\mathbf{M}(x_2, y) \neq 0$.

1. Prove that
$$R(\mathbf{M}) = \lim_{n\to\infty} \frac{1}{n} \cdot \log_{|\mathbb{D}|}\left(\alpha\left(\mathscr{G}_{\mathbf{M}}^{n}\right)\right).^{5} \tag{6.21}$$

2. A *clique cover* for a graph $\mathscr{G} = (V, E)$ is a partition of the vertices $V = \{V_1, \ldots, V_c\}$ (i.e. V_i and V_j are disjoint for every $i \neq j \in [c]$ and $\cup_i V_i = V$) such that the graph induced on V_i is a *complete graph* (i.e. for every $i \in [c]$ and $x \neq y \in V_i$, we have $(x, y) \in E$). We call c to be the *size* of the clique cover $V_1, \ldots, V_c$. Finally, define $\nu(\mathscr{G})$ to be the size of the smallest clique cover for $\mathscr{G}$. Argue that
$$\alpha(\mathscr{G})^n \leq \alpha(\mathscr{G}^n) \leq \nu(\mathscr{G})^n.$$
Conclude that
$$\log_{|\mathbb{D}|} \alpha(\mathscr{G}) \leq R(\mathbf{M}) \leq \log_{|\mathbb{D}|} \nu(\mathscr{G}). \tag{6.22}$$

3. Consider any transition matrix $\mathbf{M}$ such that the corresponding graph $\mathscr{C}_4 = \mathscr{G}_{\mathbf{M}}$ is a 4-cycle (i.e. the graph $(\{0,1,2,3\}, E)$ where $(i, i+1 \mod 4) \in E$ for every $0 \leq i \leq 3$). Using part 2 or otherwise, argue that $R(\mathbf{M}) = \frac{1}{2}$.

4. Consider any transition matrix $\mathbf{M}$ such that the corresponding graph $\mathscr{C}_5 = \mathscr{G}_{\mathbf{M}}$ is a 5-cycle (i.e. the graph $(\{0,1,2,4\}, E)$ where $(i, i+1 \mod 5) \in E$ for every $0 \leq i \leq 4$). Using part 2 or otherwise, argue that $R(\mathbf{M}) \geq \frac{1}{2} \cdot \log_5 5$. (This lower bound is known to be tight: see Section 6.6 for more.)

6.6 Bibliographic Notes

Shannon's results that were discussed in this chapter appeared in his seminal 1948 paper [65]. All the channels mentioned in this chapter were considered by Shannon except for the *BEC* channel, which was introduced by Elias.

The proof method used to prove Shannon's result for BSC_p has its own name– "random coding with expurgation."

Elias [18] answered Question 6.3.2 (the argument in Exercise 6.13 is due to him).

[5] In literature, $R(\mathbf{M})$ is defined with $\log_{|\mathbb{D}|}$ replaced by $\log_2$. We used the definition in (6.21) to be consistent with our definition of capacity of a noisy channel. See Section 6.6 for more.

Chapter 7

Bridging the Gap Between Shannon and Hamming: List Decoding

In Section 6.4, we made a qualitative comparison between Hamming and Shannon's world. We start this chapter by doing a more quantitative comparison between the two threads of coding theory. In Section 7.2 we introduce the notion of list decoding, which potentially allows us to go beyond the (quantitative) results of Hamming and approach those of Shannon's. Then in Section 7.3, we show how list decoding allows us to go beyond half the distance bound for any code. Section 7.4 proves the optimal trade-off between rate and fraction of correctable errors via list decoding. Finally, in Section 7.5, we formalize why list decoding could be a useful primitive in practical communication setups.

7.1 Hamming versus Shannon: part II

Let us compare Hamming and Shannon theories in terms of the asymptotic bounds we have seen so far (recall rate $R = \frac{k}{n}$ and relative distance $\delta = \frac{d}{n}$).

- Hamming theory: Can correct $\leq \frac{\delta}{2}$ fraction of worse case errors for codes of relative distance δ. By the Singleton bound (Theorem 4.3.1),

  $$\delta \leq 1 - R,$$

  which by Proposition 1.4.1 implies that p fraction of errors can be corrected has to satisfy

  $$p \leq \frac{1-R}{2}.$$

  The above can be achieved via efficient decoding algorithms for example for Reed-Solomon codes (we'll see this later in the book).

- Shannon theory: In $q\mathrm{SC}_p$, for $0 \leq p < 1 - 1/q$, we can have reliable communication with $R < 1 - H_q(p)$. It can be shown that

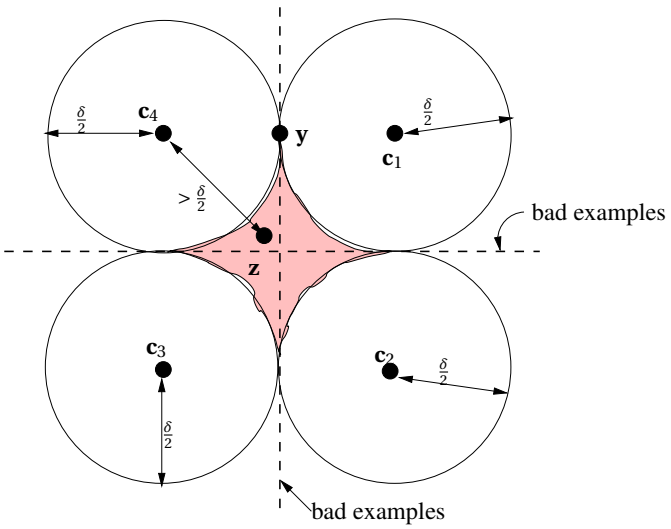


Figure 7.1: In this example vectors are embedded into Euclidean space such that the Euclidean distance between two mapped points is the same as the Hamming distance between vectors. $\mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3, \mathbf{c}_4$ are codewords. The dotted lines contain the "bad examples," that is, the received words for which unique decoding is not possible.

1. $1 - H_q(p) \leq 1 - p$ (this is left as an exercise); and
2. $1 - H_q(p) \geq 1 - p - \varepsilon$, for large enough q– in particular, $q = 2^{\Omega(1/\varepsilon)}$ (Proposition 3.3.2).

Thus, we can have reliable communication with $p \sim 1 - R$ on $q\mathrm{SC}_p$ for large enough q.

There is a gap between Shannon and Hamming world: one can correct twice as many errors in Shannon's world. One natural question to ask is whether we can somehow "bridge" this gap. Towards this end, we will now re-visit the the bad example for unique decoding (Figure 1.3) and consider an extension of the bad example as shown in Figure 7.1.

Recall that $\mathbf{y}$ and the codewords $\mathbf{c}_1$ and $\mathbf{c}_2$ form the bad example for unique decoding that we have already seen before. Recall that for this particular received word we can not do error recovery by unique decoding since there are two codewords $\mathbf{c}_1$ and $\mathbf{c}_2$ having the same distance $\frac{\delta}{2}$ from vector $\mathbf{y}$. On the other hand, the received word $\mathbf{z}$ has an unique codeword $\mathbf{c}_1$ with distance $p > \frac{\delta}{2}$. However, unique decoding does not allow for error recovery from $\mathbf{z}$. This is because by definition of unique decoding, the decoder must be able to recover from *every* error pattern (with a given Hamming weight bound). Thus, by Proposition 1.4.1, the decoded codeword cannot have relative Hamming distance larger than $\delta/2$ from the received word. In this example, because of the received word $\mathbf{y}$, unique decoding gives up on the opportunity to decode $\mathbf{z}$.

Let us consider the example in Figure 7.1 for the binary case. It can be shown that the number of vectors in dotted lines is insignificant compared to volume of shaded area (for large enough block length of the code). The volume of all Hamming balls of radius $\frac{\delta}{2}$ around all the

2^k codewords is roughly equal to:

$$2^k 2^{nH(\frac{\delta}{2})},$$

which implies that the volume of the shaded area (without the dotted lines) is approximately equal to:

$$2^n - 2^k 2^{nH(\frac{\delta}{2})}.$$

In other words, the volume when expressed as a fraction of the volume of the ambient space is roughly:

$$1 - 2^{-n(1-H(\frac{\delta}{2})-R)}, \tag{7.1}$$

where $k = Rn$ and by the Hamming bound (Theorem 1.3) $R \leq 1 - H(\frac{\delta}{2})$. If $R < 1 - H(\frac{\delta}{2})$ then second term of (7.1) is very small. Therefore the number of vectors in shaded area (without the bad examples) is almost all of the ambient space. Note that by the stringent condition on unique decoding none of these received words can be decoded (even though for such received words there is a unique closest codeword). Thus, in order to be able to decode such received vectors, we need to relax the notion of unique decoding. We will consider such a relaxation called *list decoding* next.

7.2 List Decoding

The new notion of decoding that we will discuss is called *list decoding* as the decoder is allowed to output a list of answers. We now formally define (the combinatorial version of) list decoding:

Definition 7.2.1. Given $0 \leq \rho \leq 1, L \geq 1$, a code $C \subseteq \Sigma^n$ is (ρ, L)-list decodable if for every received word $\mathbf{y} \in \Sigma^n$,

$$\left|\{c \in C | \Delta(\mathbf{y}, c) \leq \rho n\}\right| \leq L$$

Given an error parameter ρ, a code C and a received word $\mathbf{y}$, a list-decoding algorithm should output all codewords in C that are within (relative) Hamming distance ρ from $\mathbf{y}$. Note that if the fraction of errors that occurred during transmission is at most ρ then the transmitted codeword is *guaranteed* to be in the output list. Further, note that if C is (ρ, L)-list decodable then the algorithm will always output at most L codewords for any received word. In other words, for efficient list-decoding algorithm, L should be a polynomial in the block length n (as otherwise the algorithm will have to output a super-polynomial number of codewords and hence, cannot have a polynomial running time). Thus, the restriction of L being at most some polynomial in n is an *a priori* requirement enforced by the fact that we are interested in efficient polynomial time decoding algorithms. Another reason for insisting on a bound on L is that otherwise the decoding problem can become trivial: for example, one can output all the codewords in the code. Finally, it is worthwhile to note that one can always have an exponential time list-decoding algorithm: go through all the codewords in the code and pick the ones that are within ρ (relative) Hamming distance of the received word.

Note that in the communication setup, we need to recover the transmitted message. In such a scenario, outputting a list might not be useful. There are two ways to get around this "problem":

1. Declare a decoding error if list size > 1. Note that this generalizes unique decoding (as when the number of errors is at most half the distance of the code then there is a unique codeword and hence, the list size will be at most one). However, the gain over unique decoding would be substantial only if for most error patterns (of weight significantly more than half the distance of the code) the output list size is at most one. Fortunately, it can be show that:

   - For random codes, with high probability, for most error patterns, the list size is at most one. In other words, for *most* codes, we can hope to see a gain over unique decoding. The proof of this fact follows from Shannon's proof for the capacity for qSC: the details are left as an exercise.
   - In Section 7.5, we show that the above behavior is in fact general: i.e. for *any* code (over a large enough alphabet) it is true that with high probability, for most error patterns, the list size is at most one.

   Thus, using this option to deal with multiple answers, we still deal with worse case errors but can correct more error patterns than unique decoding.

2. If the decoder has access to some side information, then it can use that to prune the list. Informally, if the worst-case list size is L, then the amount of extra information one needs is $O(\log L)$. This will effectively decrease[1] the dimension of the code by $O(\log L)$, so if L is small enough, this will have negligible effect on the rate of the code. There are also application (especially in complexity theory) where one does not really care about the rate being the best possible.

Recall that Proposition 1.4.1 implies that $\delta/2$ is the maximum fraction of errors correctable with unique decoding. Since list decoding is a relaxation of unique decoding, it is natural to wonder

Question 7.2.1. *Can we correct more than $\delta/2$ fraction of errors using list decoding?*

and if so

Question 7.2.2. *What is the maximum fraction of errors correctable using list decoding?*

In particular, note that the intuition from Figure 7.1 states that the answer to Question 7.2.1 should be yes.

[1] Note that side information effectively means that not all possible vectors are valid messages.

7.3 Johnson Bound

In this section, we will indeed answer Question 7.2.1 in the affirmative by stating a bound due to Johnson. To setup the context again, recall that Proposition 1.4.1 implies that any code with relative distance δ is $(\delta/2, 1)$-list decodable.

Notice that if we can show a code for some $e > \left\lfloor \frac{d-1}{2} \right\rfloor$ is $(e/n, n^{O(1)})$-list decodable, then (combinatorially) it is possible to list decode that code up to e errors. We'll show by proving the Johnson bound that this is indeed the case for any code.

Theorem 7.3.1 (Johnson Bound)**.** *Let $C \subseteq [q]^n$ be a code of distance d. If $\rho < J_q\left(\frac{d}{n}\right)$, then C is a (ρ, qdn)-list decodable code, where the function $J_q(\delta)$ is defined as*

$$J_q(\delta) = \left(1 - \frac{1}{q}\right)\left(1 - \sqrt{1 - \frac{q\delta}{q-1}}\right).$$

Proof (for $q = 2$). The proof technique that we will use has a name: double counting. The main idea is to count the same quantity in two different ways to get both an upper and lower bound on the same quantity. These bounds then imply an inequality and we will derive our desired bound from this inequality.

We have to show that for every binary code $C \subseteq \{0,1\}^n$ with distance d (i.e. for every $\mathbf{c}_1 \neq \mathbf{c}_2 \in C$, $\Delta(\mathbf{c}_1, \mathbf{c}_2) \geq d$) and every $\mathbf{y} \in \{0,1\}^n$, $|B(\mathbf{y}, e) \bigcap C| \leq 2dn$.

Fix arbitrary C and $\mathbf{y}$. Let $\mathbf{c}_1, \ldots, \mathbf{c}_M \in B(\mathbf{y}, e)$. We need to show that $M \leq 2dn$. Define $\mathbf{c}'_i = \mathbf{c}_i - \mathbf{y}$ for $1 \leq i \leq M$. Then we have the following:

(i) $wt(\mathbf{c}'_i) \leq e$ for $1 \leq i \leq M$ because $\mathbf{c}_i \in B(\mathbf{y}, e)$.

(ii) $\Delta(\mathbf{c}'_i, \mathbf{c}'_j) \geq d$ for every $i \neq j$ because $\Delta(\mathbf{c}_i, \mathbf{c}_j) \geq d$.

Define

$$S = \sum_{i<j} \Delta(\mathbf{c}'_i, \mathbf{c}'_j).$$

We will prove both an upper and a lower bound on S from which we will extract the required upper bound on M. Then from (ii) we have

$$S \geq \binom{M}{2} d \tag{7.2}$$

Consider the $n \times M$ matrix $(\mathbf{c}'^T_1, \cdots, \mathbf{c}'^T_M)$. Define m_i as the number of 1's in the i-th row for $1 \leq i \leq n$. Then the i-th row of the matrix contributes the value $m_i(M - m_i)$ to S because this is the number of 0-1 pairs in that row. (Note that each such pair contributes one to S.) This implies that

$$S = \sum_{i=1}^{n} m_i(M - m_i). \tag{7.3}$$

Define

$$\bar{e} = \sum_i \frac{m_i}{M}.$$

Note that

$$\sum_{i=1}^{n} m_i = \sum_{j=1}^{M} wt(\mathbf{c}_i) \le eM,$$

where the inequality follows From (i) above. Thus, we have

$$\bar{e} \le e.$$

Using the Cauchy-Schwartz inequality (i.e., $\langle \mathbf{x}, \mathbf{y} \rangle \le ||\mathbf{x}|| \cdot ||\mathbf{y}||$ for $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$) by taking $\mathbf{x} = (m_1, \cdots, m_n)$, $\mathbf{y} = (1/n, \cdots, 1/n)$, we have

$$\left(\frac{\sum_{i=1}^{n} m_i}{n} \right)^2 \le \left(\sum_{i=1}^{n} m_i^2 \right) \frac{1}{n}. \tag{7.4}$$

Thus, from (7.3)

$$S = \sum_{i=1}^{n} m_i (M - m_i) = M^2 \bar{e} - \sum_{i=1}^{n} m_i^2 \le M^2 \bar{e} - \frac{(M\bar{e})^2}{n} = M^2 (\bar{e} - \frac{\bar{e}^2}{n}), \tag{7.5}$$

where the inequality follows from (7.4). By (7.2) and (7.5),

$$M^2 \left(\bar{e} - \frac{\bar{e}^2}{n} \right) \ge \frac{M(M-1)}{2} d,$$

which implies that

$$\begin{aligned} M \le \frac{dn}{dn - 2n\bar{e} + 2\bar{e}^2} = \frac{2dn}{2dn - n^2 + n^2 - 4n\bar{e} + 4\bar{e}^2} &= \frac{2dn}{(n-2\bar{e})^2 - n(n-2d)} \\ &\le \frac{2dn}{(n-2e)^2 - n(n-2d)}, \end{aligned} \tag{7.6}$$

where the last inequality follows from the fact that $\bar{e} \le e$. Then from

$$\frac{e}{n} < \frac{1}{2} \left(1 - \sqrt{1 - \frac{2d}{n}} \right),$$

we get

$$n - 2e > \sqrt{n(n-2d)}.$$

In other words

$$(n-2e)^2 > n(n-2d).$$

Thus, $(n-2e)^2 - n(n-2d) \ge 1$ because n, e are all integers and therefore, from (7.6), we have $M \le 2dn$ as desired. □

Next, we prove the following property of the function $J_q(\cdot)$, which along with the Johnson bound answers Question 7.2.1 in the affirmative.

Lemma 7.3.2. *Let $q \geq 2$ be an integer and let $0 \leq x \leq 1 - \frac{1}{q}$. Then the following inequalities hold:*

$$J_q(x) \geq 1 - \sqrt{1-x} \geq \frac{x}{2},$$

where the second inequality is tight for $x > 0$.

Proof. We start with by proving the inequality

$$\left(1 - \frac{1}{q}\right)\left(1 - \sqrt{1 - \frac{xq}{q-1}}\right) \geq 1 - \sqrt{1-x}.$$

Indeed, both the LHS and RHS of the inequality are zero at $x = 0$. Further, it is easy to check that the derivatives of the LHS and RHS are $\frac{1}{\sqrt{1-\frac{xq}{q-1}}}$ and $\frac{1}{\sqrt{1-x}}$ respectively. The former is always larger than the latter quantity. This implies that the LHS increases more rapidly than the RHS, which in turn proves the required inequality.

The second inequality follows from the subsequent relations. As $x \geq 0$,

$$1 - x + \frac{x^2}{4} \geq 1 - x,$$

which implies that

$$\left(1 - \frac{x}{2}\right)^2 \geq 1 - x,$$

which in turn implies the required inequality. (Note that the two inequalities above are strict for $x > 0$, which implies that $1 - \sqrt{1-x} > x/2$ for every $x > 0$, as desired.) □

Theorem 7.3.1 and Lemma 7.3.2 imply that for *any* code, list decoding can potentially correct strictly more errors than unique decoding in polynomial time, as long as q is at most some polynomial in n (which will be true of all the codes that we discuss in this book). This answers Question 7.2.1 in the affirmative. See Figure 7.2 for an illustration of the gap between the Johnson bound and the unique decoding bound.

Theorem 7.3.1 and Lemma 7.3.2 also implies the following "alphabet-free" version of the Johnson bound.

Theorem 7.3.3 (Alphabet-Free Johnson Bound)**.** *If $e \leq n - \sqrt{n(n-d)}$, then any code with distance d is $(e/n, qnd)$-list decodable for all the q.*

A natural question to ask is the following:

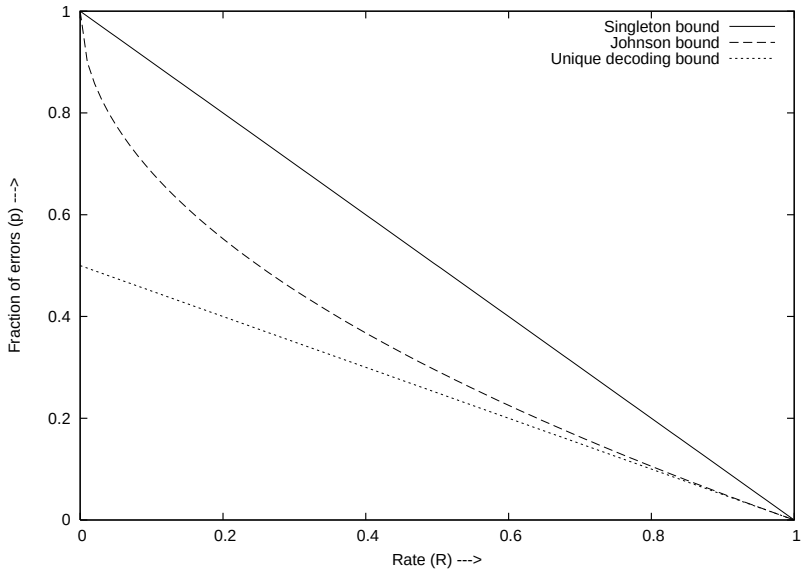


Figure 7.2: The trade-off between rate R and the fraction of errors that can be corrected. $1-\sqrt{R}$ is the trade-off implied by the Johnson bound. The bound for unique decoding is $(1-R)/2$ while $1-R$ is the Singleton bound (and the list decoding capacity for codes over large alphabets).

Question 7.3.1. *Is the Johnson bound tight?*

The answer is yes in the sense that there exist linear codes with relative distance δ such that we can find Hamming ball of radius larger than $J_q(\delta)$ with super-polynomially many codewords. On the other hand, in the next section, we will show that, in some sense, it is not tight.

7.4 List-Decoding Capacity

In the previous section, we saw what can one achieve with list decoding in terms of distance of a code. In this section, let us come back to Question 7.2.2. In particular, we will consider the trade-off between rate and the fraction of errors correctable by list decoding. Unlike the case of unique decoding and like the case of BSC_p, we will be able to prove an optimal trade-off.

Next, we will prove the following result regarding the optimal trade-off between rate of a code and the fraction of errors that can be corrected via list decoding.

Theorem 7.4.1. *Let $q \geq 2$, $0 \leq p < 1-\frac{1}{q}$, and $\varepsilon > 0$. Then the following holds for codes of large enough block length n:*

(i) If $R \leq 1 - H_q(p) - \varepsilon$, then there exists a $\left(p, O\left(\frac{1}{\varepsilon}\right)\right)$-list decodable code.

(ii) If $R > 1 - H_q(p) + \varepsilon$, every $\left(\rho, L\right)$-list decodable code has $L \geq q^{\Omega(n)}$.

Thus, the *List-decoding capacity*[2] is $1 - H_q(p)$ (where p is the fraction of errors). Further, this fully answers Question 7.2.2. Finally, note that this exactly matches capacity for $q\text{SC}_p$ and hence, list decoding can be seen as a bridge between Shannon's world and Hamming's world. The remarkable aspect of this result is that we bridge the gap between these worlds by allowing the decoder to output at most $O(1/\varepsilon)$ many codewords.

7.4.1 Proof of Theorem 7.4.1

We begin with the basic idea behind the proof of part (i) of the theorem.

As in Shannon's proof for capacity of BSC_p, we will use the probabilistic method (Section 3.2). In particular, we will pick a random code and show that it satisfies the required property with non-zero probability. In fact, we will show that a random code is (ρ, L)-list decodable with high probability as long as:

$$R \leq 1 - H_q(p) - \frac{1}{L}$$

The analysis will proceed by proving that probability of a "bad event" is small. "Bad event" means there exist messages $\mathbf{m}_0, \mathbf{m}_1, \cdots, \mathbf{m}_L \in [q]^{Rn}$ and a received code $\mathbf{y} \in [q]^n$ such that:

$$\Delta\big(C(\mathbf{m}_i), \mathbf{y}\big) \leq \rho n, \text{for every } 0 \leq i \leq L.$$

Note that if a bad event occurs, then the code is not a (ρ, L)-list decodable code. The probability of the occurrence of any bad event will then be calculated by an application of the union bound.

Next, we restate Theorem 7.4.1 and prove a stronger version of part (i). (Note that $L = \lceil \frac{1}{\varepsilon} \rceil$ in Theorem 7.4.2 implies Theorem 7.4.1.)

Theorem 7.4.2 (List-Decoding Capacity)**.** *Let $q \geq 2$ be an integer, and $0 < \rho < 1 - \frac{1}{q}$ be a real number.*

(i) Let $L \geq 1$ be an integer, then there exists an $\big(\rho, L\big)$-list decodable code with rate

$$R \leq 1 - H_q(\rho) - \frac{1}{L}$$

(ii) For every $\big(\rho, L\big)$ code of rate $1 - H_q(\rho) + \varepsilon$, it is necessary that $L \geq 2^{\Omega(\varepsilon n)}$.

Proof. We start with the proof of (i). Pick a code C at random where

$$|C| = q^k, \text{ where } k \leq \left(1 - H_q(\rho) - \frac{1}{L}\right) n.$$

That is, as in Shannon's proof, for every message $\mathbf{m}$, pick $C(\mathbf{m})$ uniformly and independently at random from $[q]^n$.

[2] Actually the phrase should be something like "capacity of worst case noise model under list decoding" as the capacity is a property of the channel. However, in the interest of brevity we will only use the term list-decoding capacity.

Given $\mathbf{y} \in [q]^n$, and $\mathbf{m}_0, \cdots, \mathbf{m}_L \in [q]^k$, the tuple $(\mathbf{y}, \mathbf{m}_0, \cdots, \mathbf{m}_L)$ defines a *bad event* if

$$C(\mathbf{m}_i) \in B(\mathbf{y}, \rho n), 0 \leq i \leq L.$$

Note that a code is (ρ, L)-list decodable if and only if there does not exist any bad event.

Fix $\mathbf{y} \in [q]^n$ and $\mathbf{m}_0, \cdots, \mathbf{m}_L \in [q]^k$.
Note that for fixed i, by the choice of C, we have:

$$\Pr[C(\mathbf{m}_i) \in B(\mathbf{y}, \rho n)] = \frac{Vol_q(\rho n, n)}{q^n} \leq q^{-n(1-H_q(\rho))}, \tag{7.7}$$

where the inequality follows from the upper bound on the volume of a Hamming ball (Proposition 3.3.1). Now the probability of a bad event given $(\mathbf{y}, \mathbf{m}_0, \cdots, \mathbf{m}_L)$ is

$$\Pr\left[\bigwedge_{i=0}^{L} C(\mathbf{m}_i) \in B(\mathbf{y}, \rho n)\right] = \prod_{0}^{L} \Pr[C(\mathbf{m}_i) \in B(\mathbf{y}, \rho n)] \leq q^{-n(L+1)(1-H_q(\rho))}, \tag{7.8}$$

where the equality follows from the fact that the random choices of codewords for distinct messages are independent and the inequality follows from (7.7). Then,

$$\begin{aligned}
\Pr[\text{ There is a bad event}] &\leq q^n \binom{q^k}{L+1} q^{-n(L+1)(1-H_q(\rho))} && (7.9)\\
&\leq q^n q^{Rn(L+1)} q^{-n(L+1)(1-H_q(\rho))} && (7.10)\\
&= q^{-n(L+1)[1-H_q(\rho)-\frac{1}{L+1}-R]} \\
&\leq q^{-n(L+1)[1-H_q(\rho)-\frac{1}{L+1}-1+H_q(\rho)+\frac{1}{L}]} && (7.11)\\
&= q^{-\frac{n}{L}} \\
&< 1
\end{aligned}$$

In the above, (7.9) follows by the union bound (Lemma 3.1.3) with (7.8) and by counting the number of $\mathbf{y}$'s (which is q^n), and the number of $L+1$ tuples (which is $\binom{q^k}{L+1}$). (7.10) follows from the fact that $\binom{a}{b} \leq a^b$ and $k = Rn$. (7.11) follows by assumption $R \leq 1 - H_q(\rho) - \frac{1}{L}$. The rest of the steps follow from rearranging and canceling the terms. Therefore, by the probabilistic method, there exists C such that it is (ρ, L)-list decodable.

Now we turn to the proof of part (ii). For this part, we need to show the existence of a $\mathbf{y} \in [q]^n$ such that $|C \cap B(\mathbf{y}, \rho n)|$ is exponentially large for every C of rate $R \geq 1 - H_q(\rho) + \varepsilon$. We will again use the probabilistic method to prove this result.

Pick $\mathbf{y} \in [q]^n$ uniformly at random. Fix $\mathbf{c} \in C$. Then

$$\begin{aligned}
\Pr[\mathbf{c} \in B(\mathbf{y}, \rho n)] &= \Pr[\mathbf{y} \in B(\mathbf{c}, \rho n)] \\
&= \frac{Vol_q(\rho n, n)}{q^n} && (7.12)\\
&\geq q^{-n(1-H_q(\rho))-o(n)}, && (7.13)
\end{aligned}$$

where (7.12) follows from the fact that $\mathbf{y}$ is chosen uniformly at random from $[q]^n$ and (7.13) follows by the lower bound on the volume of the Hamming ball (Proposition 3.3.1).

We have

$$\begin{aligned} E[|C \cap B(\mathbf{y}, \rho n)|] &= \sum_{\mathbf{c} \in C} E[\mathbb{1}_{\mathbf{c} \in B(\mathbf{y}, \rho n)}] && (7.14) \\ &= \sum_{\mathbf{c} \in C} \Pr[\mathbf{c} \in B(\mathbf{y}, \rho n)] \\ &\geq \sum_{\mathbf{c} \in C} q^{-n(1-H_q(\rho)+o(n))} && (7.15) \\ &= q^{n[R-1+H_q(\rho)-o(1)]} \\ &\geq q^{\Omega(\varepsilon n)} \\ & && (7.16) \end{aligned}$$

In the above, (7.14) follows by the linearity of expectation (Proposition 3.1.2), (7.15) follows from (7.13), and (7.16) follows by choice of R. Hence, by the probabilistic method, there exists $\mathbf{y}$ such that $|B(\mathbf{y}, \rho n) \cap C|$ is $q^{\Omega(n)}$, as desired. □

The above proof can be modified to work for random linear codes. (See Exercise **??**.)

We now return to Question 7.3.1. Note that by the Singleton bound, the Johnson bound implies that for any code one can hope to list decode from about $p \leq 1 - \sqrt{R}$ fraction of errors. However, this trade-off between p and R is not tight. Note that Lemma 3.3.2 along with Theorem 7.4.1 implies that for large q, the list decoding capacity is $1 - R > 1 - \sqrt{R}$. Figure 7.2 plots and compares the relevant trade-offs.

Finally, we have shown that the list decoding capacity is $1 - H_q(p)$. However, we showed the existence of a code that achieves the capacity by the probabilistic method. This then raises the following question:

Question 7.4.1. *Do there exist explicit codes that achieve list decoding capacity?*

Also the only list decoding algorithm that we have seen so far is the brute force algorithm that checks every codeword to see if they need to be output. This also leads to the follow-up question

Question 7.4.2. *Can we achieve list decoding capacity with efficient list decoding algorithms?*

A more modest goal related to the above would be the following:

Question 7.4.3. *Can we design an efficient list decoding algorithm that can achieve the Johnson bound? In particular, can we efficiently list decode a code of rate R from $1-\sqrt{R}$ fraction of errors?*

7.5 List Decoding from Random Errors

In this section, we formalize the intuition we developed from Figure 7.1. In particular, recall that we had informally argued that for most error patterns we can correct beyond the $\delta/2$ bound for unique decoding (Proposition 1.4.1). Johnson bound (Theorem 7.3.1) tells us that one can indeed correct beyond $\delta/2$ fraction of errors. However, there are two shortcomings. The first is that the Johnson bounds tells us that the output list size is qdn but it does not necessarily imply that for most error patterns, there is unique by closest codewords (i.e. one can uniquely recover the transmitted codeword). In other words, Johnson bound is a "true" list decoding result and tells us nothing about the behavior of codes on the "average." The second aspect is that the Johnson bound holds for up to $1-\sqrt{1-\delta}$ fraction of errors. Even though it is more than $\delta/2$ for every $\delta > 0$, the bound e.g. is not say twice the unique decoding bound for every $\delta > 0$.

Next we show that for *any* code with relative distance δ (over a large enough alphabet size) for most error patterns, the output of a list decoder for any fraction of errors arbitrarily close to δ will have size one. In fact, the result is somewhat stronger: it show that even if one fixes the error locations *arbitrarily*, for most error patterns the output list size is one.

Theorem 7.5.1. *Let $\varepsilon > 0$ be a real and $q \geq 2^{\Omega(1/\varepsilon)}$ be an integer. Then the following is true for any $0 < \delta < 1 - 1/q$ and large enough n. Let $C \subseteq \{0,1,\ldots q-1\}^n$ be a code with relative distance δ and let $S \subseteq [n]$ such that $|S| = (1-\rho)n$, where $(0 < \rho \leq \delta - \varepsilon)$.*

Then, for all $\mathbf{c} \in C$ and all but a $q^{-\Omega(\varepsilon n)}$ fraction of error patterns, $\mathbf{e} \in \{0,1\ldots q-1\}^n$ such that

$$\mathbf{e}_S = \mathbf{0} \text{ and } wt(\mathbf{e}) = \rho n \tag{7.17}$$

the only codeword within Hamming distance ρn of $\mathbf{c}+\mathbf{e}$ is $\mathbf{c}$ itself.

For illustration of the kinds of error pattern we will deal with, see Figure 7.3.

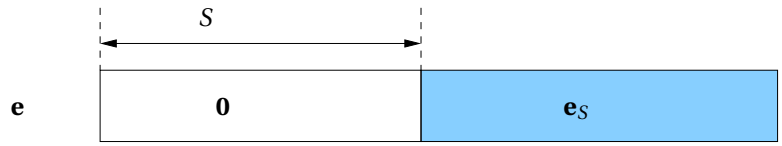


Figure 7.3: Illustration of the kind of error patterns we are trying to count.

Before we present the proof, we present certain corollaries (the proofs of which we leave as exercises). First the result above implies a similar result of the output list size being one for the

following two random noise models: (i) uniform distribution over *all* error patterns of weight ρn and (ii) $q\mathrm{SC}_p$. In fact, we claim that the result also implies that any code with distance at least $p+\varepsilon$ allows for reliable communication over $q\mathrm{SC}_p$. (Contrast the $2p+\varepsilon$ distance that was needed for a similar result that was implied by Proposition 6.4.1.)

Finally, we present a lemma (the proof is left as an exercise) that will be crucial to the proof of Theorem 7.5.1.

Lemma 7.5.2. *Let be C be an $(n,k,d)_q$ code. If we fix the values in $n-d+1$ out of the n positions in a possible codeword, then at most one codeword in C can agree with the fixed values.*

Proof of Theorem 7.5.1. For the rest of the proof, fix a $\mathbf{c} \in C$. For notational convenience define $\mathscr{E}_S$ to be the set of all error patterns $\mathbf{e}$ such that $\mathbf{e}_S = 0$ and $wt(\mathbf{e}) = \rho n$. Note that as every error position has $(q-1)$ non-zero choices and there are ρn such positions in $[n] \setminus S$, we have

$$|\mathscr{E}_s| = (q-1)^{\rho n}. \tag{7.18}$$

Call an error pattern $\mathbf{e} \in \mathscr{E}_s$ as *bad* if there exists another codeword $\mathbf{c}' \neq \mathbf{c}$ such that

$$\triangle(\mathbf{c}', \mathbf{c}+\mathbf{e}) \leq \rho n.$$

Now, we need to show that the number of bad error patterns is at most

$$q^{-\Omega(\varepsilon n)}|\mathscr{E}_s|.$$

We will prove this by a somewhat careful counting argument.

We begin with a definition.

Definition 7.5.1. Every error pattern $\mathbf{e}$ is associated with a codeword $c(\mathbf{e})$, which is the closest codeword which lies within Hamming distance ρn from it.

For a bad error pattern we insist on having $c(\mathbf{e}) \neq \mathbf{c}$– note that for a bad error pattern such a codeword always exists. Let A be the set of positions where $c(\mathbf{e})$ agrees with $\mathbf{c}+e$.

The rest of the argument will proceed as follows. For each possible A, we count how many bad patterns $\mathbf{e}$ are associated with it (i.e. $\mathbf{c}+\mathbf{e}$ and $c(\mathbf{e})$ agree exactly in the positions in A). To bound this count non-trivially, we will use Lemma 7.5.2.

Define a real number α such that $|A| = \alpha n$. Note that since $c(\mathbf{e})$ and $\mathbf{c}+\mathbf{e}$ agree in at least $1-\rho$ positions,

$$\alpha \geq 1-\rho \geq 1-\delta+\varepsilon. \tag{7.19}$$

For now let us fix A with $|A| = \alpha n$ and to expedite the counting of the number of bad error patterns, let us define two more sets:

$$A_1 = A \cap S,$$

and

$$A_2 = A \setminus A_1.$$

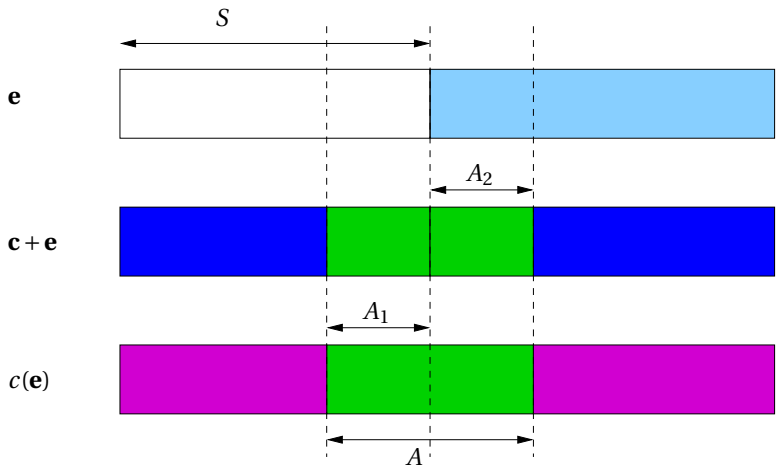


Figure 7.4: Illustration of notation used in the proof of Theorem 7.5.1. Positions in two different vectors that agree have the same color.

.

See Figure 7.4 for an illustration of the notation that we have fixed so far.

Define β such that

$$|A_1| = \beta n. \tag{7.20}$$

Note that this implies that

$$|A_2| = (\alpha - \beta)n. \tag{7.21}$$

Further, since $A_1 \subseteq A$, we have

$$\beta \leq \alpha.$$

To recap, we have argued that every bad error pattern $\mathbf{e}$ corresponds to a codeword $c(\mathbf{e}) \neq \mathbf{c}$ and is associated with a pair of subsets (A_1, A_2). So, we fix (A_1, A_2) and then count the number of bad $\mathbf{e}$'s that map to (A_1, A_2). (Later on we will aggregate this count over all possible choices of (A_1, A_2).)

Towards this end, first we overestimate the number of error patterns $\mathbf{e}$ that map to (A_1, A_2) by allowing such $\mathbf{e}$ to have arbitrary values in $[n] \setminus (S \cup A_2)$. Note that all such values have to be non-zero (because of (7.17). This implies that the number of possible distinct $\mathbf{e}_{[n]\setminus(S\cup A_2)}$ is at most

$$(q-1)^{n-|S|-|A_2|} = q^{n-(1-\rho)n-(\alpha-\beta)n}, \tag{7.22}$$

where the equality follows from the given size of S and (7.21). Next fix a non-zero $\mathbf{x}$ and let us only consider error patterns $\mathbf{e}$ such that

$$\mathbf{e}_{[n]\setminus(S\cup A_2)} = \mathbf{x}.$$

Note that at this stage we have an error pattern $\mathbf{e}$ as depicted in Figure 7.5.

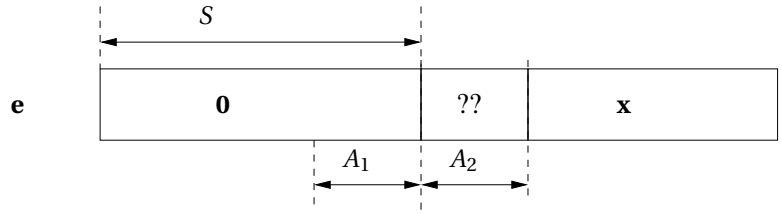


Figure 7.5: Illustration of the kind of error patterns we are trying to count now. The ? denote values that have not been fixed yet.

Now note that if we fix $c(\mathbf{e})_{A_2}$, then we would also fix $\mathbf{e}_{A_2}$ (as $(\mathbf{c}+\mathbf{e})_{A_2} = (c(\mathbf{e}))_{A_2}$). Recall that $\mathbf{c}$ is already fixed and hence, this would fix $\mathbf{e}$ as well. Further, note that

$$c(\mathbf{e})_{A_1} = (\mathbf{c}+\mathbf{e})_{A_1} = \mathbf{c}_{A_1}.$$

This implies that $c(\mathbf{e})_{A_1}$ is already fixed and hence, by Lemma 7.5.2 we would fix $c(\mathbf{e})$ if we fix (say the first) $(1-\delta)n+1-|A_1|$ positions in $c(\mathbf{e})_{A_2}$. Or in other words, by fixing the first $(1-\delta)n+1-|A_1|$ positions in $\mathbf{e}_{A_2}$, $\mathbf{e}$ would be completely determined. Thus, the number of choices for $\mathbf{e}$ that have the pattern in Figure 7.5 is upper bounded by

$$q^{(1-\delta)n+1-|A_1|} = (q-1)^{(1-\delta)n+1-\beta n}, \tag{7.23}$$

where the equality follows from (7.20).

Thus, by (7.22) and (7.23) the number of possible bad error patterns $\mathbf{e}$ that map to (A_1, A_2) is upper bounded by

$$(q-1)^{n-(1-\rho)n-\alpha n+\beta n+(1-\delta)n+1-\beta n} \leq (q-1)^{\rho n-\varepsilon n+1} = (q-1)^{-\varepsilon n+1}|\mathscr{E}_s|,$$

where the inequality follows from (7.19) and the equality follows from (7.18).

Finally, summing up over all choices of $A = (A_1, A_2)$ (of which there are at most 2^n), we get that the total number of bad patterns is upper bounded by

$$2^n \cdot (q-1)^{-\varepsilon n+1} \cdot |\mathscr{E}_S| \leq q^{\frac{n}{\log_2 q} - \frac{\varepsilon n}{2} + \frac{1}{2}} \cdot |\mathscr{E}_A| \leq q^{-\varepsilon n/4} \cdot |\mathscr{E}_S|,$$

where the first inequality follows from $q-1 \geq \sqrt{q}$ (which in turn is true for $q \geq 3$) while the last inequality follows from the fact that for $q \geq \Omega(1/\varepsilon)$ and large enough n, $\frac{n+1/2}{\log_2 q} < \frac{\varepsilon n}{4}$. This completes the proof. □

It can be shown that Theorem 7.5.1 is not true for $q = 2^{o(1/\varepsilon)}$. The proof is left as an exercise.

7.6 Exercises

Exercise 7.1. Show that with high probability, a random linear code is (ρ, L)-list decodable code as long as

$$R \leq 1 - H_q(\rho) - \frac{1}{\lceil \log_q(L+1) \rceil}. \tag{7.24}$$

Hint: Think how to fix (7.8) for random linear code.

Exercise 7.2. In this exercise we will see how we can "fix" the dependence on L is the rate of random linear codes from Exercise 7.1. In particular, we will consider the following family of codes that are somewhere between linear and general codes and are called pseudolinear codes, which are defined as follows.

Let q be a prime power and let $1 \le k \le n$ and $L \ge 1$ be integers. Then an (n,k,L,r,q)-family of pseudolinear codes is defined as follows. Let $\mathbf{H}$ be the parity check matrix of an $[q^k-1, q^k-1-r, L+1]_q$ linear code and $\mathbf{H}'$ be an extension of $\mathbf{H}$ with the first column being $\mathbf{0}$ (and the rest being $\mathbf{H}$). Every code in the family is indexed by a matrix $\mathbf{A} \in \mathbb{F}_q^{n \times r}$. Fix such a $\mathbf{A}$. Then the corresponding code $C_{\mathbf{A}}$ is defined as follows. For any $\mathbf{x} \in \mathbb{F}_q^k$, we have

$$C_{\mathbf{A}}(\mathbf{x}) = \mathbf{A} \cdot \mathbf{H}'_{\mathbf{x}},$$

where $\mathbf{H}'_{\mathbf{x}}$ is the column corresonding to $\mathbf{x}$, when though of as an integer between 0 and q^k-1.

Next, we will argue that random pseudolinear codes have near optimal list decodability:

1. Fix non-zero messages $\mathbf{m}_1, \dots \mathbf{m}_L$. Then for a random code $C_{\mathbf{A}}$ from an (n,k,L,r,q)-family of pseudolinear code family, the codewords $C_{\mathbf{A}}(\mathbf{m}_1), \dots, C_{\mathbf{A}}(\mathbf{m}_L)$ are *independent* random vectors in $\mathbb{F}_q^n$.

2. Define (n,k,L,q)-family of pseudolinear codes to be $(n,k,L,O(kL),q)$-family of pseudolinear codes. Argue that (n,k,L,q)-family of pseudolinear codes exist.
   Hint: Exercise 5.10 might be helpful.

3. Let $\varepsilon > 0$ and $q \ge 2$ be a prime power. Further let $0 \le \rho < 1 - 1/q$. Then for a large enough n and k such that
   $$\frac{k}{n} \ge 1 - H_q(\rho) - \frac{1}{L} - \varepsilon,$$
   a random (n,k,L,q)-pesudolinear code is (ρ, L)-list decodable.

4. Show that one can construct a (ρ, L)-list decodable pseudolinear code with rate at least $1 - H_q(\rho) - \frac{1}{L} - \varepsilon$ in $q^{O(kL+n)}$ time.
   Hint: Use method of conditional expectations.

Exercise 7.3. In this exercise we will consider a notion of “average" list decoding that is closely related to our usual notion of list decoding. As we will see in some subsequent exercises, sometimes it is easier to work with this average list decoding notion.

1. We begin with an equivalent definition of our usual notion of list decoding. Argue that a code C is (ρ, L) list decodable if and only if for every $\mathbf{y} \in [q]^n$ and every subset of $L+1$ codewords $\mathbf{c}_0, \dots, \mathbf{c}_L$ we have that
   $$\max_{0 \le i \le L} \Delta(\mathbf{y}, \mathbf{c}_i) > \rho n.$$

2. We define a code C to be (ρ, L)-*average list decodable* if for every $\mathbf{y} \in [q]^n$ and $L+1$ codewords $\mathbf{c}_0, \ldots, \mathbf{c}_L$ we have

$$\frac{1}{L} \cdot \sum_{i=0}^{L} \Delta(\mathbf{y}, \mathbf{c}_i) > \rho n.$$

   Argue that if C is (ρ, L)-average list decodable then it is also (ρ, L)-list decodable.

3. Argue that if C is (ρ, L)-list decodable then it is also $(\rho(1-\gamma), \lceil L/\gamma \rceil)$-average list decodable (for any $0 < \gamma < \rho$).

Exercise 7.4. In Section 7.5 we saw that for any code one can correct arbitrarily close to relative distance fraction of random errors. In this exercise we will see that one can prove a weaker result. In particular let $\mathscr{D}$ be an arbitrary distribution on $B_q(\mathbf{0}, \rho n)$. Then argue that for most codes, the list size with high probability is 1. In other words, show that for $1 - o(1)$, fraction of codes C we have that for every codeword $\mathbf{c} \in C$

$$\Pr_{\mathbf{e} \leftarrow \mathscr{D}} \left[|B_q(\mathbf{c} + \mathbf{e}, \rho n) \cap C| > 1 \right] = o(1).$$

Hint: Adapt the proof of Theorem 6.3.1 from Section 6.3.2.

Exercise 7.5. We call a code (ρ, L)-*erasure list-decodable* is informally for any received word with at most ρ fraction of erasures at most L codewords agree with it in the unerased positions. More formally, an $(n, k)_q$-code C is (ρ, L)-erasure list-decodable if for every $\mathbf{y} \in [q]^{(1-\rho)n}$ and every subset $T \subseteq [n]$ with $|T| = (1-\rho)n$, we have that

$$\left| \{ \mathbf{c} \in C | \mathbf{c}_T = \mathbf{y} \} \right| \leq L.$$

In this exercise you will prove some simple bounds on the best possible rate for erasure-list decodable code.

1. Argue that if C has distance d then it is $\left(\frac{d-1}{n}, 1\right)$-erasure list decodable.

2. Show that there exists a (ρ, L)-erasure list decodable code of rate

$$\frac{L}{L+1} \cdot (1-\rho) - \frac{H_q(\rho)}{L} - \gamma,$$

   for any $\gamma > 0$.

3. Argue that there exists a *linear* (ρ, L)-erasure list decodable code with rate

$$\frac{J-1}{J} \cdot (1-\rho) - \frac{H_q(\rho)}{J-1} - \gamma,$$

   where $J = \left\lceil \log_q(L+1) \right\rceil$ and $\gamma > 0$.

4. Argue that the bound in item 2 is tight for large enough L by showing that if a code of rate $1 - \rho + \varepsilon$ is (ρ, L)-erasure list decodable then L is $2^{\Omega_\varepsilon(n)}$.

Exercise 7.6. In this exercise we will see an alternate characterization of erasure list-decodable code for linear codes, which we will use to show separation between linear and non-linear code in the next exercise.

Given a linear code $C \subseteq \mathbb{F}_q^n$ and an integer $1 \leq r \leq n$, define the r'th *generalized Hamming distance*, denoted by $d_r(C)$, as follows. First given a set $D \subseteq \mathbb{F}_q^N$, we define the *support* of D as the union of the supports of vectors in D. More precisely

$$\mathrm{supp}(D) = \{i | \text{ there exists } (u_1, \ldots, u_n) \in S \text{ such that } u_i \neq 0\}.$$

Then $d_r(C)$ is size of the smallest support of all r-dimensional subcodes of C.

Argue the following:

1. (Warmup) Convince yourself that $d_1(C)$ is the usual Hamming distance of C.

2. Prove that C is $(\rho n, L)$-erasure list-decodable if and only if $d_{1+\lfloor \log_q L \rfloor}(C) > \rho n$.

Exercise 7.7. In this exercise we use the connection between generalized Hamming distance and erasure list decodability from Exercise 7.6 to show an "exponential separation" between linear and non-linear codes when it comes to list decoding from erasure.

Argue the following:

1. Let C be an $[n,k]_q$ code. Then show that the average support size of r-dimensional subcodes of C is exactly
$$\frac{q^r - 1}{q^r} \cdot \frac{|C|}{|C| - 1} \cdot n.$$

2. From previous part or otherwise, conclude that if for an $[n,k]_q$ code C we have $d_r(C) > n(1 - q^{-r})$, then we have
$$|C| \leq \frac{d_r(C)}{d_r(C) - n(1 - q^{-r})},$$
Note that the above bound for $r = 1$ recovers the Plotkin bound (second part of Theorem 4.4.1).

3. Argue that any (family) of code C with $d_r(C) = \delta_r \cdot n$, its rate satisfies:
$$R(C) \leq 1 - \frac{q^r}{q^r - 1} \cdot \delta_r + o(1).$$
Hint: Use a the result from previous part on a code related to C.

4. Argue that for small enough $\varepsilon > 0$, any linear $(1 - \varepsilon, L)$-erasure list decodable code with positive rate must have $L \geq \Omega(1/\varepsilon)$.

5. Argue that there exist $(1 - \varepsilon, O(\log(1/\varepsilon)))$-erasure list decodable code with positive (in fact $\Omega(\varepsilon)$) rate. Conclude that there exists non-linear codes that have the same erasure list decodability but with exponentially smaller list sizes than linear codes.

Exercise 7.8. In this exercise we will prove an analog of the Johnson bound (Theorem 7.3.1) but for erasure list-decodable codes. In particular, let C be an $(n,k,\delta n)_q$ code. Then show that for any $\varepsilon > 0$, C is an $\left(\left(\frac{q}{q-1}-\varepsilon\right)\delta, \frac{q}{(q-1)\varepsilon}\right)$-erasure list decodable.

Hint: The Plotkin bound (Theorem 4.4.1) might be useful.

Exercise 7.9. Let C be a q-ary (ρ, L)-(average) list decodable of rate R, then show that there exists another (ρ, L)-(average) list decodable code with rate at least

$$R + H_q(\lambda) - 1 - o(1),$$

for any $\lambda \in (\rho, 1-1/q]$ such that *all* codewords in C' have Hamming weight *exactly* λn.

Hint: Try to translate C.

Exercise 7.10. In this exercise, we will prove a lower bound on the list size of list decodable codes that have optimal rate. We do this via a sequence of following steps:

1. Let $C \subseteq [q]^n$ be a $(\rho, L-1)$-list decodable code such that all codewords have Hamming weight exactly λn for
$$\lambda = \rho + \frac{1}{2L} \cdot \rho^L.$$
Then prove that
$$|C| < \frac{2L^2}{\lambda^L}.$$

   Hint: It might be useful to use the following result due to Erdös [20] (where we choose the variables to match the relevant ones in the problem). Let $\mathcal{A}$ be a family of subsets of $[n]$. Then if every $A \in \mathcal{A}$ has size at least $2L^2/\lambda^L$, then there exist distinct $A_1, \ldots, A_L \in \mathcal{A}$ such that $\cap_{i=1}^{L} A_i$ has size at least $\frac{n\lambda^L}{2}$.

2. Argue that any q-ary $(\rho, L-1)$-list decodable code C (for large enough block length) has rate at most $1 - H_q(\rho) - b_{\rho,q} \cdot \frac{\rho^L}{L}$ for some constant $b_{\rho,q}$ that only depends on ρ and q.
   Hint: Use the previous part and Exercise 7.9.

3. Argue that any q-ary (ρ, L)-list decodable C with rate $1-H_q(\rho)-\varepsilon$ mush satisfy $L \geq \Omega_{\rho,q}(\log(1/\varepsilon))$.

Exercise 7.11. It follows from Theorem 7.4.1 that a random code of rate $1 - H_q(\rho) - \varepsilon$ with high probability is $(\rho, O(1/\varepsilon))$-list decodable. On the other hand, the best lower bound on the list size for codes of rate $1 - H_q(\rho) - \varepsilon$ (for constant p, q) is $\Omega(\log(1/\varepsilon))$ (as we just showed in Exercise 7.10). It is natural to wonder if one can perhaps do a better argument for random codes. In this exercise, we will show that our argument for random codes is the best possible (for random codes). We will show this via the following sequence of steps:

1. Let C be a random $(n,k)_q$ code of rate $1 - H_q(\rho) - \varepsilon$. For any $\mathbf{y} \in [q]^n$ and any subset $S \subseteq [q]^k$ of size $L+1$, define the random event $\mathcal{E}(\mathbf{y}, S)$ that for every $\mathbf{m} \in S$, $C(\mathbf{m})$ is at Hamming distance at most ρn from $\mathbf{y}$. Define
$$W = \sum_{\mathbf{y}, S} \mathcal{E}(\mathbf{y}, S).$$
Argue that C is (ρ, L)-list decodable if and only if $W = 0$.

2. Define
$$\mu = q^{-n} \cdot Vol_q(\rho n, n).$$
Argue that
$$\mathbb{E}[W] \geq \frac{1}{(L+1)^{L+1}} \cdot \mu^{L+1} \cdot q^n \cdot q^{k(L+1)}.$$

3. Argue that
$$\sigma^2(Z) \leq q^{2n} \cdot \sum_{\ell=1}^{L+1} (L+1)^{2(L+1)} \cdot q^{k(2L+2-\ell)} \cdot \mu^{2L-\ell+3}.$$
Hint: Analyze the probability of both events $\mathcal{E}(\mathbf{y}, S)$ and $\mathcal{E}(\mathbf{z}, T)$ happening together for various intersection sizes $\ell = |S \cap T|$.

4. Argue that C is $\left(\rho, \frac{1-H_q(\rho)}{2\varepsilon}\right)$-list decodable with probability at most $q^{-\Omega_{\rho,\varepsilon}(n)}$.
Hint: Use Chebyschev's inequality.

7.7 Bibliographic Notes

List decoding was defined by Elias [19] and Wozencraft [78].

The result showing that for random error patterns, the list size with high probability is one for the special case of Reed-Solomon codes was shown by McEliece [55]. The result for all codes was proved by Rudra and Uurtamo [64]

In applications of list decoding in complexity theory (see for example [72],[27, Chap. 12]), side information is used crucially to prune the output of a list decoding algorithm to compute a unique answer.

Guruswami [26] showed that the answer to Question 7.3.1 is yes in the sense that there exist linear codes with relative distance δ such that we can find Hamming ball of radius larger than $J_q(\delta)$ with super-polynomially many codewords. This result was proven under a number-theoretic assumption, which was later removed by [36].

(7.24) implies that there exist linear codes with rate $1 - H_q(\rho) - \varepsilon$ that are $\left(\rho, q^{O(1/\varepsilon)}\right)$-list decodable. (This is also true for most linear codes with the appropriate parameters.) However, for a while just for $q = 2$, we knew the *existence* of $\left(\rho, O(1/\varepsilon)\right)$-list decodable codes [30] (though it was not a high probability result). Guruswami, Håstad and Kopparty resolved this "gap" by showing that random linear codes of rate $1 - H_q(\rho) - \varepsilon$ are $(\rho, O(1/\varepsilon))$-list decodable (with high probability) [29].

Chapter 8

What Cannot be Done-II

In this brief interlude of a chapter, we revisit the trade-offs between rate and relative distance for codes. Recall that the best (and only) lower bound on R that we have seen is the GV bound and the best upper bound on R that we have have seen so far is a combination of the Plotkin and Hamming bounds (see Figure 4.5). In this chapter, we will prove the final upper bound on R in this book due to Elias and Bassalygo. Then we will mention the best known upper bound on rate (but without stating or proving it). Finally, we will conclude by summarizing what we have seen so far and laying down the course for the rest of the book.

8.1 Elias-Bassalygo bound

We begin with the statement of a new upper bound on the rate called the Elias-Bassalygo bound.

Theorem 8.1.1 (Elias-Bassalygo bound)**.** *Every q-ary code of rate R, distance δ, and large enough block length, satisfies the following:*

$$R \leq 1 - H_q\left(J_q(\delta)\right) + o(1)$$

See Figure 8.1 for an illustration of the Elias-Bassalygo bound for binary codes. Note that this bound is tighter than all the previous upper bounds on rate that we have seen so far.

The proof of Theorem 8.1.1 uses the following lemma:

Lemma 8.1.2. *Given a q-ary code, $C \subseteq [q]^n$ and $0 \leq e \leq n$, there exists a Hamming ball of radius e with at least $\frac{|C| Vol_q(e,n)}{q^n}$ codewords in it.*

Proof. We will prove the existence of the required Hamming ball by the probabilistic method. Pick a received word $\mathbf{y} \in [q]^n$ at random. It is easy to check that the expected value of $|B(\mathbf{y}, e) \cap C|$ is $\frac{|C| Vol_q(e,n)}{q^n}$. (We have seen this argument earlier in the proof of part (ii) of Theorem 7.4.2.)

This by the probabilistic method implies the existence of a $\mathbf{y} \in [q]^n$ such that

$$|B(\mathbf{y}, e) \cap C| \geq \frac{|C|\, Vol_q(e, n)}{q^n},$$

as desired. □

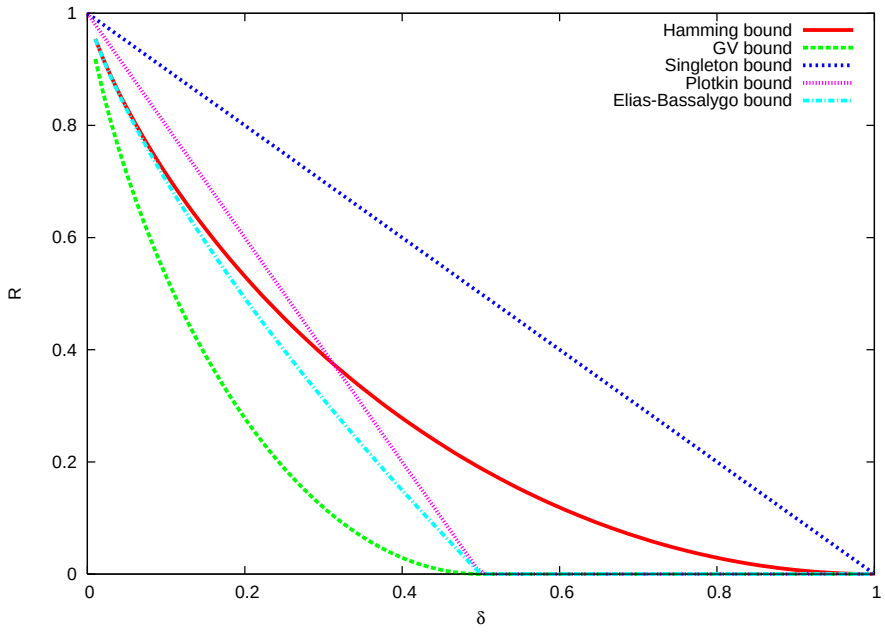


Figure 8.1: Singleton, Hamming, Plotkin, GV and Elias-Bassalygo bounds on rate versus distance for binary codes.

Proof of Theorem 8.1.1. Let $C \subseteq [q]^n$ be any code with relative distance δ. Define $e = nJ_q(\delta) - 1$. By Lemma 8.1.2, there exists a Hamming ball with $\mathscr{B}$ codewords such that the following inequality is true:

$$\mathscr{B} \geq \frac{|C| \, Vol_q(e,n)}{q^n}.$$

By our choice of e and the Johnson bound (Theorem 7.3.1), we have

$$\mathscr{B} \leq qdn.$$

Combining the upper and lower bounds on $\mathscr{B}$ implies the following

$$|C| \leq qnd \cdot \frac{q^n}{Vol_q(e,n)} \leq q^{n\left(1-H_q\left(J_q(\delta)\right)+o(1)\right)},$$

where the second inequality follows from our good old lower bound on the volume of a Hamming ball (Proposition 3.3.1) and the fact that $qdn \leq qn^2 \leq q^{o(n)}$ for large enough n. This implies that the rate R of C satisfies:

$$R \leq 1 - H_q\left(J_q(\delta)\right) + o(1),$$

as desired. □

8.2 The MRRW bound: A better upper bound

The MRRW bound (due to McEliece, Rodemich, Rumsey and Welch) is based on a linear programming approach introduced by Delsarte to bound the rate of a code. The MRRW bound is a better upper bound than the Elias-Bassalygo bound (though we will not state or prove the bound in this book). However, there is a gap between the Gilbert-Varshamov (GV) bound and the MRRW bound. The gap still exists to this day. To give one data point on the gap, consider $\delta = \frac{1}{2} - \varepsilon$ (think of $\varepsilon \to 0$), the GV bound gives a lower bound on R of $\Omega\left(\varepsilon^2\right)$ (see Proposition 3.3.5), while the MRRW bound gives an upper bound on R of $O\left(\varepsilon^2 \log\left(\frac{1}{\varepsilon}\right)\right)$.

8.3 A Breather

Let us now recap the combinatorial results that we have seen so far. Table 8.1 summarizes what we have seen so far for binary codes in Shannon's world and Hamming's world (under both unique and list decoding settings).

Shannon	Hamming	
BSC_p	Unique	List
1-H$\left(p\right)$ is capacity	$R \geq 1 - H(\delta)$	$1 - H\left(p\right)$ is list decoding capacity
	$R \leq MRRW$	
Explicit codes at capacity?	Explicit Asymptotically good codes?	Explicit codes at capacity?
Efficient decoding algorithm?	Efficient decoding algorithms?	Efficient decoding algorithms?

Table 8.1: High level summary of results seen so far.

For the rest of the section, we remind the reader about the definition of explicit codes (Definition 6.3.1) and strongly explicit codes (Definition 6.3.2).

We begin with BSC_p. We have seen that the capacity of BSC_p is $1 - H(p)$. The most natural open question is to obtain the capacity result but with explicit codes along with efficient decoding (and encoding) algorithms (Question 6.3.1).

Next we consider Hamming's world under unique decoding. For large enough alphabets, we have seen that Reed-Solomon codes (Chapter 5) meet the Singleton bound (Theorem 4.3.1). Further, the Reed-Solomon codes are strongly explicit[1]. The natural question then is

Question 8.3.1. *Can we decode Reed-Solomon codes up to half its distance?*

For smaller alphabets, especially binary codes, as we have seen in the last section, there is a gap between the best known lower and upper bounds on the rate of a code with a given relative

[1] The proof is left as an exercise.

distance. Further, we do not know of an explicit construction of a binary code that lies on the GV bound. These lead to the following questions that are still wide open:

Open Question 8.3.1. *What is the optimal trade-off between R and δ?*

Open Question 8.3.2.
Does there exist an explicit construction of (binary) codes on the GV bound?

If we scale down our ambitions, the following is a natural weaker version of the second question above:

Question 8.3.2. *Do there exist explicit asymptotically good binary codes?*

We also have the following algorithmic counterpart to the above question:

Question 8.3.3. *If one can answer Question 8.3.2, then can we decode such codes efficiently from a non-zero fraction of errors?*

For list decoding, we have seen that the list decoding capacity is $1 - H_q(p)$. The natural open questions are whether we can achieve the capacity with explicit codes (Question 7.4.1) along with efficient list decoding algorithms (Question 7.4.2).

In the remainder of the book, we will focus on the questions mentioned above (and summarized in the last two rows of Table 8.1).

8.4 Bibliographic Notes

The McEliece-Rodemich-Rumsey-Welch (MRRW) bound was introduced in 1977 in the paper [56].

Part III

The Codes

Chapter 9

When Polynomials Save the Day: Polynomial Based Codes

As we saw in Chapter 5, The Reed-Solomon codes give a remarkable family of codes with optimal dimension vs. distance tradeoff. They even match the Singleton bound (recall Theorem 4.3.1), get $k = n - d + 1$ for a code of block length n, distance d and dimension k. However they achieve this remarkable performance only over large alphabets, namely when the alphabet size $q \geq n$. In fact, so far in this book, we have not seen *any explicit* asymptotically good code other than a Reed-Solomon code. This naturally leads to the following question (which is a weaker form for Question 8.3.2):

Question 9.0.1. *Do there exist explicit asymptotically good codes for small alphabets $q \ll n$?*

In this chapter we study an extension of Reed-Solomon codes, called the (generalized) Reed-Muller codes, that lead to codes over smaller alphabets while losing in the dimension-distance tradeoff (but under certain settings do answer Question 9.0.1 in the affirmative).

The main idea is to extend the notion of functions we work with, to multivariate functions. (See Exercise 5.2 for equivalence between certain Reed-Solomon codes and univariate functions.) Just working with bivariate functions (functions on two variables), allows us to get codes of block length $n = q^2$, and more variables can increase the length further for the same alphabet size. We look at functions of *total degree* at most r. Analysis of the dimension of the code reduces to simple combinatorics. Analysis of the distance follows from "polynomial-distance" lemmas, whose use is ubiquitous in algebra, coding theory and computer science, and we describe these in the sections below. We start with the generic construction.

9.1 The generic construction

Recall that for a monomial $\mathbf{X}^{\mathbf{d}} = X_1^{d_1} \cdot X_2^{d_2} \cdots X_m^{d_m}$ its total degree is $d_1 + d_2 + \cdots + d_m$. We next extend this to the definition of the degree of a polynomial:

Definition 9.1.1. The total degree of a polynomial $P(\mathbf{X}) = \sum_{\mathbf{d}} c_{\mathbf{d}} \mathbf{X}^{\mathbf{d}}$ over $\mathbb{F}_q$ (i.e. every $c_{\mathbf{d}} \in \mathbb{F}_q$) is the maximum over $\mathbf{d}$ such that $c_{\mathbf{d}} \neq 0$, of the total degree of $\mathbf{X}^{\mathbf{d}}$. We denote the total degree of P by $\deg(P)$.

For example, the degree of the polynomial $3X^3Y^4 + X^5 + Y^6$ is 7.

In turns out that when talking about Reed-Muller codes, it is convenient to switch back and forth between multivariate functions and multivariate polynomials. We can extend the notion above to functions from $\mathbb{F}_q^m \to \mathbb{F}_q$. For $f : \mathbb{F}_q^m \to \mathbb{F}_q$ let $\deg(f)$ be the minimal degree of a polynomial $P \in \mathbb{F}_q[X_1, \ldots, X_m]$ (where $\mathbb{F}_q[X_1, \ldots, X_m]$ denotes the set of all m-variate polynomials with coefficients from $\mathbb{F}_q$) such that $f(\alpha) = P(\alpha)$ for every $\alpha \in \mathbb{F}_q^m$. Note that since (by Exercise 2.3) for every $a \in \mathbb{F}_q$ we have $a^q - a = 0$, it follows that a minimal degree polynomial does not contain monomials with degree more than $q-1$ any single variable. In what follows,

Definition 9.1.2. We use $\deg_{X_i}(p)$ to denote the degree of polynomial p in variable X_i and $\deg_{X_i}(f)$ to denote the degree of (the minimal polynomial corresponding to) a function f in variable X_i.

For example $\deg_X(3X^3Y^4 + X^5 + Y^6) = 5$ and $\deg_Y(3X^3Y^4 + X^5 + Y^6) = 6$. Further, in this notation we have for every function $f : \mathbb{F}_q^m \to \mathbb{F}_q$, $\deg_{X_i}(f) \leq q-1$ for every $i \in [m]$.

Reed-Muller codes are given by three parameters: a prime power q and positive integers m and r, and consist of the evaluations of m-variate polynomials of degree at most r over all of the domain $\mathbb{F}_q^m$.

Definition 9.1.3 (Reed-Muller Codes)**.** The Reed-Muller code with parameters q, m, r, denoted $\mathrm{RM}(q, m, r)$, is the set of evaluations of all m-variate polynomials in $\mathbb{F}_q[X_1, \ldots, X_m]$ of total degree at most r and individual degree at most $q-1$ over all points in $\mathbb{F}_q^m$. Formally

$$\mathrm{RM}(q, m, r) \stackrel{\text{def}}{=} \left\{ f : \mathbb{F}_q^m \to \mathbb{F}_q \,|\, \deg(f) \leq r \right\}.$$

For example consider the case of $m = q = 2$ and $r = 1$. Note that all bivariate polynomials over $\mathbb{F}_2$ of degree at most 1 are 0, 1, X_1, X_2, $1 + X_1$, $1 + X_2$, $X_1 + X_2$ and $1 + X_1 + X_2$. Thus, we have that (where the evaluation points for (X_1, X_2) are ordered as $(0,0), (0,1), (1,0), (1,1)$):

$$\mathrm{RM}(2,2,1) = \{(0,0,0,0), (1,1,1,1), (0,0,1,1), (0,1,0,1), (1,1,0,0), (1,0,1,0), (0,1,1,0), (1,0,0,1)\}.$$

Also note that $\mathrm{RM}(q, m, 1)$ is almost the Hadamard code (see Exercise 5.6).

The Reed-Muller code with parameters (q, m, r) clearly has alphabet $\mathbb{F}_q$ and block length $n = q^m$. Also it can be verified that $\mathrm{RM}(q, m, r)$ is a linear code (see Exercise 9.1.) This leads to the following question, which will be the primary focus of this chapter:

Question 9.1.1. *What are the dimension and distance of an* $\mathrm{RM}(q,m,r)$ *code?*

The dimension of the code is the number of m-variate monomials of degree at most r, with the condition that degree in each variable is at most $q-1$. No simple closed form expression for this that works for all choices of q, m and r is known, so we will describe the effects only in some cases. The distance analysis of these codes takes a little bit more effort and we will start with two simple settings before describing the general result.

9.2 The low degree case

We start by considering $\mathrm{RM}(q,m,r)$ when $r < q$, i.e., the degree is smaller than the field size. We refer to this setting as the "low-degree" setting.

Dimension. The dimension of $\mathrm{RM}(q,m,r)$ in the low-degree case turns out to have a nice closed form, since we do not have to worry about the constraint that each variable has degree at most $q-1$: this is already imposed by restricting the total degree to at most $r \leq q-1$. This leads to a nice expression for the dimension:

Proposition 9.2.1. *The dimension of the Reed Muller code* $\mathrm{RM}(q,m,r)$ *equals* $\binom{m+r}{r}$ *when* $r < q$.

Proof. The dimension equals the size of the set

$$D = \left\{ (d_1, \ldots, d_m) \in \mathbb{Z}^m | d_i \geq 0 \text{ for all } i \in [m], \sum_{i=1}^{m} d_i \leq r \right\}, \tag{9.1}$$

since for every $(d_1, \ldots, d_m) \in D$, the monomial $X_1^{d_1} \cdots X_m^{d_m}$ is a monomial of degree at most r and these are all such monomials. The closed form expression for the dimension follows by a simple counting argument. (See Exercise 9.2). □

Distance. Next we turn to the analysis of the distance of the code. To understand the distance we will first state and prove a simple fact about the number of zeroes a multivariate polynomial can have. (We will have three versions of this in this chapter - with the third subsuming the first (Lemma 9.2.2) and second (Lemma 9.3.1), but the first two will be slightly simpler to state and remember.)

Lemma 9.2.2 (Polynomial Zero Lemma (low-degree case))**.** *Let* $f \in \mathbb{F}_q[X_1, \ldots, X_m]$ *be a non-zero polynomial with* $\deg(f) \leq r$. *Then the fraction of zeroes of* f *is at most* $\frac{r}{q}$, *i.e.,*

$$\frac{|\{\mathbf{a} \in \mathbb{F}_q^m | f(\mathbf{a}) = 0\}|}{q^m} \leq \frac{r}{q}.$$

We make couple of remarks. First note that the above lemma for $m = 1$ is the degree mantra (Proposition 5.2.3). We note that for every $m \geq 1$ the above lemma is tight (see Exercise 9.3). However, there exists polynomials for which the lemma is not tight (see Exercise 9.4).

Proof of Lemma 9.2.2. Note that the lemma statement is equivalent to saying that the probability that $f(\mathbf{a}) = 0$ is at most $\frac{\deg(f)}{q}$ when $\mathbf{a} = (a_1, \ldots, a_m)$ is chosen uniformly at random from $\mathbb{F}_q^m$. We claim that this holds by induction on m.

We will prove the lemma by induction on $m \geq 1$. Note that the base case follows from the degree mantra (Proposition 5.2.3). Now consider the case of $m > 1$ (and we assume that the lemma is true for $m-1$). To apply inductive hypothesis we first write f as a polynomial in X_m with coefficients that are themselves polynomials in $X_1, \ldots, X_{m-1}$. So let

$$f = f_0 X_m^0 + f_1 X_m^1 + \ldots f_t X_m^t,$$

where each $f_i(X_1, \ldots, X_{m-1})$ is a polynomial from $\mathbb{F}_q[X_1, \ldots, X_{m-1}]$ and $\deg(f_i) \leq r - i$. Furthermore let t be the largest index such that f_t is not zero. Now we consider picking $\mathbf{a} \in \mathbb{F}_q^m$ in two steps: We first pick $(a_1, \ldots, a_{m-1})$ uniformly at random from $\mathbb{F}_q^{m-1}$, and then we pick a_m uniformly from $\mathbb{F}_q$. Let

$$f^{(a_1,\ldots,a_{m-1})}(X_m) = f_0(a_1, \ldots, a_{m-1}) X_m^0 + \cdots + \ldots f_t(a_1, \ldots, a_{m-1}) X_m^t.$$

We consider two possible events:

$$\mathscr{E}_1 = \{(a_1, \ldots, a_m) | f_t(a_1, \ldots, a_{m-1}) = 0\}$$

and

$$\mathscr{E}_2 = \{((a_1, \ldots, a_m) | f_t(a_1, \ldots, a_{m-1}) \neq 0 \text{ and } f^{(a_1,\ldots,a_{m-1})}(a_m) = 0\}.$$

By the inductive hypothesis, we have that

$$\Pr[\mathscr{E}_1] \leq \frac{r-t}{q}, \tag{9.2}$$

since $\deg(f_t) \leq r - t$ and $f_t \neq 0$.

For every $(a_1, \ldots, a_{m-1}) \in \mathbb{F}_q^{m-1}$ such that $f_t(a_1, \ldots, a_{m-1}) \neq 0$ we also have that the univariate polynomial $f^{(a_1,\ldots,a_{m-1})}(X_m)$ is non-zero and of degree at most t, and so by the degree mantra it has at most t roots. It follows that for every such $(a_1, \ldots, a_{m-1})$ the probability, over a_m, that $f^{(a_1,\ldots,a_{m-1})}(a_m) = 0$ is at most $\frac{t}{q}$. In turn, it now immediately follows that

$$\Pr[\mathscr{E}_2] \leq \frac{t}{q}. \tag{9.3}$$

Finally, we claim that if neither $\mathscr{E}_1$ nor $\mathscr{E}_2$ occur, then $f(\mathbf{a}) \neq 0$. This is immediate from the definitions of $\mathscr{E}_1$ and $\mathscr{E}_2$, since if $f(a_1, \ldots, a_m) = 0$, it must either be the case that $f_t(a_1, \ldots, a_{m-1}) = 0$ (corresponding to $\mathscr{E}_1$) or it must be that $f_t(a_1, \ldots, a_{m-1}) \neq 0$ and $f^{(a_1,\ldots,a_{m-1})}(a_m) = 0$ (covered by $\mathscr{E}_2$). Note that this implies that $\Pr_{\mathbf{a}}[f(\mathbf{a}) = 0] \leq \Pr[\mathscr{E}_1 \cup \mathscr{E}_2]$. The lemma now follows from the fact that

$$\Pr_{\mathbf{a}}[f(\mathbf{a}) = 0] \leq \Pr[\mathscr{E}_1 \cup \mathscr{E}_2] \leq \Pr[\mathscr{E}_1] + \Pr[\mathscr{E}_2] \leq \frac{r}{q},$$

where the second inequality follows from the union bound (Proposition 3.1.3) and the final inequality follows from (9.2) and (9.3). □

Comparison with other codes

The lemmas above, while quite precise may not be fully transparent in explaining the asymptotics of the performance of the Reed-Muller codes, or contrast them with other codes we have seen. We mention a few basic facts here to get a clearer comparison.

If we set $m = 1$ and $r = k-1$, then we get the Reed-Solomon codes evaluated on all of F_q (see Chapter 5). If we set $m = k-1$, $r = 1$ and $q = 2$, then we get family of extended Hadamard codes (extended by including all Hadamard codewords and their complements). For more on this, see Exercise 5.6.

Thus Reed-Muller codes generalize some previously known codes - some with large alphabets and some with small alphabets. Indeed if we wish the alphabet to be small compared to the block length, then we can pick m to be a constant. For instance if we choose $m = 2$, we get codes of length n over an alphabets of size $\sqrt{n}$, while for any choice of relative distance δ, the code has rate $\frac{(1-\delta)^2}{2}$. In general for larger values of m, the code has alphabet size $n^{1/m}$ and rate $\frac{(1-\delta)^m}{m!}$. (See Exercise 9.5.) Thus for small values of m and fixed positive distance $\delta < 1$ there is a rate $R > 0$ such that, by choosing q appropriately large, one get codes on infinitely long block length n and alphabet $n^{1/m}$ with rate R and distance δ, which answers Question 9.0.1 in the affirmative.

This is one of the simplest such families of codes with this feature. We will do better in later in the book (e.g. Chapter 10), and indeed get alphabet size q independent of n with $R > 0$ and $\delta > 0$. But for now this is best we have.

9.3 The case of the binary field

Next we turn to a different extreme of parameter choices for the Reed-Muller codes. Here we fix the alphabet size $q = 2$ and see what varying m and r gets us.

Since we will prove a stronger statement later in Lemma 9.4.1, we only state the distance of the code $\mathrm{RM}(2, m, r)$ below, leaving the proof to Exercise 9.6.

Lemma 9.3.1 (Polynomial distance (binary case))**.** *Let f be a non-zero polynomial from $\mathbb{F}_2[X_1, \ldots, X_m]$ with $\deg_{X_i}(f) \leq 1$ for every $i \in [m]$. Then $|\{\mathbf{a} \in \mathbb{F}_2^m | f(\mathbf{a}) \neq 0\}| \geq 2^{m-\deg(f)}$.*

Further, it can be established that the bound in Lemma 9.3.1 is tight (see Exercise 9.7).

The dimension of the code is relatively straightforward to analyze. The dimension is again given by the number of monomials of degree at most r. Since the degree in each variable is either zero or one, this just equals the number of subsets of $[m]$ of size at most r. Thus we have:

Proposition 9.3.2. *For any $r \leq m$, the dimension of the Reed-Muller code $\mathrm{RM}(2, m, r)$ is exactly $\sum_{i=0}^{r} \binom{m}{i}$.*

Lemma 9.3.1 and Proposition 9.3.2 imply the following result:

Theorem 9.3.3. *For every $r \leq m$, the Reed-Muller code $\mathrm{RM}(2, m, r)$ is a code of block length 2^m, dimension $\sum_{i=0}^{r} \binom{m}{i}$ and distance 2^{m-r}.*

Again, to get a sense of the asymptotics of this code, we can fix $\tau > 0$ and set $r = \tau \cdot m$ and let $m \to \infty$. In this case we get a code of block length n (for infinitely many n) with rate roughly $n^{H(\tau)-1}$ and distance $n^{-\tau}$ (see Exercise 9.8). So both the rate and the distance tend to zero at a rate that is a small polynomial in the block length but the code has a constant sized alphabet. (Note that this implies that we have made some progress towards answering Question 8.3.2.)

9.4 The general case

We now turn to the general case, where q is general and r is allowed to be larger than $q-1$. We will try to analyze the dimension and distance of this code. The distance turns out to still have a clean expression, so we will do that first. The dimension does not have a simple expression describing it exactly, so we will give a few lower bounds that may be generally useful (and are often asymptotically tight).

9.4.1 The general case: Distance

Lemma 9.4.1 (Polynomial distance (general case))**.** *Let f be a non-zero polynomial from $\mathbb{F}_q[X_1, \ldots, X_m]$ with $\deg_{X_i}(f) \leq q-1$ for every $i \in [m]$ and $\deg(f) \leq r$. Furthermore, let s, t be the unique non-negative integers such that $t \leq q-2$ and*

$$s(q-1) + t = r.$$

Then

$$|\{\mathbf{a} \in \mathbb{F}_q^m \mid f(\mathbf{a}) \neq 0\}| \geq (q-t) \cdot q^{m-s-1} \geq q^{m-\frac{r}{q-1}}.$$

Hence, $\mathrm{RM}(q, m, r)$ *has distance at least* $q^{m-\frac{r}{q-1}}$.

Before proving the lemma we make a few observations: The above lemma clearly generalizes both Lemma 9.2.2 (which corresponds to the case $s = 0$) and Lemma 9.3.1 (where $q = 2$, $s = r-1$ and $t = 1$). In the general case the second lower bound is a little simpler and it shows that the probability that a polynomial is non-zero at a uniformly chosen point in $\mathbb{F}_q^m$ is at least $q^{-r/(q-1)}$. Finally, we note that Lemma 9.4.1 is tight for all settings of parameters (see Exercise 9.9).

Proof of Lemma 9.4.1. The proof is similar to the proof of Lemma 9.2.2 except we take advantage of the fact that the degree in a single variable is at most $q-1$. We also need to prove some simple inequalities.

As in the proof of Lemma 9.2.2 we prove that for a random choice of $\mathbf{a} = (a_1, \ldots, a_m) \in \mathbb{F}_q^m$, the probability that $f(\mathbf{a}) \neq 0$ is at least

$$(q-t) \cdot q^{-(s+1)}. \tag{9.4}$$

Note that in contrast to the proof of Lemma 9.2.2 we focus on the good events — the polynomial being non-zero — rather than on the bad events.

We prove the lemma by induction on m. In the case of $m = 1$ we have by the degree mantra (Proposition 5.2.3) that the probability that $f(a_1) \neq 0$ is at least $\frac{q-r}{q}$. If $r < q-1$ we have $s = 0$ and $t = r$ and so the expression in (9.4) satisfies

$$(q-t) \cdot q^{-1} = \frac{q-r}{q} \leq \Pr[f(a_1) \neq 0].$$

If $r = q-1$ we have $s = 1$ and $t = 0$, but then again we have that (9.4) equals

$$q \cdot q^{-2} = \frac{q-(q-1)}{q} \leq \Pr[f(a_1) \neq 0],$$

where the inequality follows from the degree mantra.

Now we turn to the inductive step. Assume the hypothesis is true for $(m-1)$-variate polynomials and let $f = \sum_{i=0}^{b} f_i X_m^i$ where $f_i \in \mathbb{F}_q[X_1, \ldots, X_{m-1}]$ with $f_b \neq 0$. Note $0 \leq b \leq q-1$ and $\deg(f_b) \leq r-b$. Let $\mathscr{E}$ be the event of interest to us, i.e.,

$$\mathscr{E} = \{(a_1, \ldots, a_m) | f(a_1, \ldots, a_m) \neq 0\}.$$

Let

$$\mathscr{E}_1 = \{(a_1, \ldots, a_{m-1}) | f_b(a_1, \ldots, a_{m-1}) \neq 0\}.$$

We first bound $\Pr[\mathscr{E}|\mathscr{E}_1]$. Fix $a_1, \ldots, a_{m-1}$ such that $f_b(a_1, \ldots, a_{m-1}) \neq 0$ and let

$$P(Z) = \sum_{i=0}^{b} f_i(a_1, \ldots, a_{m-1}) Z^i.$$

Note P is a non-zero polynomial of degree b and we have

$$\Pr[f(a_1, \ldots, a_m) = 0 | a_1, \ldots, a_{m-1}] = \Pr_{a_m}[P(a_m) \neq 0].$$

Since by the degree mantra, a univariate polynomial of degree b has at most b roots, we have

$$\Pr_{a_m}[P(a_m) \neq 0] \geq \frac{q-b}{q}.$$

We conclude

$$\Pr[\mathscr{E}|\mathscr{E}_1] \geq 1 - \frac{b}{q}.$$

Next we will bound $\Pr[\mathscr{E}_1]$. This will allow us to lower bound the probability of $\mathscr{E}$ since

$$\Pr[\mathscr{E}] \geq \Pr[\mathscr{E} \text{ and } \mathscr{E}_1] = \Pr[\mathscr{E}_1] \cdot \Pr[\mathscr{E}|\mathscr{E}_1].$$

Recall that $\deg(f_b) \leq r-b$. Write $r-b = s'(q-1)+t'$ where $s', t' \geq 0$ and $t' \leq q-2$. By induction we have

$$\Pr[\mathscr{E}_1] = \Pr[f_b(a_1, \ldots, a_{m-1}) \neq 0] \geq (q-t') \cdot q^{-(s'+1)}.$$

Putting the two bounds together, we get

$$\Pr[\mathscr{E}] \geq \Pr[\mathscr{E}|\mathscr{E}_1] \cdot \Pr[\mathscr{E}_1] \geq \frac{q-b}{q} \cdot (q-t') \cdot q^{-(s'+1)}.$$

We are now left with a calculation to verify that the bound above is indeed lower bounded by $(q-t) \cdot q^{-(s+1)}$ and we do so in Claim 9.4.2 using the facts that $t, t' \leq q-2$, $b \leq q-1$, $r = s(q-1)+t$, and $r-b = s'(q-1)+t'$. In the claim further below (Claim 9.4.3), we also prove $(q-t) \cdot q^{-(s+1)} \geq q^{-r/(q-1)}$ and this concludes the proof of the lemma. □

Claim 9.4.2. *If q, r, s, t, s', t', b are non-negative integers such that $r = s(q-1)+t$, $r-b = s'(q-1)+t'$, $t, t' \leq q-2$ and $b \leq q-1$ then we have*

$$\frac{q-b}{q} \cdot (q-t') \cdot q^{-(s'+1)} \geq (q-t) \cdot q^{-(s+1)}.$$

Proof. The proof breaks up in to two cases depending on $s-s'$. Note that an equivalent definition of s and s' are that these are the quotients when we divide r and $r-b$ respectively by $q-1$. Since $0 \leq b \leq q-1$, it follows that either $s' = s$ or $s' = s-1$. We consider the two cases separately.

If $s = s'$ we have $t = t' + b$ and then it suffices to show that

$$\frac{q-b}{q} \cdot (q-t') \geq q - (t'+b).$$

In turn this is equivalent to showing

$$(q-b)(q-t') \geq q(q-(t'+b)).$$

But this is immediate since the expression on the left is

$$(q-b)(q-t') = q^2 - (b+t')q + bt' = q(q-(b+t')) + bt' \geq q(q-(b+t')),$$

where the final inequality uses $bt' \geq 0$.

If $s = s'+1$ we have a bit more work. Here we have $t+q-1 = t'+b$ and it suffices to show that

$$\frac{q-b}{q} \cdot (q-t') \cdot q \geq (q-t) = (2q-(t'+b+1)).$$

Write $q-b = \alpha$ and $q-t' = \beta$. The expression on the left above simplifies to $\alpha\beta$ and on the right to $\alpha+\beta-1$. Since $b, t' \leq q-1$, we also have $\alpha, \beta \geq 1$. So it suffices to show that $\alpha\beta \geq \alpha+\beta-1$. This is true since $\alpha\beta = \alpha + \alpha(\beta-1)$ and we have $\alpha(\beta-1) \geq \beta-1$ since $\alpha \geq 1$ and $\beta - 1 \geq 0$.

We thus conclude that the inequality holds for both $s = s'$ and $s = s'+1$ and this yields the claim. □

Claim 9.4.3. *Let q, r, s, t be non-negative real numbers such that $q \geq 2$, $r = s(q-1)+t$ and $t \leq q-2$. Then*

$$(q-t) \cdot q^{-(s+1)} \geq q^{-r/(q-1)}.$$

We remark that while the inequality is quite useful, the proof below is not particularly insightful. We include it for completeness, but we recommend that the reader skip it unless necessary.

Proof of Claim 9.4.3. We have four parameters in the inequality above. We will simplify it in steps removing parameters one at a time. First we get rid of r by substituting $r = s(q-1)+t$. So it suffices to prove:

$$(q-t)\cdot q^{-(s+1)} \geq q^{-(s(q-1)+t)/(q-1)} = q^{-s}\cdot q^{-t/(q-1)}.$$

We can get rid of q^{-s} from both sides (since the remaining terms are non-negative) and so it suffices to prove:

$$\frac{q-t}{q} \geq q^{-t/(q-1)}.$$

Let $f_q(t) = \frac{t}{q} + q^{-t/(q-1)} - 1$. The inequality above is equivalent to proving $f_q(t) \leq 0$ for $0 \leq t \leq q-2$. We use some basic calculus to prove the above. Note that the first and second derivatives of f_q with respect to t are given by $f'_q(t) = \frac{1}{q} - \frac{\ln q}{q-1} q^{-t/(q-1)}$ and $f''_q(t) = \left(\ln(q)/(q-1)\right)^2 q^{-t/(q-1)}$. In particular the second derivative is always positive which means $f_q(t)$ is maximized at one of the two end points of the interval $t \in [0, q-2]$. We have $f_q(0) = 0 \leq 0$ as desired and so it suffices to prove that

$$f_q(q-2) = q^{-(q-2)/(q-1)} - \frac{2}{q} \leq 0.$$

Multiplying the expression above by q we have that it suffices to show $q^{1/(q-1)} \leq 2$ which in turn is equivalent to proving $q \leq 2^{q-1}$ for every $q \geq 2$. The final inequality follows easily from Bernoulli's inequality (Lemma B.1.4) $1 + kx \leq (1+x)^k$ which holds for every $x \geq -1$ and $k \geq 1$. In our case we substitute $x = 1$ and $k = q-1$ to conclude $q \leq 2^{q-1}$ as desired. □

9.4.2 The general case: Dimension

For integers q, m, r let

$$S_{q,m,r} = \left\{ \mathbf{d} = (d_1, \ldots, d_m) \in \mathbb{Z}^m | 0 \leq d_i \leq q-1 \text{ for all } i \in [m] \text{ and }, \sum_{i=1}^{m} d_i \leq r \right\} \tag{9.5}$$

and let

$$K_{q,m,r} = |S_{q,m,r}|.$$

We start with the following, almost tautological, proposition.

Proposition 9.4.4. *For every prime power q and integers $m \geq 1$ and $r \geq 0$, the dimension of the code* $\mathrm{RM}(q,m,r)$ *is* $K_{q,m,r}$.

Proof. Follows from the fact that for every $\mathbf{d} = (d_1, \ldots, d_m) \in S_{q,m,r}$ the associated monomial $\mathbf{X}^{\mathbf{d}} = X_1^{d_1} \cdots X_m^{d_m}$ is a monomial of degree at most r and individual degree at most $q-1$. Thus these monomials (i.e., their evaluations) form a basis for the Reed-Muller code $\mathrm{RM}(q,m,r)$. (See Exercise 9.10.) □

The definition of $K_{q,m,r}$ does not give a good hint about its growth so below we give a few bounds on $K_{q,m,r}$ that help estimate its growth. Specifically the proposition below gives a lower bound $K^-_{q,m,r}$ and an upper bound $K^+_{q,m,r}$ on $K_{q,m,r}$ that are (1) given by simple expressions and (2) within polynomial factors of each other for every setting of q, m, and r.

Proposition 9.4.5. *For integers $q \geq 2$, $m \geq 1$ and $r \geq 0$, let*

$$K^+_{q,m,r} \triangleq \min\left\{ q^m, \binom{m+r}{r} \right\}$$

and let

$$K^-_{q,m,r} \triangleq \begin{cases} \max\left\{ q^m/2, q^m - K^+_{q,m,(q-1)m-r} \right\} & \text{if } r \geq (q-1)m/2 \\ \max\left\{ \binom{m}{r}, \frac{1}{2}\left(\left\lfloor \frac{2r+m}{m} \right\rfloor \right)^m \right\} & \text{if } r < (q-1)m/2 \end{cases}$$

Then there are universal constants c_1, c_2 ($c_1 < 3.1$ and $c_2 < 8.2$ suffice) such that

$$K^-_{q,m,r} \leq K_{q,m,r} \leq K^+_{q,m,r} \leq c_1 \cdot (K^-_{q,m,r})^{c_2}$$

.

Proof. We tackle the inequalities in order of growing complexity of the proof. In our bounds we use the fact that $K_{q,m,r}$ is monotone non-decreasing in q as well as r (when other parameters are fixed)– see Exercise 9.11.

First we prove $K_{q,m,r} \leq K^+_{q,m,r}$. On the one hand we have

$$K_{q,m,r} \leq K_{q,m,(q-1)m} = q^m,$$

which follows by ignoring the total degree restriction and on the other hand we have

$$K_{q,m,r} \leq K_{r,m,r} = \binom{m+r}{r},$$

whereas here we ignored the individual degree restriction.

Next we show $K^-_{q,m,r} \leq K_{q,m,r}$. First we consider the case $r \geq (q-1)m/2$. Here we argue via symmetry. Consider a map that maps vectors $\mathbf{d} = (d_1, \ldots, d_m) \in \mathbb{Z}^m$ with $0 \leq d_i < q$ to $\overline{\mathbf{d}} = (q-1-d_1, \ldots, q-1-d_m)$. The map $\mathbf{d} \to \overline{\mathbf{d}}$ is a one-to-one map which maps vectors with $\sum_i d_i > r$ to vectors with $\sum_i \overline{d_i} < (q-1)m - r$. In other words either $d \in \{0, \ldots, q-1\}^m$ is in $S_{q,m,r}$ or $\overline{\mathbf{d}} \in S_{q,m,(q-1)m-r}$, thus establishing

$$K_{q,m,r} = q^m - K_{q,m,(q-1)m-r}.$$

Since $r \geq (q-1)m/2$ we have $(q-1)m - r \leq r$ and so

$$K_{q,m,r} \geq K_{q,m,(q-1)m-r},$$

which in turn implies

$$K_{q,m,r} \geq q^m/2.$$

This establishes $K_{q,m,r} \geq K^-_{q,m,r}$ when $r \geq (q-1)m/2$. Next, turning to the case $r < (q-1)m/2$, first let $q' = \left\lfloor \frac{2r+m}{m} \right\rfloor$. We have

$$K_{q,m,r} \geq K_{q',m,r} \geq (q')^m/2$$

since $r \geq (q'-1)m/2$, and this yields

$$K_{q,m,r} \geq (q')^m/2 = \frac{1}{2}\left(\left\lfloor \frac{2r+m}{m} \right\rfloor\right)^m.$$

Finally we also have

$$K_{q,m,r} \geq K_{2,m,r} = \sum_{i=0}^{r} \binom{m}{i} \geq \binom{m}{r},$$

thus establishing $K_{q,m,r} \geq K^-_{q,m,r}$ when $r < (q-1)m/2$.

Finally we turn to the inequalities showing $K^+_{q,m,r} \leq c_1 \cdot (K^-_{q,m,r})^{c_2}$. If $r \geq (q-1)m/2$ we have

$$\frac{q^m}{2} \leq K^-_{q,m,r} \leq K^+_{q,m,r} \leq q^m$$

establishing $K^+_{q,m,r} \leq 2K^-_{q,m,r}$. Next we consider the case $r < m/2$. In this case we have

$$K^-_{q,m,r} \geq \binom{m}{r} \geq (m/r)^r \geq 2^r.$$

On the other hand we also have

$$\binom{m+r}{r} \leq \left(\frac{e(m+r)}{r}\right)^r \leq \left(\frac{e\cdot(3/2)\cdot m}{r}\right)^r = \left(\frac{3e}{2}\right)^r \cdot \left(\frac{m}{r}\right)^r.$$

From $2^r \leq K^-_{q,m,r}$ we get $\left(\frac{3e}{2}\right)^r \leq (K^-_{q,m,r})^{\log_2(3e/2)}$. Combining with $\left(\frac{m}{r}\right)^r \leq K^-_{q,m,r}$ and $K^+_{q,m,r} \leq \binom{m+r}{r}$ we get

$$K^+_{q,m,r} \leq \left(\frac{3e}{2}\right)^r \cdot \left(\frac{m}{r}\right)^r \leq (K^-_{q,m,r})^{1+\log_2(3e/2)}$$

. Finally, we consider the case $m/2 \leq r < (q-1)m/2$. In this range we have

$$\left\lfloor \frac{2r+m}{m} \right\rfloor = 1 + \left\lfloor \frac{2r}{m} \right\rfloor \geq 1 + \frac{r}{m} = \frac{m+r}{m}.$$

Thus

$$K^-_{q,m,r} \geq \frac{1}{2}\left(\left\lfloor \frac{2r+m}{m} \right\rfloor\right)^m \geq \frac{1}{2}\left(\frac{m+r}{m}\right)^m \geq \frac{1}{2}\left(\frac{3}{2}\right)^m.$$

On the other hand we have

$$K^+_{q,m,r} \leq \binom{m+r}{m} \leq \left(\frac{e(m+r)}{m}\right)^m = e^m \cdot \left(\frac{m+r}{m}\right)^m.$$

Again we have $\left(\frac{m+r}{m}\right)^m \leq 2K^-_{q,m,r}$ and $e^m \leq (2K^-_{q,m,r})^{\log_2(3e/2)}$ and so $K^+_{q,m,r} \leq (2K^-_{q,m,r})^{1+\log_2(3e/2)}$. Thus in all cases we have $K^+_{q,m,r} \leq c_1 \cdot (K^-_{q,m,r})^{c_2}$ for $c_2 = 1+\log_2(3e/2) < 3.1$ and $c_1 = 2^{c_2} < 8.2$, as desired. □

We now give a few examples of codes that can be derived from the bounds above, to illustrate the variety offered by Reed-Muller codes. In each of the cases we set one or more of the parameters among alphabet size, rate, (relative) distance or absolute distance to a constant and explore the behavior in the other parameters. In all cases we use Lemma 9.4.1 to lower bound the distance and Proposition 9.4.5 to lower bound the dimension.

Example 9.4.6 (RM Codes of constant alphabet size and (relative) distance.)**.** *Fix q and $r < q-1$ and consider $m \to \infty$. Then the Reed-Muller codes* $\mathrm{RM}(q,m,r)$ *are $[N,K,D]_q$ codes with block length $N = q^m$, distance $D = \delta \cdot N$ for $\delta = 1 - r/q$, with dimension*

$$K \geq \binom{m}{r} \geq \left(\frac{\log_q N}{r}\right)^r.$$

In other words Reed-Muller codes yield codes of constant alphabet size and relative distance with dimension growing as an arbitrary polynomial in the logarithm of the block length.

Example 9.4.7 (Binary RM Codes of rate close to 1 with constant (absolute) distance.)**.** *Fix $q = 2$ and d and let $m \to \infty$. Then the Reed-Muller codes* $\mathrm{RM}(2,m,m-d)$ *are $[N,K,D]_2$ codes with $N = 2^m$, $D = 2^d$ and*

$$K \geq N - \binom{\log_2 N + d}{d} \geq N - (\log_2 N)^d.$$

(See Exercise 9.12 for bound on K.) Note that the rate $\to 1$ as $N \to \infty$.

Example 9.4.8 (RM codes of constant rate and relative distance over polynomially small alphabets.)**.** *Given any $\varepsilon > 0$ and let $m = \lceil \frac{1}{\varepsilon} \rceil$ and now consider $q \to \infty$ with $r = q/2$. Then the Reed-Muller codes* $\mathrm{RM}(q,m,r)$ *are $[N,K,D]_q$ codes with $N = q^m$, $D = \frac{N}{2}$ and*

$$K \geq \frac{1}{2}\left(\frac{q+m}{m}\right)^m \geq \frac{1}{2m^m} \cdot N.$$

Expressed in terms of N and ε, the codes have length N, dimension $\Omega\left(\varepsilon^{1/\varepsilon}\right) \cdot N$ and relative distance $1/2$ over an alphabet of size N^ε.

Another natural regime is to consider the case of constant rate $1/2$: see Exercise 9.13 for more.

Finally we mention a range of parameters that has been very useful in the theory of computer science. Here the alphabet size is growing with N, but very slowly. But the code has a fixed relative distance and dimension that is polynomially related to the block length.

Example 9.4.9 (RM Codes over polylogarithmic alphabets with polynomial dimension.)**.** *Given $0 < \varepsilon < 1$, let $q \to \infty$ and let $r = q/2$ and $m = q^\varepsilon$. Then the Reed-Muller codes* $\mathrm{RM}(q,m,r)$ *are $[N,K,D]_q$ codes with $N = q^m$, $D = \frac{N}{2}$ and*

$$K \geq \frac{1}{2}\left(\frac{q+m}{m}\right)^m \geq \frac{1}{2}\left(q^{1-\varepsilon}\right)^m = \frac{1}{2} \cdot N^{1-\varepsilon}.$$

Expressed in terms of N and ε, the codes have length N, dimension $\Omega(N^{1-\varepsilon})$ and relative distance $1/2$ over an alphabet of size $(\log N)^{1/\varepsilon}$. (See Exercise 9.14 for claim on the bound on q.)

9.5 Exercises

Exercise 9.1. Argue that any RM(q,m,r) is a linear code.

Exercise 9.2. Argue that for D as defined in (9.1), we have

$$|D| = \binom{m+r}{r}.$$

Exercise 9.3. Show that Lemma 9.2.2 is tight in the sense that for every prime power q and integers $m \geq 1$ and $1 \leq r \leq q-1$, there exists a polynomial with exactly $r \cdot q^{m-1}$ roots.

Exercise 9.4. Show that Lemma 9.2.2 is not tight for most polynomials. In particular show that for every prime power q and integers $m \geq 1$ and $1 \leq r \leq q-1$, a random polynomial in $\mathbb{F}_q[X_1,\ldots,X_m]$ of degree r has q^{m-1} expected number of roots.

Exercise 9.5. Show that the Reed-Muller codes of Section 9.2 give rise to codes of relative distance δ (for any $0 < \delta < 1$) and block length n such that they have alphabet size of $\sqrt[m]{n}$ and rate $\frac{(1-\delta)^m}{m!}$.

Exercise 9.6. Prove Lemma 9.3.1.

Exercise 9.7. Prove that the lower bound in Lemma 9.3.1 is tight.

Exercise 9.8. Show that there exists a binary RM code with block length n, rate $n^{H(\tau)-1}$ and relative distance $n^{-\tau}$ for any $0 < \tau < 1/2$.

Exercise 9.9. Prove that the (first) lower bound in Lemma 9.4.1 is tight for all settings of the parameters.

Exercise 9.10. Prove that the evaluations of $\mathbf{X}^{\mathbf{d}}$ for every $\mathbf{d} \in S_{q,m,r}$ (as in (9.5)) form a basis for RM(q,m,r).

Exercise 9.11. Argue that $K_{q,m,r}$ is monotone non-decreasing in q as well as r (when other parameters are fixed).

Exercise 9.12. Argue the claimed bound on K in Example 9.4.7.

Exercise 9.13. Figure out a RM code that has rate $\frac{1}{2}$ and has as large a distance as possible and as small an alphabet as possible.

Exercise 9.14. Prove the claimed bound on q in Example 9.4.9.

9.6 Bibliographic Notes

We point out that the original code considered by Reed and Muller is the one in Section 9.3.

Chapter 10

From Large to Small Alphabets: Code Concatenation

Recall Question 8.3.2 that we had asked before: Is there an explicit asymptotically good binary code (that is, rate $R > 0$ and relative distance $\delta > 0$)? In this chapter, we will consider this question when we think of explicit code in the sense of Definition 6.3.1 as well as the stronger notion of a strongly explicit code (Definition 6.3.2).

Let us recall all the (strongly) explicit codes that we have seen so far. (See Table 10.1 for an overview.)

Code	R	δ
Hamming	$1 - O\left(\frac{\log n}{n}\right)$	$O\left(\frac{1}{n}\right)$
Hadamard	$O\left(\frac{\log n}{n}\right)$	$\frac{1}{2}$
Reed-Solomon	$\frac{1}{2}$	$O\left(\frac{1}{\log n}\right)$

Table 10.1: Strongly explicit binary codes that we have seen so far.

Hamming code (Section 2.4), which has rate $R = 1 - O(\log n/n)$ and relative distance $\delta = O(1/n)$ and the Hadamard code (Section 2.7), which has rate $R = O(\log n/n)$ and relative distance $1/2$. Both of these codes have extremely good R or δ at the expense of the other parameter. Next, we consider the Reed-Solomon code (of say $R = 1/2$) as a binary code, which does much better– $\delta = \frac{1}{\log n}$, as we discuss next.

Consider the Reed-Solomon over $\mathbb{F}_{2^s}$ for some large enough s. It is possible to get an $\left[n, \frac{n}{2}, \frac{n}{2}+1\right]_{2^s}$ Reed-Solomon code (i.e. $R = 1/2$). We now consider a Reed-Solomon codeword, where every symbol in $\mathbb{F}_{2^s}$ is represented by an s-bit vector. Now, the "obvious" binary code created by viewing symbols from $\mathbb{F}_{2^s}$ as bit vectors as above is an $\left[ns, \frac{ns}{2}, \frac{n}{2}+1\right]_2$ code[1]. Note that the distance of this code is only $\Theta\left(\frac{N}{\log N}\right)$, where $N = ns$ is the block length of the final binary code. (Recall that $n = 2^s$ and so $N = n \log n$.)

[1] The proof is left as an exercise.

The reason for the (relatively) poor distance is that the bit vectors corresponding to two different symbols in $\mathbb{F}_{2^s}$ may only differ by one bit. Thus, d positions which have different $\mathbb{F}_{2^s}$ symbols might result in a distance of only d as bit vectors.

To fix this problem, we can consider applying a function to the bit-vectors to increase the distance between those bit-vectors that differ in smaller numbers of bits. Note that such a function is simply a code! We define this recursive construction more formally next. This recursive construction is called concatenated codes and will help us construct (strongly) explicit asymptotically good codes.

10.1 Code Concatenation

For $q \geq 2$, $k \geq 1$ and $Q = q^k$, consider two codes which we call *outer* code and *inner* code:

$$C_{\text{out}} : [Q]^K \to [Q]^N,$$

$$C_{\text{in}} : [q]^k \to [q]^n.$$

Note that the alphabet size of C_{out} exactly matches the number of messages for C_{in}. Then given $\mathbf{m} = (m_1, \ldots, m_K) \in [Q]^K$, we have the code $C_{\text{out}} \circ C_{\text{in}} : [q]^{kK} \to [q]^{nN}$ defined as

$$C_{\text{out}} \circ C_{\text{in}}(\mathbf{m}) = (C_{\text{in}}(C_{\text{out}}(\mathbf{m})_1), \ldots, C_{\text{in}}(C_{\text{out}}(\mathbf{m})_N)),$$

where

$$C_{\text{out}}(\mathbf{m}) = (C_{\text{out}}(\mathbf{m})_1, \ldots, C_{\text{out}}(\mathbf{m})_N).$$

This construction is also illustrated in Figure 10.1.

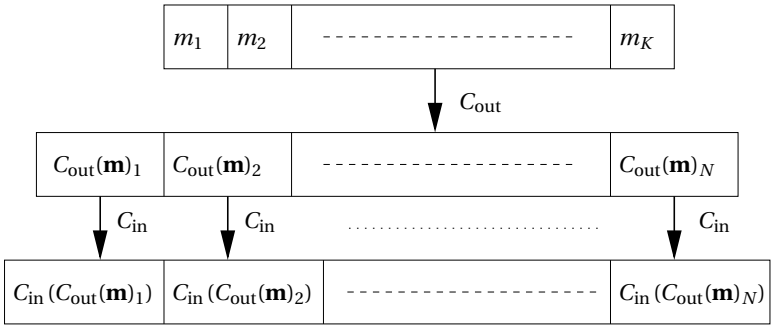


Figure 10.1: Concatenated code $C_{\text{out}} \circ C_{\text{in}}$.

We now look at some properties of a concatenated code.

Theorem 10.1.1. *If C_{out} is an $(N, K, D)_{q^k}$ code and C_{in} is an $(n, k, d)_q$ code, then $C_{\text{out}} \circ C_{\text{in}}$ is an $(nN, kK, dD)_q$ code. In particular, if C_{out} (C_{in} resp.) has rate R (r resp.) and relative distance δ_{out} (δ_{in} resp.) then $C_{\text{out}} \circ C_{\text{in}}$ has rate Rr and relative distance $\delta_{\text{out}} \cdot \delta_{\text{in}}$.*

Proof. The first claim immediately implies the second claim on the rate and relative distance of $C_{\text{out}} \circ C_{\text{in}}$. The claims on the block length, dimension and alphabet of $C_{\text{out}} \circ C_{\text{in}}$ follow from the definition.[2] Next we show that the distance is at least dD. Consider arbitrary $\mathbf{m}_1 \neq \mathbf{m}_2 \in [Q]^K$. Then by the fact that C_{out} has distance D, we have

$$\Delta\left(C_{\text{out}}\left(\mathbf{m}_1\right), C_{\text{out}}\left(\mathbf{m}_2\right)\right) \geq D. \tag{10.1}$$

Thus for each position $1 \leq i \leq N$ that contributes to the distance above, we have

$$\Delta\left(C_{\text{in}}\left(C_{\text{out}}\left(\mathbf{m}_1\right)_i\right), C_{\text{in}}\left(C_{\text{out}}\left(\mathbf{m}_2\right)_i\right)\right) \geq d, \tag{10.2}$$

as C_{in} has distance d. Since there are at least D such positions (from (10.1)), (10.2) implies

$$\Delta\left(C_{\text{out}} \circ C_{\text{in}}\left(\mathbf{m}_1\right), C_{\text{out}} \circ C_{\text{in}}\left(\mathbf{m}_2\right)\right) \geq dD.$$

The proof is complete as the choices of $\mathbf{m}_1$ and $\mathbf{m}_2$ were arbitrary. □

If C_{in} and C_{out} are linear codes, then so is $C_{\text{out}} \circ C_{\text{in}}$, which can be proved for example, by defining a generator matrix for $C_{\text{out}} \circ C_{\text{in}}$ in terms of the generator matrices of C_{in} and C_{out}. The proof is left as an exercise.

10.2 Zyablov Bound

We now instantiate an outer and inner codes in Theorem 10.1.1 to obtain a new lower bound on the rate given a relative distance. We'll initially just state the lower bound (which is called the Zyablov bound) and then we will consider the explicitness of such codes.

We begin with the instantiation of C_{out}. Note that this is a code over a large alphabet and we have seen an optimal code over large enough alphabet: Reed-Solomon codes (Chapter 5). Recall that the Reed-Solomon codes are optimal because they meet the Singleton bound 4.3.1. Hence, let us assume that C_{out} meets the Singleton bound with rate of R, i.e. C_{out} has relative distance $\delta_{\text{out}} > 1 - R$. Note that now we have a chicken and egg problem here. In order for $C_{\text{out}} \circ C_{\text{in}}$ to be an asymptotically good code, C_{in} needs to have rate $r > 0$ and relative distance $\delta_{\text{in}} > 0$ (i.e. C_{in} also needs to be an asymptotically good code). This is precisely the kind of code we are looking for to answer Question 8.3.2! However the saving grace will be that k can be much smaller than the block length of the concatenated code and hence, we can spend "more" time searching for such an inner code.

Suppose C_{in} meets the GV bound (Theorem 4.2.1) with rate of r and thus with relative distance $\delta_{\text{in}} \geq H_q^{-1}(1-r) - \varepsilon$, for some $\varepsilon > 0$. Then by Theorem 10.1.1, $C_{\text{out}} \circ C_{\text{in}}$ has rate of rR and $\delta = (1-R)(H_q^{-1}(1-r) - \varepsilon)$. Expressing R as a function of δ and r, we get the following:

$$R = 1 - \frac{\delta}{H_q^{-1}(1-r) - \varepsilon}.$$

[2] Technically, we need to argue that the q^{kK} messages map to distinct codewords to get the dimension of kK. However, this follows from the fact, which we will prove soon, that $C_{\text{out}} \circ C_{\text{in}}$ has distance $dD \geq 1$, where the inequality follows for $d, D \geq 1$.

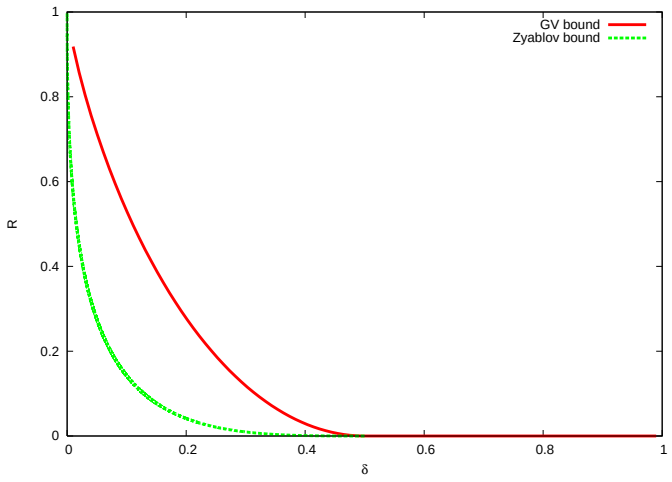


Figure 10.2: The Zyablov bound for binary codes. For comparison, the GV bound is also plotted.

Then optimizing over the choice of r, we get that the rate of the concatenated code satisfies

$$\mathscr{R} \geq \max_{0<r<1-H_q(\delta+\varepsilon)} r\left(1-\frac{\delta}{H_q^{-1}(1-r)-\varepsilon}\right),$$

where the bound of $r < 1-H_q(\delta+\varepsilon)$ is necessary to ensure that $\mathscr{R} > 0$. This lower bound on the rate is called the Zyablov bound. See Figure 10.2 for a plot of this bound for binary codes.

To get a feel for how the bound behaves, consider the case when $\delta = \frac{1}{2} - \varepsilon$. We claim that the Zybalov bound states that $\mathscr{R} \geq \Omega(\varepsilon^3)$. (Recall that the GV bound for the same δ has a rate of $\Omega(\varepsilon^2)$.) The proof of this claim is left as an exercise.

Note that the Zyablov bound implies that for every $\delta > 0$, there exists a (concatenated) code with rate $R > 0$. However, we already knew about the existence of an asymptotically good code by the GV bound (Theorem 4.2.1). Thus, a natural question to ask is the following:

Question 10.2.1. *Can we construct an explicit code on the Zyablov bound?*

We will focus on linear codes in seeking an answer to the question above because linear codes have polynomial size representation. Let C_{out} be an $[N,K]_Q$ Reed-Solomon code where $N = Q-1$ (evaluation points being $\mathbb{F}_Q^*$ with $Q = q^k$). This implies that $k = \Theta(\log N)$. However we still need an efficient construction of an inner code that lies on the GV bound. We do not expect to construct such a C_{in} in time poly(k) as that would answer Open Question 8.3.2! However, since $k = O(\log N)$, note that an exponential time in k algorithm is still a polynomial (in N) time algorithm.

There are two options for this exponential (in k) time construction algorithm for C_{in}:

- Perform an exhaustive search among all generator matrices for one satisfying the required property for C_{in}. One can do this because the Varshamov bound (Theorem 4.2.1) states that there exists a linear code which lies on the GV bound. This will take $q^{O(kn)}$ time. Using $k = rn$ (or $n = O(k)$), we get $q^{O(kn)} = q^{O(k^2)} = N^{O(\log N)}$, which is upper bounded by $(nN)^{O(\log(nN))}$, a quasi-polynomial time bound.

- The second option is to construct C_{in} in $q^{O(n)}$ time and thus use $(nN)^{O(1)}$ time overall. See Exercise 10.1 for one way to construct codes on the GV bound in time $q^{O(n)}$.

Thus,

Theorem 10.2.1. *We can construct a code that achieves the Zyablov bound in polynomial time.*

In particular, we can construct explicit asymptotically good code in polynomial time, which answers Question 10.2.1 in the affirmative.

A somewhat unsatisfactory aspect of this construction (in the proof of Theorem 10.2.1) is that one needs a brute force search for a suitable inner code (which led to the polynomial construction time). A natural followup question is

Question 10.2.2. *Does there exist a strongly explicit asymptotically good code?*

10.3 Strongly Explicit Construction

We will now consider what is known as the *Justesen code*. The main insight in these codes is that if we are only interested in asymptotically good codes, then the arguments in the previous section would go through even if (i) we pick different inner codes for each of the N outer codeword positions and (ii) most (but not necessarily all) inner code lie on the GV bound. It turns out that constructing an "ensemble" of codes such that most of the them lie on the GV bound is much easier than constructing a single code on the GV bound. For example, the ensemble of all linear codes have this property– this is exactly what Varshamov proved. However, it turns out that we need this ensemble of inner codes to be a smaller one than the set of all linear codes.

Justesen code is concatenated code with multiple, *different* linear inner codes. Specifically, it is composed of an $(N, K, D)_{q^k}$ outer code C_{out} and different inner codes $C_{\text{in}}^i : 1 \le i \le N$. Formally, the concatenation of these codes, denoted by $C_{\text{out}} \circ \left(C_{\text{in}}^1, \ldots, C_{\text{in}}^N\right)$, is defined as follows: given a message $\mathbf{m} \in \left[q^k\right]^K$, let the outer codeword be denoted by $(c_1, \ldots, c_N) \stackrel{\text{def}}{=} C_{\text{out}}(\mathbf{m})$. Then $C_{\text{out}} \circ \left(C_{\text{in}}^1, \ldots, C_{\text{in}}^N\right)(\mathbf{m}) = \left(C_{\text{in}}^1(c_1), C_{\text{in}}^2(c_2), \ldots, C_{\text{in}}^n(c_N)\right)$.

We will need the following result.

Theorem 10.3.1. *Let $\varepsilon > 0$. There exists an ensemble of inner codes $C_{\text{in}}^1, C_{\text{in}}^2, \ldots, C_{\text{in}}^N$ of rate $\frac{1}{2}$, where $N = q^k - 1$, such that for at least $(1-\varepsilon)N$ values of i, C_{in}^i has relative distance $\ge H_q^{-1}\left(\frac{1}{2} - \varepsilon\right)$.*

In fact, this ensemble is the following: for $\alpha \in \mathbb{F}_{q^k}^*$, the inner code $C_{\text{in}}^\alpha : \mathbb{F}_q^k \to \mathbb{F}_q^{2k}$ is defined as $C_{\text{in}}^\alpha(x) = (x, \alpha x)$. This ensemble is called the *Wozencraft ensemble*. We claim that C_{in}^α for every $\alpha \in \mathbb{F}_{q^k}^*$ is linear and is strongly explicit. (The proof if left as an exercise.)

10.3.1 Justesen code

For the Justesen code, the outer code C_{out} is a Reed-Solomon code evaluated over $\mathbb{F}_{q^k}^*$ of rate R, $0 < R < 1$. The outer code C_{out} has relative distance $\delta_{\text{out}} = 1 - R$ and block length of $N = q^k - 1$. The set of inner codes is the Wozencraft ensemble $\{C_{\text{in}}^\alpha\}_{\alpha \in \mathbb{F}_{q^k}^*}$ from Theorem 10.3.1. So the Justesen code is the concatenated code $C^* \stackrel{\text{def}}{=} C_{\text{out}} \circ (C_{\text{in}}^1, C_{\text{in}}^2, \ldots, C_{\text{in}}^N)$ with the rate $\frac{R}{2}$. The following proposition estimates the distance of C^*.

Proposition 10.3.2. *Let $\varepsilon > 0$. C^* has relative distance at least $(1 - R - \varepsilon) \cdot H_q^{-1}\left(\frac{1}{2} - \varepsilon\right)$*

Proof. Consider $\mathbf{m}^1 \neq \mathbf{m}^2 \in (\mathbb{F}_{q^k})^K$. By the distance of the outer code $|S| \geq (1 - R)N$, where

$$S = \left\{ i | C_{\text{out}}(\mathbf{m}^1)_i \neq C_{\text{out}}(\mathbf{m}^2)_i \right\}.$$

Call the ith inner code *good* if C_{in}^i has distance at least $d \stackrel{\text{def}}{=} H_q^{-1}\left(\frac{1}{2} - \varepsilon\right) \cdot 2k$. Otherwise, the inner code is considered bad. Note that by Theorem 10.3.1, there are at most εN bad inner codes. Let S_g be the set of all good inner codes in S, while S_b is the set of all bad inner codes in S. Since $S_b \leq \varepsilon N$,

$$|S_g| = |S| - |S_b| \geq (1 - R - \varepsilon)N. \tag{10.3}$$

For each good $i \in S$, by definition we have

$$\Delta\left(C_{\text{in}}^i\left(C_{\text{out}}\left(\mathbf{m}^1\right)_i\right), C_{\text{in}}^i\left(C_{\text{out}}\left(\mathbf{m}^2\right)_i\right)\right) \geq d. \tag{10.4}$$

Finally, from (10.3) and (10.4), we obtain that the distance of C^* is at least

$$(1 - R - \varepsilon) \cdot Nd = (1 - R - \varepsilon) H_q^{-1}\left(\frac{1}{2} - \varepsilon\right) N \cdot 2k,$$

as desired. □

Since the Reed-Solomon codes as well as the Wozencraft ensemble are strongly explicit, the above result implies the following:

Corollary 10.3.3. *The concatenated code C^* from Proposition 10.3.2 is an asymptotically good code and is strongly explicit.*

Thus, we have now satisfactorily answered Question 10.2.2 modulo Theorem 10.3.1, which we prove next.

Proof of Theorem 10.3.1. Fix $\mathbf{y} = (\mathbf{y}_1, \mathbf{y}_2) \in \mathbb{F}_q^{2k} \setminus \{\mathbf{0}\}$. Note that this implies that $\mathbf{y}_1 = \mathbf{0}$ and $\mathbf{y}_2 = \mathbf{0}$ are not possible. We claim that $\mathbf{y} \in C_{\text{in}}^{\alpha}$ for at most one $\alpha \in \mathbf{F}_{2^k}^*$. The proof is by a simple case analysis. First, note that if $\mathbf{y} \in C_{\text{in}}^{\alpha}$, then it has to be the case that $\mathbf{y}_2 = \alpha \cdot \mathbf{y}_1$.

- Case 1: $\mathbf{y}_1 \neq \mathbf{0}$ and $\mathbf{y}_2 \neq \mathbf{0}$, then $\mathbf{y} \in C_{\text{in}}^{\alpha}$, where $\alpha = \frac{\mathbf{y}_2}{\mathbf{y}_1}$.
- Case 2: $\mathbf{y}_1 \neq \mathbf{0}$ and $\mathbf{y}_2 = \mathbf{0}$, then $\mathbf{y} \notin C_{\text{in}}^{\alpha}$ for every $\alpha \in \mathbb{F}_{2^k}^*$ (as $\alpha \mathbf{y}_1 \neq \mathbf{0}$ since product of two elements in $\mathbb{F}_{2^k}^*$ also belongs to $\mathbb{F}_{2^k}^*$).
- Case 3: $\mathbf{y}_1 = \mathbf{0}$ and $\mathbf{y}_2 \neq \mathbf{0}$, then $\mathbf{y} \notin C_{\text{in}}^{\alpha}$ for every $\alpha \in \mathbb{F}_{2^k}^*$ (as $\alpha \mathbf{y}_1 = \mathbf{0}$).

Now assume that $wt(\mathbf{y}) < H_q^{-1}(1-\varepsilon)n$. Note that if $\mathbf{y} \in C_{\text{in}}^{\alpha}$, then C_{in}^{α} is "bad"(i.e. has relative distance $< H_q^{-1}\left(\frac{1}{2} - \varepsilon\right)$). Since $\mathbf{y} \in C_{\text{in}}^{\alpha}$ for at most one value of α, the total number of bad codes is at most

$$\begin{aligned}
\left|\left\{\mathbf{y} | wt(\mathbf{y}) < H_q^{-1}\left(\frac{1}{2} - \varepsilon\right) \cdot 2k\right\}\right| &\leq Vol_q\left(H_q^{-1}\left(\frac{1}{2} - \varepsilon\right) \cdot 2k, 2k\right) \\
&\leq q^{H_q(H_q^{-1}(\frac{1}{2}-\varepsilon)) \cdot 2k} \qquad (10.5) \\
&= q^{(\frac{1}{2}-\varepsilon) \cdot 2k} \\
&= \frac{q^k}{q^{2\varepsilon k}} \\
&< \varepsilon(q^k - 1) \qquad (10.6) \\
&= \varepsilon N. \qquad (10.7)
\end{aligned}$$

In the above, (10.5) follows from our good old upper bound on the volume of a Hamming ball (Proposition 3.3.1) while (10.6) is true for large enough k. Thus for at least $(1-\varepsilon)N$ values of α, C_{in}^{α} has relative distance at least $H_q^{-1}\left(\frac{1}{2} - \varepsilon\right)$, as desired. □

By concatenating an outer code of distance D and an inner code of distance d, we can obtain a code of distance at least $\geq Dd$ (Theorem 10.1.1). Dd is called the concatenated code's *design distance*. For asymptotically good codes, we have obtained polynomial time construction of such codes (Theorem 10.2.1, as well as strongly explicit construction of such codes (Corollary 10.3.3). Further, since these codes were linear, we also get polynomial time encoding. However, the following natural question about decoding still remains unanswered.

Question 10.3.1. *Can we decode concatenated codes up to half their design distance in polynomial time?*

10.4 Exercises

Exercise 10.1. In Section 4.2.1, we saw that the Gilbert construction can compute an $(n,k)_q$ code in time $q^{O(n)}$. Now the Varshamov construction (Section 4.2.2) is a randomized construction

and it is natural to ask how quickly can we compute an $[n,k]_q$ code that meets the GV bound. In this exercise, we will see that this can also be done in $q^{O(n)}$ deterministic time, though the deterministic algorithm is not that straight-forward anymore.

1. (*A warmup*) Argue that Varshamov's proof gives a $q^{O(kn)}$ time algorithm that constructs an $[n,k]_q$ code on the GV bound. (Thus, the goal of this exercise is to "shave" off a factor of k from the exponent.)

2. A $k \times n$ *Toeplitz Matrix* $A = \{A_{i,j}\}_{i=1,\,j=1}^{k,\,n}$ satisfies the property that $A_{i,j} = A_{i-1,j-1}$. In other words, any diagonal has the same value. For example, the following is a 4×6 Toeplitz matrix:

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 7 & 1 & 2 & 3 & 4 & 5 \\ 8 & 7 & 1 & 2 & 3 & 4 \\ 9 & 8 & 7 & 1 & 2 & 3 \end{pmatrix}$$

   A *random* $k \times n$ Toeplitz matrix $T \in \mathbb{F}_q^{k \times n}$ is chosen by picking the entries in the first row and column uniformly (and independently) at random.

   Prove the following claim: For any non-zero $\mathbf{m} \in \mathbb{F}_q^k$, the vector $\mathbf{m} \cdot T$ is uniformly distributed over $\mathbb{F}_q^n$, that is for every $\mathbf{y} \in \mathbb{F}_q^n$, $\Pr\left[\mathbf{m} \cdot T = \mathbf{y}\right] = q^{-n}$.

   (*Hint:* Write down the expression for the value at each of the n positions in the vector $\mathbf{m} \cdot T$ in terms of the values in the first row and column of T. Think of the values in the first row and column as variables. Then divide these variables into two sets (this "division" will depend on $\mathbf{m}$) say S and $\overline{S}$. Then argue the following: for every fixed $\mathbf{y} \in \mathbb{F}_q^n$ and for every fixed assignment to variables in S, there is a unique assignment to variables in $\overline{S}$ such that $\mathbf{m}T = \mathbf{y}$.)

3. Briefly argue why the claim in part (b) implies that a random code defined by picking its generator matrix as a random Toeplitz matrix with high probability lies on the GV bound.

4. Conclude that an $[n,k]_q$ code on the GV bound can be constructed in time $2^{O(k+n)}$.

10.5 Bibliographic Notes

Code concatenation was first proposed by Forney[21].

Justesen codes were constructed by Justesen [44]. In his paper, Justesen attributes the Wozencraft ensemble to Wozencraft.

Part IV

The Algorithms

Chapter 11

Decoding Concatenated Codes

In this chapter we study Question 10.3.1. Recall that the concatenated code $C_{\text{out}} \circ C_{\text{in}}$ consists of an outer $[N, K, D]_{Q=q^k}$ code C_{out} and an inner $[n, k, d]_q$ code C_{in}, where $Q = O(N)$. (Figure 11.1 illustrates the encoding function.) Then $C_{\text{out}} \circ C_{\text{in}}$ has design distance Dd and Question 10.3.1 asks if we can decode concatenated codes up to half the design distance (say for concatenated codes that we saw in Section 10.2 that lie on the Zyablov bound). In this chapter, we begin with a very natural unique decoding algorithm that can correct up to $Dd/4$ errors. Then we will consider a more sophisticated algorithm that will allow us to answer Question 10.3.1 in the affirmative.

11.1 A Natural Decoding Algorithm

We begin with a natural decoding algorithm for concatenated codes that “reverses" the encoding process (as illustrated in Figure 11.1). In particular, the algorithm first decodes the inner code and then decodes the outer code.

For the time being let us assume that we have a polynomial time unique decoding algorithm $D_{C_{\text{out}}} : \left[q^k\right]^N \to \left[q^k\right]^K$ for the outer code that can correct up to $D/2$ errors.

This leaves us with the task of coming up with a polynomial time decoding algorithm for the inner code. Our task of coming up with such a decoder is made easier by the fact that the running time needs to be polynomial in the *final* block length. This in turn implies that we would be fine if we pick a decoding algorithm that runs in singly exponential time in the inner block length as long as the inner block length is logarithmic in the outer code block length. (Recall that we put this fact to good use in Section 10.2 when we constructed explicit codes on the Zyablov bound.) Note that the latter is what we have assumed so far and thus, we can use the Maximum Likelihood Decoder (or MLD) (e.g. its implementation in Algorithm 1, which we will refer to as $D_{C_{\text{in}}}$). Algorithm 8 formalizes this algorithm.

It is easy to check that each step of Algorithm 8 can be implemented in polynomial time. In particular,

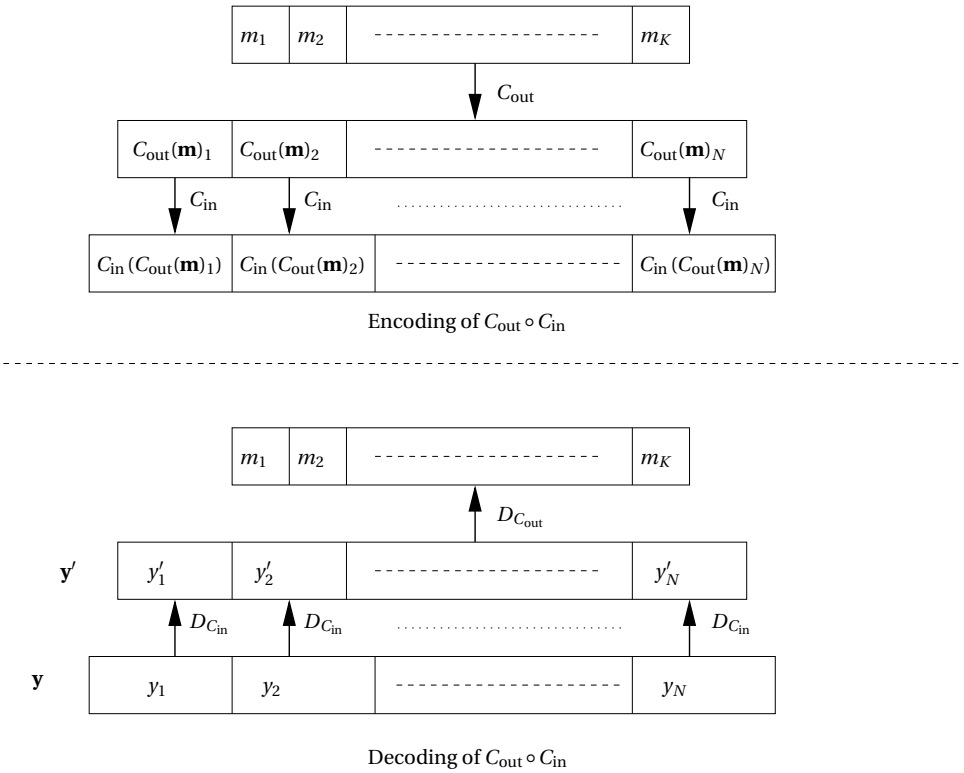


Figure 11.1: Encoding and Decoding of the concatenated code $C_{\text{out}} \circ C_{\text{in}}$. $D_{C_{\text{out}}}$ is a unique decoding algorithm for C_{out} and $D_{C_{\text{in}}}$ is a unique decoding algorithm for the inner code (e.g. MLD).

Algorithm 8 Natural Decoder for $C_{\text{out}} \circ C_{\text{in}}$

INPUT: Received word $\mathbf{y} = (y_1, \cdots, y_N) \in \left[q^n\right]^N$

OUTPUT: Message $\mathbf{m}' \in \left[q^k\right]^K$

1: $\mathbf{y}' \leftarrow (y'_1, \cdots, y'_N) \in \left[q^k\right]^N$ where

$$C_{\text{in}}(y'_i) = D_{C_{\text{in}}}(y_i) \quad 1 \le i \le N.$$

2: $\mathbf{m}' \leftarrow D_{C_{\text{out}}}(\mathbf{y}')$

3: RETURN $\mathbf{m}'$

1. The time complexity of Step 1 is $O(nq^k)$, which for our choice of $k = O(\log N)$ (and constant rate) for the inner code, is $(nN)^{O(1)}$ time.

2. Step 2 needs polynomial time by our assumption that the unique decoding algorithm $D_{C_{\text{out}}}$ takes $N^{O(1)}$ time.

Next, we analyze the error-correction capabilities of Algorithm 8:

Proposition 11.1.1. *Algorithm 8 can correct $< \frac{Dd}{4}$ many errors.*

Proof. Let $\mathbf{m}$ be the (unique) message such that $\Delta\left(C_{\text{out}} \circ C_{\text{in}}(\mathbf{m}), \mathbf{y}\right) < \frac{Dd}{4}$.

We begin the proof by defining a bad event as follows. We say a *bad event* has occurred (at position $1 \leq i \leq N$) if $y_i \neq C_{\text{in}}(C_{\text{out}}(\mathbf{m})_i)$. More precisely, define the set of all bad events to be

$$\mathscr{B} = \left\{i | y_i \neq C_{\text{in}}(C_{\text{out}}(\mathbf{m})_i)\right\}.$$

Note that if $|\mathscr{B}| < \frac{D}{2}$, then the decoder in Step 2 will output the message $\mathbf{m}$. Thus, to complete the proof, we only need to show that $|\mathscr{B}| < D/2$. To do this, we will define a superset $\mathscr{B}' \supseteq \mathscr{B}$ and then argue that $|\mathscr{B}'| < D/2$, which would complete the proof.

Note that if $\Delta\left(y_i, C_{\text{in}}(C_{\text{out}}(\mathbf{m})_i)\right) < \frac{d}{2}$ then $i \notin \mathscr{B}$ (by the proof of Proposition 1.4.1)– though the other direction does not hold. We define $\mathscr{B}'$ to be the set of indices where $i \in \mathscr{B}'$ if and only if

$$\Delta\left(y_i, C_{\text{in}}(C_{\text{out}}(\mathbf{m})_i)\right) \geq \frac{d}{2}.$$

Note that $\mathscr{B} \subseteq \mathscr{B}'$.

Now by definition, note that the total number of errors is at least $|\mathscr{B}'| \cdot \frac{d}{2}$. Thus, if $|\mathscr{B}'| \geq \frac{D}{2}$, then the total number of errors is at least $\frac{D}{2} \cdot \frac{d}{2} = \frac{Dd}{4}$, which is a contradiction. Thus, $|\mathscr{B}'| < \frac{D}{2}$, which completes the proof. □

Note that Algorithm 8 (as well the proof of Proposition 11.1.1) can be easily adapted to work for the case where the inner codes are different, e.g. Justesen codes (Section 10.3).

Thus, Proposition 11.1.1 and Theorem 11.3.3 imply that

Theorem 11.1.2. *There exist an explicit linear code on the Zyablov bound that can be decoded up to a fourth of the Zyablov bound in polynomial time.*

This of course is predicated on the fact that we need a polynomial time unique decoder for the outer code. Note that Theorem 11.1.2 implies the existence of an explicit asymptotically good code that can be decoded from a constant fraction of errors.

We now state two obvious open questions. The first is to get rid of the assumption on the existence of $D_{C_{\text{out}}}$:

Question 11.1.1. *Does there exist a polynomial time unique decoding algorithm for outer codes, e.g. for Reed-Solomon codes?*

Next, note that Proposition 11.1.1 does not quite answer Question 10.3.1. We move to answering this latter question next.

11.2 Decoding From Errors and Erasures

Now we digress a bit from answering Question 10.3.1 and talk about decoding Reed-Solomon codes. For the rest of the chapter, we will assume the following result.

Theorem 11.2.1. *An $[N,K]_q$ Reed-Solomon code can be corrected from e errors (or s erasures) as long as $e < \frac{N-K+1}{2}$ (or $s < N-K+1$) in $O(N^3)$ time.*

We defer the proof of the result on decoding from errors to Chapter 13 and leave the proof of the erasure decoder as an exercise. Next, we show that we can get the best of both worlds by correcting errors and erasures simultaneously:

Theorem 11.2.2. *An $[N,K]_q$ Reed-Solomon code can be corrected from e errors and s erasures in $O(N^3)$ time as long as*

$$2e + s < N - K + 1. \tag{11.1}$$

Proof. Given a received word $\mathbf{y} \in (\mathbb{F}_q \cup \{?\})^N$ with s erasures and e errors, let $\mathbf{y}'$ be the sub-vector with no erasures. This implies that $\mathbf{y}' \in \mathbb{F}_q^{N-s}$ is a valid received word for an $[N-s,K]_q$ Reed-Solomon code. (Note that this new Reed-Solomon code has evaluation points that corresponding to evaluation points of the original code, in the positions where an erasure did not occur.) Now run the error decoder algorithm from Theorem 11.2.1 on $\mathbf{y}'$. It can correct $\mathbf{y}'$ as long as

$$e < \frac{(N-s)-K+1}{2}.$$

This condition is implied by (11.1). Thus, we have proved one can correct e errors under (11.1). Now we have to prove that one can correct the s erasures under (11.1). Let $\mathbf{z}'$ be the output after correcting e errors. Now we extend $\mathbf{z}'$ to $\mathbf{z} \in (\mathbb{F}_q \cup \{?\})^N$ in the natural way. Finally, run the erasure decoding algorithm from Theorem 11.2.1 on $\mathbf{z}$. This works as long as $s < (N-K+1)$, which in turn is true by (11.1).

The time complexity of the above algorithm is $O(N^3)$ as both the algorithms from Theorem 11.2.1 can be implemented in cubic time. □

Next, we will use the above errors and erasure decoding algorithm to design decoding algorithms for certain concatenated codes that can be decoded up to half their design distance (i.e. up to $Dd/2$).

11.3 Generalized Minimum Distance Decoding

Recall the natural decoding algorithm for concatenated codes from Algorithm 8. In particular, we performed MLD on the inner code and then fed the resulting vector to a unique decoding algorithm for the outer code. A drawback of this algorithm is that it does not take into account the information that MLD provides. For example, it does not distinguish between the situations where a given inner code's received word has a Hamming distance of one vs where the received word has a Hamming distance of (almost) half the inner code distance from the closest codeword. It seems natural to make use of this information. Next, we study an algorithm called the Generalized Minimum Distance (or GMD) decoder, which precisely exploits this extra information.

In the rest of the section, we will assume C_{out} to be an $[N, K, D]_{q^k}$ code that can be decoded (by $D_{C_{\text{out}}}$) from e errors and s erasures in polynomial time as long as $2e + s < D$. Further, let C_{in} be an $[n, k, d]_q$ code with $k = O(\log N)$ which has a unique decoder $D_{C_{\text{in}}}$ (which we will assume is the MLD implementation from Algorithm 1).

We will in fact look at three versions of the GMD decoding algorithm. The first two will be randomized algorithms while the last will be a deterministic algorithm. We will begin with the first randomized version, which will present most of the ideas in the final algorithm.

11.3.1 GMD algorithm- I

Before we state the algorithm, let us look at two special cases of the problem to build some intuition.

Consider the received word $\mathbf{y} = (y_1, \ldots, y_N) \in [q^n]^N$ with the following special property: for every i such that $1 \leq i \leq N$, either $y_i = y_i'$ or $\Delta(y_i, y_i') \geq d/2$, where $y_i' = MLD_{C_{\text{in}}}(y_i)$. Now we claim that if $\Delta(\mathbf{y}, C_{\text{out}} \circ C_{\text{in}}) < dD/2$, then there are $< D$ positions in $\mathbf{y}$ such that $\Delta(y_i, C_{\text{in}}(y_i')) \geq d/2$ (we call such a position *bad*). This is because, for every bad position i, by the definition of y_i', $\Delta(y_i, C_{\text{in}}) \geq d/2$. Now if there are $\geq D$ bad positions, this implies that $\Delta(\mathbf{y}, C_{\text{out}} \circ C_{\text{in}}) \geq dD/2$, which is a contradiction. Now note that we can decode $\mathbf{y}$ by just declaring an erasure at every bad position and running the erasure decoding algorithm for C_{out} on the resulting vector.

Now consider the received word $\mathbf{y} = (y_1, \ldots, y_N)$ with the special property: for every i such that $i \in [N]$, $y_i \in C_{\text{in}}$. In other words, if there is an error at position $i \in [N]$, then a valid codeword in C_{in} gets mapped to another valid codeword $y_i \in C_{\text{in}}$. Note that this implies that a position with error has at least d errors. By a counting argument similar to the ones used in the previous paragraph, we have that there can be $< D/2$ such error positions. Note that we can now decode $\mathbf{y}$ by essentially running a unique decoder for C_{out} on $\mathbf{y}$ (or more precisely on $(x_1, \ldots, x_N)$, where $y_i = C_{\text{in}}(x_i)$).

Algorithm 9 generalizes these observations to decode arbitrary received words. In particular, it smoothly "interpolates" between the two extreme scenarios considered above.

Note that if $\mathbf{y}$ satisfies one of the two extreme scenarios considered earlier, then Algorithm 9 works exactly the same as discussed above.

By our choice of $D_{C_{\text{out}}}$ and $D_{C_{\text{in}}}$, it is easy to see that Algorithm 9 runs in polynomial time (in the final block length). More importantly, we will show that the final (deterministic) version of

Algorithm 9 Generalized Minimum Decoder (ver 1)

INPUT: Received word $\mathbf{y} = (y_1, \cdots, y_N) \in \left[q^n\right]^N$

OUTPUT: Message $\mathbf{m}' \in \left[q^k\right]^K$

1: FOR $1 \leq i \leq N$ DO
2: $y_i' \leftarrow D_{C_{\text{in}}}(y_i)$.
3: $w_i \leftarrow \min\left(\Delta(y_i', y_i), \frac{d}{2}\right)$.
4: With probability $\frac{2w_i}{d}$, set $y_i'' \leftarrow ?$, otherwise set $y_i'' \leftarrow x$, where $y_i' = C_{\text{in}}(x)$.
5: $\mathbf{m}' \leftarrow D_{C_{\text{out}}}(\mathbf{y}'')$, where $\mathbf{y}'' = (y_1'', \ldots, y_N'')$.
6: RETURN $\mathbf{m}'$

Algorithm 9 can do unique decoding of $C_{\text{out}} \circ C_{\text{in}}$ up to half of its design distance.

As a first step, we will show that in expectation, Algorithm 9 works.

Lemma 11.3.1. *Let* $\mathbf{y}$ *be a received word such that there exists a codeword* $C_{\text{out}} \circ C_{\text{in}}(\mathbf{m}) = (c_1, \ldots, c_N) \in [q^n]^N$ *such that* $\Delta(C_{\text{out}} \circ C_{\text{in}}(\mathbf{m}), \mathbf{y}) < \frac{Dd}{2}$. *Further, if* $\mathbf{y}''$ *has* e' *errors and* s' *erasures (when compared with* $C_{\text{out}} \circ C_{\text{in}}(\mathbf{m})$*), then*

$$\mathbb{E}\left[2e' + s'\right] < D.$$

Note that if $2e' + s' < D$, then by Theorem 11.2.2, Algorithm 9 will output $\mathbf{m}$. The lemma above says that in expectation, this is indeed the case.

Proof of Lemma 11.3.1. For every $1 \leq i \leq N$, define $e_i = \Delta(y_i, c_i)$. Note that this implies that

$$\sum_{i=1}^{N} e_i < \frac{Dd}{2}. \tag{11.2}$$

Next for every $1 \leq i \leq N$, we define two indicator variables:

$$X_i^? = \mathbb{1}_{y_i''=?},$$

and

$$X_i^e = \mathbb{1}_{C_{\text{in}}(y_i'') \neq c_i \text{ and } y_i'' \neq ?}.$$

We claim that we are done if we can show that for every $1 \leq i \leq N$:

$$\mathbb{E}\left[2X_i^e + X_i^?\right] \leq \frac{2e_i}{d}. \tag{11.3}$$

Indeed, by definition we have: $e' = \sum_i X_i^e$ and $s' = \sum_i X_i^?$. Further, by the linearity of expectation (Proposition 3.1.2), we get

$$\mathbb{E}\left[2e' + s'\right] \leq \frac{2}{d}\sum_i e_i < D,$$

where the inequality follows from (11.2).

To complete the proof, we will prove (11.3) by a case analysis. Towards this end, fix an arbitrary $1 \le i \le N$.

Case 1: ($c_i = y'_i$) First, we note that if $y''_i \neq ?$ then since $c_i = y'_i$, we have $X^e_i = 0$. This along with the fact that $\Pr[y''_i = ?] = \frac{2w_i}{d}$ implies

$$\mathbb{E}\left[X^?_i\right] = \Pr[X^?_i = 1] = \frac{2w_i}{d},$$

and

$$\mathbb{E}\left[X^e_i\right] = \Pr[X^e_i = 1] = 0.$$

Further, by definition we have

$$w_i = \min\left(\Delta(y'_i, y_i), \frac{d}{2}\right) \le \Delta(y'_i, y_i) = \Delta(c_i, y_i) = e_i.$$

The three relations above prove (11.3) for this case.

Case 2: ($c_i \neq y'_i$) As in the previous case, we still have

$$\mathbb{E}\left[X^?_i\right] = \frac{2w_i}{d}.$$

Now in this case, if an erasure is not declared at position i, then $X^e_i = 1$. Thus, we have

$$\mathbb{E}\left[X^e_i\right] = \Pr[X^e_i = 1] = 1 - \frac{2w_i}{d}.$$

Next, we claim that as $c_i \neq y'_i$,

$$e_i + w_i \ge d, \tag{11.4}$$

which implies

$$\mathbb{E}\left[2X^e_i + X^?_i\right] = 2 - \frac{2w_i}{d} \le \frac{2e_i}{d},$$

as desired.

To complete the proof, we show (11.4) via yet another case analysis.

Case 2.1: ($w_i = \Delta(y'_i, y_i) < d/2$) By definition of e_i, we have

$$e_i + w_i = \Delta(y_i, c_i) + \Delta(y'_i, y_i) \ge \Delta(c_i, y'_i) \ge d,$$

where the first inequality follows from the triangle inequality and the second inequality follows from the fact that C_{in} has distance d.

Case 2.2: ($w_i = \frac{d}{2} \le \Delta(y'_i, y_i)$) As y'_i is obtained from MLD, we have

$$\Delta(y'_i, y_i) \le \Delta(c_i, y_i).$$

This along with the assumption on $\Delta(y'_i, y_i)$, we get

$$e_i = \Delta(c_i, y_i) \ge \Delta(y'_i, y_i) \ge \frac{d}{2}.$$

This in turn implies that

$$e_i + w_i \ge d,$$

as desired. □

11.3.2 GMD Algorithm- II

Note that Step 4 in Algorithm 9 uses "fresh" randomness for each i. Next we look at another randomized version of the GMD algorithm that uses the *same* randomness for every i. In particular, consider Algorithm 10.

Algorithm 10 Generalized Minimum Decoder (ver 2)

INPUT: Received word $\mathbf{y} = (y_1, \cdots, y_N) \in \left[q^n\right]^N$

OUTPUT: Message $\mathbf{m}' \in \left[q^k\right]^K$

1: Pick $\theta \in [0,1]$ uniformly at random.
2: FOR $1 \leq i \leq N$ DO
3: $\quad y_i' \leftarrow D_{C_{\text{in}}}(y_i)$.
4: $\quad w_i \leftarrow \min\left(\Delta(y_i', y_i), \frac{d}{2}\right)$.
5: $\quad$ If $\theta < \frac{2w_i}{d}$, set $y_i'' \leftarrow ?$, otherwise set $y_i'' \leftarrow x$, where $y_i' = C_{\text{in}}(x)$.
6: $\mathbf{m}' \leftarrow D_{C_{\text{out}}}(\mathbf{y}'')$, where $\mathbf{y}'' = (y_1'', \ldots, y_N'')$.
7: RETURN $\mathbf{m}'$

We note that in the proof of Lemma 11.3.1, we only use the randomness to show that

$$\Pr\left[y_i'' = ?\right] = \frac{2w_i}{d}.$$

In Algorithm 10, we note that

$$\Pr\left[y_i'' = ?\right] = \Pr\left[\theta \in \left[0, \frac{2w_i}{d}\right)\right] = \frac{2w_i}{d},$$

as before (the last equality follows from our choice of θ). One can verify that the proof of Lemma 11.3.1 can be used to show the following lemma:

Lemma 11.3.2. *Let $\mathbf{y}$ be a received word such that there exists a codeword $C_{\text{out}} \circ C_{\text{in}}(\mathbf{m}) = (c_1, \ldots, c_N) \in [q^n]^N$ such that $\Delta(C_{\text{out}} \circ C_{\text{in}}(\mathbf{m}), \mathbf{y}) < \frac{Dd}{2}$. Further, if $\mathbf{y}''$ has e' errors and s' erasures (when compared with $C_{\text{out}} \circ C_{\text{in}}(\mathbf{m})$), then*

$$\mathbb{E}_\theta\left[2e' + s'\right] < D.$$

Next, we will see that Algorithm 10 can be easily "derandomized."

11.3.3 Derandomized GMD algorithm

Lemma 11.3.2 along with the probabilistic method shows that there exists a value $\theta^* \in [0,1]$ such that Algorithm 10 works correctly even if we fix θ to be θ^* in Step 1. Obviously we can obtain such a θ^* by doing an exhaustive search for θ. Unfortunately, there are uncountable choices of θ because $\theta \in [0,1]$. However, this problem can be taken care of by the following discretization trick.

Define $Q = \{0,1\} \cup \{\frac{2w_1}{d}, \cdots, \frac{2w_N}{d}\}$. Then because for each i, $w_i = \min(\Delta(y'_i, y_i), d/2)$, we have

$$Q = \{0,1\} \cup \{q_1, \cdots, q_m\}$$

where $q_1 < q_2 < \cdots < q_m$ for some $m \leq \left\lfloor \frac{d}{2} \right\rfloor$. Notice that for every $\theta \in [q_i, q_{i+1})$, just before Step 6, Algorithm 10 computes the same $\mathbf{y}''$. (See Figure 11.2 for an illustration as to why this is the case.)

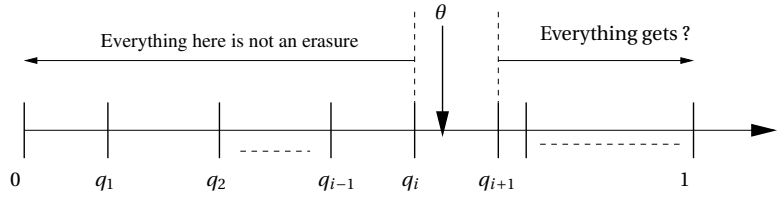


Figure 11.2: All values of $\theta \in [q_i, q_{i+1})$ lead to the same outcome

Thus, we need to cycle through all possible values of $\theta \in Q$, leading to Algorithm 11.

Algorithm 11 Deterministic Generalized Minimum Decoder'

INPUT: Received word $\mathbf{y} = (y_1, \cdots, y_N) \in \left[q^n\right]^N$
OUTPUT: Message $\mathbf{m}' \in \left[q^k\right]^K$

1: $Q \leftarrow \{\frac{2w_1}{d}, \cdots, \frac{2w_N}{d}\} \cup \{0,1\}$.
2: FOR $\theta \in Q$ DO
3: FOR $1 \leq i \leq N$ DO
4: $y'_i \leftarrow D_{C_{\text{in}}}(y_i)$.
5: $w_i \leftarrow \min\left(\Delta(y'_i, y_i), \frac{d}{2}\right)$.
6: If $\theta < \frac{2w_i}{d}$, set $y''_i \leftarrow ?$, otherwise set $y''_i \leftarrow x$, where $y'_i = C_{\text{in}}(x)$.
7: $\mathbf{m}'_\theta \leftarrow D_{C_{\text{out}}}(\mathbf{y}'')$, where $\mathbf{y}'' = (y''_1, \ldots, y''_N)$.
8: RETURN $\mathbf{m}'_{\theta^*}$ for $\theta^* = \arg\min_{\theta \in Q} \Delta\left(C_{\text{out}} \circ C_{\text{in}}\left(\mathbf{m}'_\theta\right), \mathbf{y}\right)$

Note that Algorithm 11 is Algorithm 10 repeated $|Q|$ times. Since $|Q|$ is $O(n)$, this implies that Algorithm 11 runs in polynomial time. This along with Theorem 10.2.1 implies that

Theorem 11.3.3. *For every constant rate, there exists an explicit linear binary code on the Zyablov bound. Further, the code can be decoded up to half of the Zyablov bound in polynomial time.*

Note that the above answers Question 10.3.1 in the affirmative.

11.4 Bibliographic Notes

Forney in 1966 designed the Generalized Minimum Distance (or GMD) decoding [22].

Chapter 12

Efficiently Achieving the Capacity of the BSC_p

Table 12.1 summarizes the main results we have seen so far for (binary codes).

	Shannon	Hamming	
		Unique Decoding	List Decoding
Capacity	$1-H(p)$ (Thm 6.3.1)	$\geq$ GV (Thm 4.2.1) $\leq$ MRRW (Sec 8.2)	$1-H(p)$ (Thm 7.4.1)
Explicit Codes	?	Zyablov bound (Thm 10.2.1)	?
Efficient Algorithms	?	$\frac{1}{2}\cdot$ Zyablov bound (Thm 11.3.3)	?

Table 12.1: An overview of the results seen so far

In this chapter, we will tackle the open questions in the first column of Table 12.1. Recall that there exist linear codes of rate $1-H(p)-\varepsilon$ such that decoding error probability is not more than $2^{-\delta n}$, $\delta=\Theta(\varepsilon^2)$ on the BSC_p (Theorem 6.3.1 and Exercise 6.3). This led to Question 6.3.1, which asks if we can achieve the BSC_p capacity with explicit codes and efficient decoding algorithms?

12.1 Achieving capacity of BSC_p

We will answer Question 6.3.1 in the affirmative by using concatenated codes. The main intuition in using concatenated codes is the following. As in the case of construction of codes on the Zyablov bound, we will pick the inner code to have the property that we are after: i.e. a code that achieves the BSC_p capacity. (We will again exploit the fact that since the block length of the inner code is small, we can construct such a code in a brute-force manner.) However, unlike the case of the Zyablov bound construction, we do not know of an explicit code that is optimal over say the $q\mathrm{SC}_p$ channel. The main observation here is that the fact that the BSC_p noise is memory-less can be exploited to pick the outer code that can correct from some small but constant fraction of *worst-case* errors.

Before delving into the details, we present the main ideas. We will use an outer code C_{out} that has rate close to 1 and can correct from some fixed constant (say γ) fraction of worst-case errors. We pick an inner code C_{in} that achieves the BSC_p capacity with parameters as guaranteed by Theorem 6.3.1. Since the outer code has rate almost 1, the concatenated code can be made to have the required rate (since the final rate is the product of the rates of C_{out} and C_{in}). For decoding, we use the natural decoding algorithm for concatenated codes from Algorithm 8. Assume that each of the inner decoders has a decoding error probability of (about) γ. Then the intermediate received word $\mathbf{y}'$ has an expected γ fraction of errors (with respect to the outer codeword of the transmitted message), though we might not have control over where the errors occur. However, we picked C_{out} so that it can correct up to γ fraction of worst-case errors. This shows that everything works in expectation. To make everything work with high probability (i.e. achieve exponentially small overall decoding error probability), we make use of the fact that since the noise in BSC_p is independent, the decoding error probabilities of each of the inner decodings is independent and thus, by the Chernoff bound (Theorem 3.1.6), with all but an exponentially small probability $\mathbf{y}'$ has $\Theta(\gamma)$ fraction of errors, which we correct with the worst-case error decoder for C_{out}. See Figure 12.1 for an illustration of the main ideas. Next, we present the details.

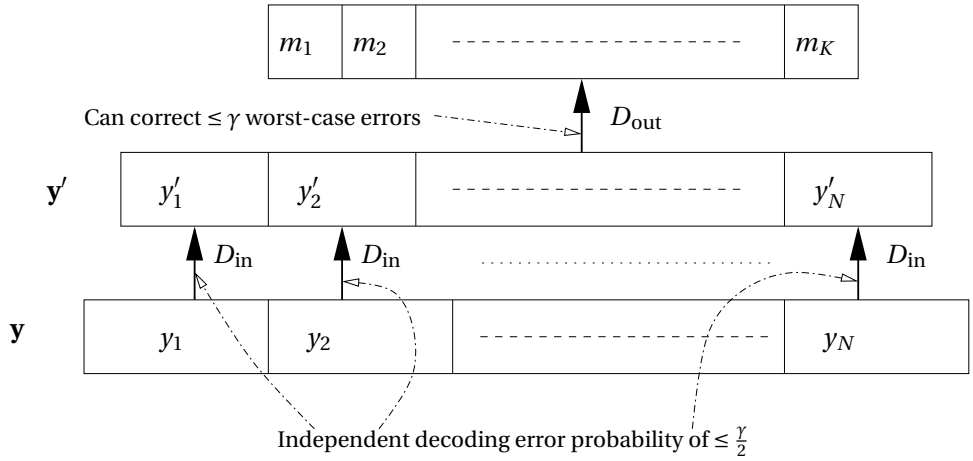


Figure 12.1: Efficiently achieving capacity of BSC_p.

We answer Question 6.3.1 in the affirmative by using a concatenated code $C_{\text{out}} \circ C_{\text{in}}$ with the following properties (where $\gamma > 0$ is a parameter that depends only on ε and will be fixed later on):

(i) C_{out}: The outer code is a linear $[N,K]_{2^k}$ code with rate $R \geq 1 - \frac{\varepsilon}{2}$, where $k = O(\log N)$. Further, the outer code has a unique decoding algorithm D_{out} that can correct at most γ fraction of worst-case errors in time $T_{\text{out}}(N)$.

(ii) C_{in}: The inner code is a linear binary $[n,k]_2$ code with a rate of $r \geq 1 - H(p) - \varepsilon/2$. Further, there is a decoding algorithm D_{in} (which returns the transmitted codeword) that runs in time $T_{\text{in}}(k)$ and has decoding error probability no more than $\frac{\gamma}{2}$ over BSC_p.

Table 12.2 summarizes the different parameters of C_{out} and C_{in}.

	Dimension	Block length	q	Rate	Decoder	Decoding time	Decoding guarantee
C_{out}	K	N	2^k	$1-\frac{\varepsilon}{2}$	D_{out}	$T_{\text{out}}(N)$	$\leq \gamma$ fraction of worst-case errors
C_{in}	$k \leq O(\log N)$	n	2	$1-H(p)-\frac{\varepsilon}{2}$	D_{in}	$T_{\text{in}}(k)$	$\leq \frac{\gamma}{2}$ decoding error probability over BSC_p

Table 12.2: Summary of properties of C_{out} and C_{in}

Suppose $C^* = C_{\text{out}} \circ C_{\text{in}}$. Then, it is easy to check that

$$R(C^*) = R \cdot r \geq \left(1-\frac{\varepsilon}{2}\right) \cdot \left(1-H(p)-\frac{\varepsilon}{2}\right) \geq 1-H(p)-\varepsilon,$$

as desired.

For the rest of the chapter, we will assume that p is an absolute constant. Note that this implies that $k = \Theta(n)$ and thus, we will use k and n interchangeably in our asymptotic bounds. Finally, we will use $\mathcal{N} = nN$ to denote the block length of C^*.

The decoding algorithm for C^* that we will use is Algorithm 8, which for concreteness we reproduce as Algorithm 12.

Algorithm 12 Decoder for efficiently achieving BSC_p capacity

INPUT: Received word $\mathbf{y} = (y_1, \cdots, y_N) \in \left[q^n\right]^N$
OUTPUT: Message $\mathbf{m}' \in \left[q^k\right]^K$

1: $\mathbf{y}' \leftarrow (y'_1, \cdots, y'_N) \in \left[q^k\right]^N$ where

$$C_{\text{in}}(y'_i) = D_{\text{in}}(y_i) \quad 1 \leq i \leq N.$$

2: $\mathbf{m}' \leftarrow D_{\text{out}}(\mathbf{y}')$
3: RETURN $\mathbf{m}'$

Note that encoding C^* takes time

$$O(N^2k^2) + O(Nkn) \leq O(N^2n^2) = O(\mathcal{N}^2),$$

as both the outer and inner codes are linear[1]. Further, the decoding by Algorithm 12 takes time

$$N \cdot T_{\text{in}}(k) + T_{\text{out}}(N) \leq \text{poly}(N),$$

[1]Note that encoding the outer code takes $O(N^2)$ operations over $\mathbb{F}_{q^k}$. The term $O(N^2k^2)$ then follows from the fact that each operation over $\mathbb{F}_{q^k}$ can be implemented with $O(k^2)$ operations over $\mathbb{F}_q$.

where the inequality is true as long as

$$T_{\text{out}}(N) = N^{O(1)} \text{ and } T_{\text{in}}(k) = 2^{O(k)}. \tag{12.1}$$

Next, we will show that decoding via Algorithm 12 leads to an exponentially small decoding error probability over BSC_p. Further, we will use constructions that we have already seen in this book to instantiate C_{out} and C_{in} with the required properties.

12.2 Decoding Error Probability

We begin by analyzing Algorithm 12.

By the properties of D_{in}, for any fixed i, there is an error at y'_i with probability $\leq \frac{\gamma}{2}$. Each such error is independent, since errors in BSC_p itself are independent by definition. Because of this, and by linearity of expectation, the expected number of errors in $\mathbf{y}'$ is $\leq \frac{\gamma N}{2}$.

Taken together, those two facts allow us to conclude that, by the (multiplicative) Chernoff bound (Theorem 3.1.6), the probability that the total number of errors will be more than γN is at most $e^{-\frac{\gamma N}{6}}$. Since the decoder D_{out} fails only when there are more than γN errors, this is also the final decoding error probability. Expressed in asymptotic terms, the error probability is $2^{-\Omega(\frac{\gamma \cdot N}{n})}$.

12.3 The Inner Code

We find C_{in} with the required properties by an exhaustive search among linear codes of dimension k with block length n that achieve the BSC_p capacity by Shannon's theorem (Theorem 6.3.1). Recall that for such codes with rate $1 - H(p) - \frac{\varepsilon}{2}$, the MLD has a decoding error probability of $2^{-\Theta(\varepsilon^2 n)}$ (Exercise 6.3). Thus, if k is at least $\Omega\left(\frac{\log(\frac{1}{\gamma})}{\varepsilon^2}\right)$, Exercise 6.3 implies the existence of a linear code with decoding error probability at most $\frac{\gamma}{2}$ (which is what we need). Thus, with the restriction on k from the outer code, we have the following restriction on k:

$$\Omega\left(\frac{\log(\frac{1}{\gamma})}{\varepsilon^2}\right) \leq k \leq O\big(\log N\big).$$

Note, however, that since the proof of Theorem 6.3.1 uses MLD on the inner code and Algorithm 1 is the only known implementation of MLD, we have $T_{\text{in}} = 2^{O(k)}$ (which is what we needed in (12.1). The construction time is even worse. There are $2^{O(kn)}$ generator matrices; for each of these, we must check the error rate for each of 2^k possible transmitted codewords, and for each codeword, computing the decoding error probability requires time $2^{O(n)}$.[2] Thus, the construction time for C_{in} is $2^{O(n^2)}$.

[2] To see why the latter claim is true, note that there are 2^n possible received words and given any one of these received words, one can determine (i) if the MLD produces a decoding error in time $2^{O(k)}$ and (ii) the probability that the received word can be realized, given the transmitted codeword in polynomial time.

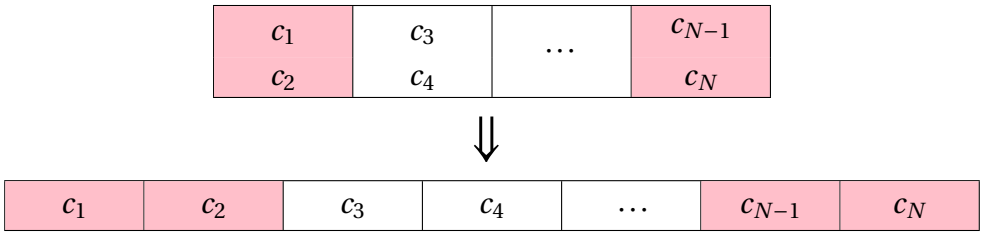


Figure 12.2: Error Correction cannot decrease during "folding." The example has $k = 2$ and a pink cell implies an error.

12.4 The Outer Code

We need an outer code with the required properties. There are several ways to do this.

One option is to set C_{out} to be a Reed-Solomon code over $\mathbb{F}_{2^k}$ with $k = \Theta(\log N)$ and rate $1-\frac{\varepsilon}{2}$. Then the decoding algorithm D_{out}, could be the error decoding algorithm from Theorem 11.2.2. Note that for this D_{out} we can set $\gamma = \frac{\varepsilon}{4}$ and the decoding time is $T_{\text{out}}(N) = O(N^3)$.

Till now everything looks on track. However, the problem is the construction time for C_{in}, which as we saw earlier is $2^{O(n^2)}$. Our choice of n implies that the construction time is $2^{O(\log^2 N)} \leq N^{O(\log N)}$, which of course is not polynomial time. Thus, the trick is to find a C_{out} defined over a smaller alphabet (certainly no larger than $2^{O(\sqrt{\log N})}$). This is what we do next.

12.4.1 Using a binary code as the outer code

The main observation is that we can also use an outer code which is some explicit binary linear code (call it C') that lies on the Zyablov bound and can be corrected from errors up to half its design distance. We have seen that such a code can be constructed in polynomial time (Theorem 11.3.3).

Note that even though C' is a binary code, we can think of C' as a code over $\mathbb{F}_{2^k}$ in the obvious way: every k consecutive bits are considered to be an element in $\mathbb{F}_{2^k}$ (say via a linear map). Note that the rate of the code does not change. Further, any decoder for C' that corrects bit errors can be used to correct errors over $\mathbb{F}_{2^k}$. In particular, if the algorithm can correct β fraction of bit errors, then it can correct that same fraction of errors over $\mathbb{F}_{2^k}$. To see this, think of the received word as $\mathbf{y} \in (\mathbb{F}_{2^k})^{N'/k}$, where N' is the block length of C' (as a binary code), which is at a fractional Hamming distance at most ρ away from $\mathbf{c} \in (\mathbb{F}_{2^k})^{N'/k}$. Here $\mathbf{c}$ is what once gets by "folding" consecutive k bits into one symbol in some codeword $\mathbf{c}' \in C'$. Now consider $\mathbf{y}' \in \mathbb{F}_2^{N'}$, which is just "unfolded" version of $\mathbf{y}$. Now note that each symbol in $\mathbf{y}$ that is in error (w.r.t. $\mathbf{c}$) leads to at most k bit errors in $\mathbf{y}'$ (w.r.t. $\mathbf{c}'$). Thus, in the unfolded version, the total number of errors is at most

$$k \cdot \rho \cdot \frac{N'}{k} = \rho \cdot N'.$$

(See Figure 12.2 for an illustration for $k = 2$.) Thus to decode $\mathbf{y}$, one can just "unfold" $\mathbf{y}$ to $\mathbf{y}'$ and use the decoding algorithm for C' (which can handle ρ fraction of errors) on $\mathbf{y}'$.

We will pick C_{out} to be C' when considered over $\mathbb{F}_{2^k}$, where we choose

$$k = \Theta\left(\frac{\log(\frac{1}{\gamma})}{\varepsilon^2}\right).$$

Further, D_{out} is the GMD decoding algorithm (Algorithm 11) for C'.

Now, to complete the specification of C^*, we relate γ to ε. The Zyablov bound gives $\delta_{\text{out}} = (1-R)H^{-1}(1-r)$, where R and r are the rates of the outer and inners codes for C'. Now we can set $1-R = 2\sqrt{\gamma}$ (which implies that $R = 1-2\sqrt{\gamma}$) and $H^{-1}(1-r) = \sqrt{\gamma}$, which implies that r is[3] $1-O\left(\sqrt{\gamma}\log\frac{1}{\gamma}\right)$. Since we picked D_{out} to be the GMD decoding algorithm, it can correct $\frac{\delta_{\text{out}}}{2} = \gamma$ fraction of errors in polynomial time, as desired.

The overall rate of C_{out} is simply $R \cdot r = \left(1-2\sqrt{\gamma}\right)\cdot\left(1-O\left(\sqrt{\gamma}\log\frac{1}{\gamma}\right)\right)$. This simplifies to $1-O\left(\sqrt{\gamma}\log\left(\frac{1}{\gamma}\right)\right)$. Recall that we need this to be at least $1-\frac{\varepsilon}{2}$. Thus, we would be done here if we could show that ε is $\Omega\left(\sqrt{\gamma}\log\frac{1}{\gamma}\right)$, which would follow by setting

$$\gamma = \varepsilon^3.$$

12.4.2 Wrapping Up

We now recall the construction, encoding and decoding time complexity for our construction of C^*. The construction time for C_{in} is $2^{O(n^2)}$, which substituting for n, is $2^{O\left(\frac{1}{\varepsilon^4}\log^2\left(\frac{1}{\varepsilon}\right)\right)}$. The construction time for C_{out}, meanwhile, is only $\text{poly}(N)$. Thus, our overall, construction time is $\text{poly}(\mathcal{N}) + 2^{O\left(\frac{1}{\varepsilon^4}\log^2\left(\frac{1}{\varepsilon}\right)\right)}$.

As we have seen in Section 12.1, the encoding time for this code is $O(\mathcal{N}^2)$, and the decoding time is $N^{O(1)} + N\cdot 2^{O(n)} = \text{poly}(\mathcal{N}) + \mathcal{N}\cdot 2^{O\left(\frac{1}{\varepsilon^2}\log\left(\frac{1}{\varepsilon}\right)\right)}$. We also have shown that the decoding error probability is exponentially small: $2^{-\Omega(\frac{\gamma\mathcal{N}}{n})} = 2^{-\Omega(\varepsilon^6\mathcal{N})}$. Thus, we have proved the following result:

Theorem 12.4.1. *For every constant p and $0 < \varepsilon < 1-H(p)$, there exists a linear code C^* of block length $\mathcal{N}$ and rate at least $1-H(p)-\varepsilon$, such that*

(a) C^* can be constructed in time $\text{poly}(\mathcal{N}) + 2^{O(\varepsilon^{-5})}$;

(b) C^* can be encoded in time $O(\mathcal{N}^2)$; and

(c) There exists a $\text{poly}(\mathcal{N}) + \mathcal{N}\cdot 2^{O(\varepsilon^{-5})}$ time decoding algorithm that has an error probability of at most $2^{-\Omega(\varepsilon^6\mathcal{N})}$ over the BSC_p.

[3]Note that $r = 1-H(\sqrt{\gamma}) = 1+\sqrt{\gamma}\log\sqrt{\gamma} + (1-\sqrt{\gamma})\log(1-\sqrt{\gamma})$. Noting that $\log(1-\sqrt{\gamma}) = -\sqrt{\gamma}-\Theta(\gamma)$, we can deduce that $r = 1-O(\sqrt{\gamma}\log(1/\gamma))$.

Thus, we have answered in the affirmative Question 6.3.1, which was the central open question from Shannon's work. However, there is a still somewhat unsatisfactory aspect of the result above. In particular, the exponential dependence on $1/\varepsilon$ in the decoding time complexity is not nice. This leads to the following question:

Question 12.4.1. *Can we bring the high dependence on ε down to $\mathrm{poly}\left(\frac{1}{\varepsilon}\right)$ in the decoding time complexity?*

12.5 Discussion and Bibliographic Notes

Forney answered Question 6.3.1 in the affirmative by using concatenated codes. (As was mentioned earlier, this was Forney's motivation for inventing code concatenation: the implication for the rate vs. distance question was studied by Zyablov later on.)

We now discuss Question 12.4.1. For the binary erasure channel, the decoding time complexity can be brought down to $\mathcal{N} \cdot \mathrm{poly}(\frac{1}{\varepsilon})$ using LDPC codes, specifically a class known as Tornado codes developed by Luby et al. [54]. The question for binary symmetric channels, however, is still open. Recently there have been some exciting progress on this front by the construction of the so-called Polar codes.

We conclude by noting an improvement to Theorem 12.4.1. We begin with a theorem due to Spielman:

Theorem 12.5.1 ([69])**.** *For every small enough $\beta > 0$, there exists an explicit C_{out} of rate $\frac{1}{1+\beta}$ and block length N, which can correct $\Omega\left(\frac{\beta^2}{(\log \frac{1}{\beta})^2}\right)$ errors, and has $O(N)$ encoding and decoding.*

Clearly, in terms of time complexity, this is superior to the previous option in Section 12.4.1. Such codes are called "Expander codes." One can essentially do the same calculations as in Section 12.4.1 with $\gamma = \Theta\left(\frac{\varepsilon^2}{\log^2(1/\varepsilon)}\right)$.[4] However, we obtain an encoding and decoding time of $\mathcal{N} \cdot 2^{\mathrm{poly}(\frac{1}{\varepsilon})}$. Thus, even though we obtain an improvement in the time complexities as compared to Theorem 12.4.1, this does not answer Question 12.4.1.

[4] This is because we need $1/(1+\beta) = 1 - \varepsilon/2$, which implies that $\beta = \Theta(\varepsilon)$.

Chapter 13

Efficient Decoding of Reed-Solomon Codes

So far in this book, we have shown how to efficiently decode explicit codes up to half of the Zyablov bound (Theorem 11.3.3) and how to efficiently achieve the capacity of the BSC_p (Theorem 12.4.1). The proofs of both of these results assumed that one can efficiently do unique decoding for Reed-Solomon codes up to half its distance (Theorem 11.2.1). In this chapter, we present such a unique decoding algorithm. Then we will generalize the algorithm to a list decoding algorithm that efficiently achieves the Johnson bound (Theorem 7.3.1).

13.1 Unique decoding of Reed-Solomon codes

Consider the $[n, k, d = n-k+1]_q$ Reed-Solomon code with evaluation points $(\alpha_1, \cdots, \alpha_n)$. (Recall Definition 5.2.1.) Our goal is to describe an algorithm that corrects up to $e < \frac{n-k+1}{2}$ errors in polynomial time. Let $\mathbf{y} = \left(y_1, \cdots, y_n\right) \in \mathbb{F}_q^n$ be the received word. We will now do a syntactic shift that will help us better visualize all the decoding algorithms in this chapter better. In particular, we will also think of $\mathbf{y}$ as the set of ordered pairs $\{(\alpha_1, y_1), (\alpha_2, y_2), \ldots, (\alpha_n, y_n)\}$, that is, we think of $\mathbf{y}$ as a collection of "points" in "2-D space." See Figure 13.1 for an illustration. From now on, we will switch back and forth between our usual vector interpretation of $\mathbf{y}$ and this new geometric notation.

Further, let us assume that there exists a polynomial $P(X)$ of degree at most $k-1$ such that $\Delta\left(\mathbf{y}, (P(\alpha_i))_{i=1}^n\right) \leq e$. (Note that if such a $P(X)$ exists then it is unique.) See Figure 13.2 for an illustration.

We will use reverse engineering to design a unique decoding algorithm for Reed-Solomon codes. We will assume that we somehow know $P(X)$ and then prove some identities involving (the coefficients of) $P(X)$. Then to design the algorithm we will just use the identities and try to solve for $P(X)$. Towards this end, let us assume that we also magically got our hands on a polynomial $E(X)$ such that

$$E(\alpha_i) = 0 \text{ if and only if } y_i \neq P(\alpha_i).$$

$E(X)$ is called an *error-locator polynomial*. We remark that there exists such a polynomial of

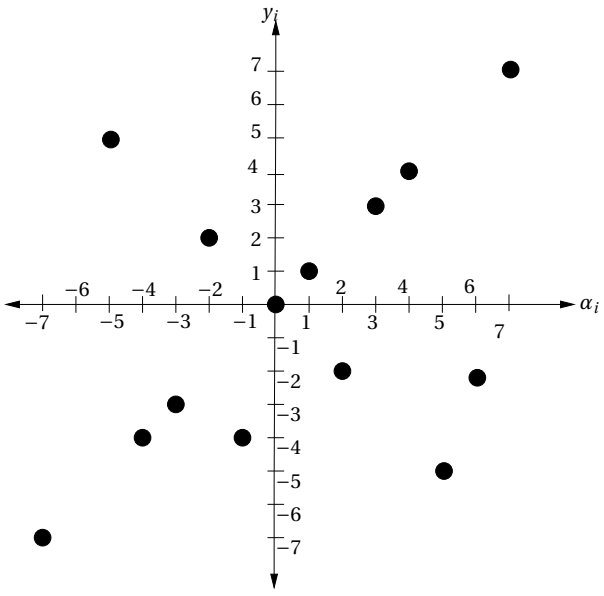


Figure 13.1: An illustration of a received word for a $[14,2]$ Reed-Solomon code (we have implicitly embedded the field $\mathbb{F}_q$ in the set $\{-7,\ldots,7\}$). The evaluations points are $(-7,-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7)$ and the received word is $(-7,5,-4,-3,2,-4,0,1,-2,3,4,-5,-2,7)$.

degree at most e. In particular, consider the polynomial:

$$E(X) = \prod_{i:y_i \neq P(\alpha_i)} (X - \alpha_i).$$

See Figure 13.3 for an illustration of the $E(X)$ corresponding to the received word in Figure 13.1.

Now we claim that for every $1 \leq i \leq n$,

$$y_i E(\alpha_i) = P(\alpha_i) E(\alpha_i). \tag{13.1}$$

To see why (13.1) is true, note that if $y_i \neq P(\alpha_i)$, then both sides of (13.1) are 0 (as $E(\alpha_i) = 0$). On the other hand, if $y_i = P(\alpha_i)$, then (13.1) is obviously true.

All the discussion above does not seem to have made any progress as both $E(X)$ and $P(X)$ are unknown. Indeed, the task of the decoding algorithm is to find $P(X)$! Further, if $E(X)$ is known then one can easily compute $P(X)$ from $\mathbf{y}$ (the proof is left as an exercise). However, note that we can now try and do reverse engineering. If we think of coefficients of $P(X)$ (of which there are k) and the coefficients of $E(X)$ (of which there are $e+1$) as variables, then we have n equations from (13.1) in $e+k+1$ variables. From our bound on e, this implies we have more equations than variables. Thus, if we could solve for these unknowns, we would be done.

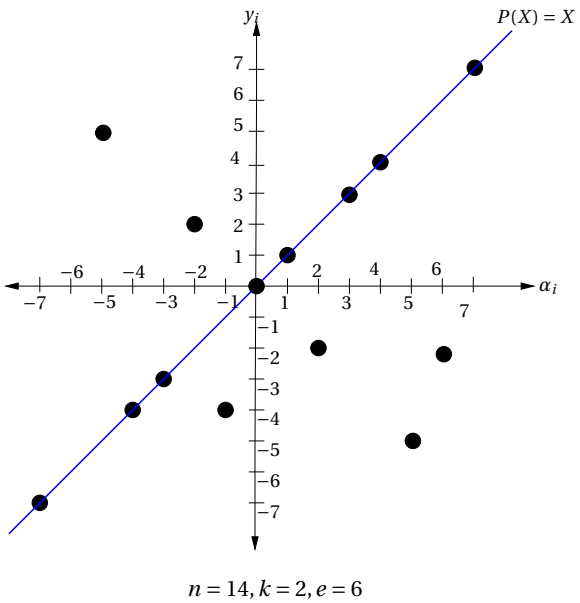


Figure 13.2: An illustration of the closest codeword $P(X) = X$ for the received word from Figure 13.1. Note that we are considering polynomials of degree 1, which are "lines."

However, there is a catch– these n equations are quadratic equations, which in general are NP-hard to solve. However, note that for our choice of e, we have $e + k - 1 \ll n$. Next, we will exploit this with a trick that is sometimes referred to as linearization. The idea is to introduce new variables so that we can convert the quadratic equations into linear equations. Care must be taken so that the number of variables after this linearization step is still smaller than the (now linear) n equations. Now we are in familiar territory as we know how to solve linear equations over a field (e.g. by Gaussian elimination). (See section 13.4 for some more discussion on the hardness of solving quadratic equations and the linearization technique.)

To perform linearization, define $N(X) \stackrel{\text{def}}{=} P(X) \cdot E(X)$. Note that $N(X)$ is a polynomial of degree less than or equal to $e + k - 1$. Further, if we can find $N(X)$ and $E(X)$, then we are done. This is because we can compute $P(X)$ as follows:

$$P(X) = \frac{N(X)}{E(X)}.$$

The main idea in the Welch-Berlekamp algorithm is to "forget" what $N(X)$ and $E(X)$ are meant to be (other than the fact that they are degree bounded polynomials).

13.1.1 Welch-Berlekamp Algorithm

Algorithm 13 formally states the Welch-Berlekamp algorithm.

As we have mentioned earlier, computing $E(X)$ is as hard as finding the solution polynomial $P(X)$. Also in some cases, finding the polynomial $N(X)$ is as hard as finding $E(X)$. E.g., given

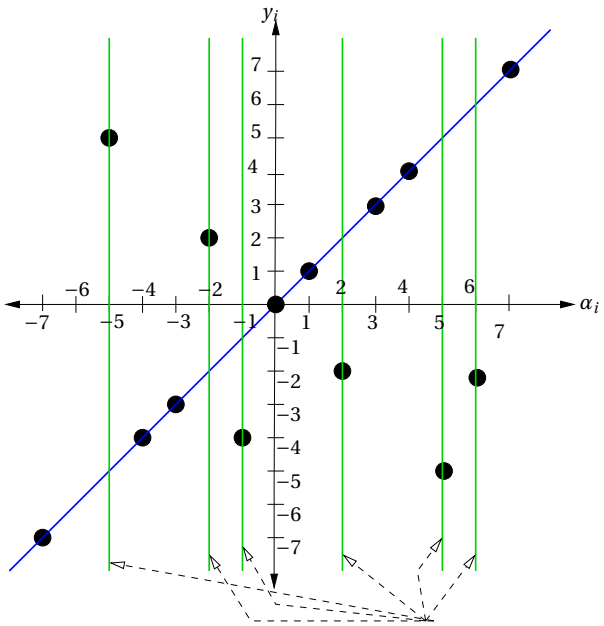


Figure 13.3: An illustration of the the error locator polynomial $E(X) = (X+5)(X+2)(X+1)(X-2)(X-5)(X-6)$ for the received word from Figure 13.1. Note that $E(X)$ is the product of the green lines.

$N(X)$ and $\mathbf{y}$ (such that $y_i \neq 0$ for $1 \leq i \leq n$) one can find the error locations by checking positions where $N(\alpha_i) = 0$. While each of the polynomials $E(X)$, $N(X)$ is hard to find individually, the main insight in the Welch-Berlekamp algorithm is that computing them together is easier.

Next we analyze the correctness and run time of Algorithm 13.

Correctness of Algorithm 13. Note that if Algorithm 13 does not output fail, then the algorithm produces a correct output. Thus, to prove the correctness of Algorithm 13, we just need the following result.

Theorem 13.1.1. *If $(P(\alpha_i))_{i=1}^n$ is transmitted (where $P(X)$ is a polynomial of degree at most $k-1$) and at most $e < \frac{n-k+1}{2}$ errors occur (i.e. $\Delta(\mathbf{y}, (P(\alpha_i))_{i=1}^n) \leq e$), then the Welch-Berlekamp algorithm outputs $P(X)$.*

The proof of the theorem above follows from the subsequent claims.

Claim 13.1.2. *There exist a pair of polynomials $E^*(X)$ and $N^*(X)$ that satisfy* Step 1 *such that $\frac{N^*(X)}{E^*(X)} = P(X)$.*

Note that now it suffices to argue that $\frac{N_1(X)}{E_1(X)} = \frac{N_2(X)}{E_2(X)}$ for any pair of solutions $((N_1(X), E_1(X))$ and $(N_2(X), E_2(X))$ that satisfy Step 1, since Claim 13.1.2 above can then be used to see that ratio must be $P(X)$. Indeed, we will show this to be the case:

Algorithm 13 Welch-Berlekamp Algorithm

INPUT: $n \geq k \geq 1$, $0 < e < \frac{n-k+1}{2}$ and n pairs $\{(\alpha_i, y_i)\}_{i=1}^{n}$ with α_i distinct
OUTPUT: Polynomial $P(X)$ of degree at most $k-1$ or fail.

1: Compute a non-zero polynomial $E(X)$ of degree exactly e, and a polynomial $N(X)$ of degree at most $e+k-1$ such that
$$y_i E(\alpha_i) = N(\alpha_i) \quad 1 \leq i \leq n. \tag{13.2}$$
2: IF $E(X)$ and $N(X)$ as above do not exist or $E(X)$ does not divide $N(X)$ THEN
3: RETURN fail
4: $P(X) \leftarrow \frac{N(X)}{E(X)}$.
5: IF $\Delta(\mathbf{y}, (P(\alpha_i))_{i=1}^{n}) > e$ THEN
6: RETURN fail
7: ELSE
8: RETURN $P(X)$

Claim 13.1.3. *If any two distinct solutions* $(E_1(X), N_1(X)) \neq (E_2(X), N_2(X))$ *satisfy* Step 1*, then they will satisfy*
$$\frac{N_1(X)}{E_1(X)} = \frac{N_2(X)}{E_2(X)}.$$

Proof of Claim 13.1.2. We just take $E^*(X)$ to be the error-locating polynomial for $P(X)$ and let $N^*(X) = P(X)E^*(X)$ where $\deg(N^*(X)) \leq \deg(P(X)) + \deg(E^*(X)) \leq e+k-1$. In particular, define $E^*(X)$ as the following polynomial of degree exactly e:
$$E^*(X) = X^{e-\Delta(\mathbf{y},(P(\alpha_i))_{i=1}^{n})} \prod_{1 \leq i \leq n | y_i \neq P(\alpha_i)} (X - \alpha_i). \tag{13.3}$$
By definition, $E^*(X)$ is a non-zero polynomial of degree e with the following property:
$$E^*(\alpha_i) = 0 \quad \text{iff} \quad y_i \neq P(\alpha_i).$$
We now argue that $E^*(X)$ and $N^*(X)$ satisfy (13.2). Note that if $E^*(\alpha_i) = 0$, then $N^*(\alpha_i) = P(\alpha_i)E^*(\alpha_i) = y_i E^*(\alpha_i) = 0$. When $E^*(\alpha_i) \neq 0$, we know $P(\alpha_i) = y_i$ and so we still have $P(\alpha_i)E^*(\alpha_i) = y_i E^*(\alpha_i)$, as desired. □

Proof of Claim 13.1.3. Note that the degrees of the polynomials $N_1(X)E_2(X)$ and $N_2(X)E_1(X)$ are at most $2e+k-1$. Let us define polynomial $R(X)$ with degree at most $2e+k-1$ as follows:
$$R(X) = N_1(X)E_2(X) - N_2(X)E_1(X). \tag{13.4}$$
Furthermore, from Step 1 we have, for every $i \in [n]$,
$$N_1(\alpha_i) = y_i E_1(\alpha_i) \quad \text{and} \quad N_2(\alpha_i) = y_i E_2(\alpha_i). \tag{13.5}$$

Substituting (13.5) into (13.4) we get for $1 \le i \le n$:

$$\begin{aligned} R(\alpha_i) &= (y_i E_1(\alpha_i))E_2(\alpha_i) - (y_i E_2(\alpha_i))E_1(\alpha_i) \\ &= 0. \end{aligned}$$

The polynomial $R(X)$ has n roots and

$$\begin{aligned} \deg(R(X)) &\le e + k - 1 + e \\ &= 2e + k - 1 \\ &< n, \end{aligned}$$

Where the last inequality follows from the upper bound on e. Since $\deg(R(X)) < n$, by the degree mantra (Proposition 5.2.3) we have $R(X) \equiv 0$. This implies that $N_1(X)E_2(X) \equiv N_2(X)E_1(X)$. Note that as $E_1(X) \neq 0$ and $E_2(X) \neq 0$, this implies that $\frac{N_1(X)}{E_1(X)} = \frac{N_2(X)}{E_2(X)}$, as desired. □

Implementation of Algorithm 13. In Step 1, $N(X)$ has $e+k$ unknowns and $E(X)$ has $e+1$ unknowns. For each $1 \le i \le n$, the constraint in (13.2) is a linear equation in these unknowns. Thus, we have a system of n linear equations in $2e+k+1 < n+2$ unknowns. By claim 13.1.2, this system of equations have a solution. The only extra requirement is that the degree of the polynomial $E(X)$ should be exactly e. We have already shown $E(X)$ in equation (13.3) to satisfy this requirement. So we add a constraint that the coefficient of X^e in $E(X)$ is 1. Therefore, we have $n+1$ linear equation in at most $n+1$ variables, which we can solve in time $O(n^3)$, e.g. by Gaussian elimination.

Finally, note that Step 4 can be implemented in time $O(n^3)$ by "long division." Thus, we have proved

Theorem 13.1.4. *For any $[n,k]_q$ Reed-Solomon code, unique decoding can be done in $O(n^3)$ time up to $\frac{d-1}{2} = \frac{n-k}{2}$ number of errors.*

Recall that the above is a restatement of the error decoding part of Theorem 11.2.1. Thus, this fills in the final missing piece from the proofs of Theorem 11.3.3 (decoding certain concatenated codes up to half of their design distance) and Theorem 12.4.1 (efficiently achieving the BSC_p capacity).

13.2 List Decoding Reed-Solomon Codes

Recall Question 7.4.3, which asks if there is an efficient list decoding algorithm for a code of rate $R > 0$ that can correct $1 - \sqrt{R}$ fraction of errors. Note that in the question above, explicitness is not an issue as e.g., a Reed-Solomon code of rate R by the Johnson bound is $(1 - \sqrt{R}, O(n^2))$-list decodable (Theorem 7.3.1).

We will study an efficient list decoding algorithm for Reed-Solomon codes that can correct up to $1 - \sqrt{R}$ fraction of errors. To this end, we will present a sequence of algorithms for (list) decoding Reed-Solomon codes that ultimately will answer Question 7.4.3.

Before we talk about the algorithms, we restate the (list) decoding problem for Reed-Solomon codes. Consider any $[n,k]_q$ Reed-Solomon code that has the evaluation set $\{\alpha_1,\ldots,\alpha_m\}$. Below is a formal definition of the decoding problem for Reed-Solomon codes:

- **Input**: Received word $\mathbf{y} = (y_1,\ldots,y_n)$, $y_i \in \mathbb{F}_q$ and error parameter $e = n - t$.
- **Output**: All polynomials $P(X) \in \mathbb{F}_q[X]$ of degree at most $k-1$ such that $P(\alpha_i) = y_i$ for at least t values of i.

Our main goal of course is to make t as small as possible.

We begin with the unique decoding regime, where $t > \frac{n+k}{2}$. We looked at the Welch-Berlekamp algorithm in Algorithm 13, which we restate below in a slightly different form (that will be useful in developing the subsequent list decoding algorithms).

- **Step 1:** Find polynomials $N(X)$ of degree $k+e-1$, and $E(X)$ of degree e such that

$$N(\alpha_i) = y_i E(\alpha_i), \text{ for every } 1 \leq i \leq n$$

- **Step 2:** If $Y - P(X)$ divides $Q(X,Y) = YE(X) - N(X)$, then output $P(X)$ (assuming $\Delta(\mathbf{y}, (P(\alpha_i))_{i=1}^n) \leq e$).

Note that $Y - P(X)$ divides $Q(X,Y)$ in **Step 2** above if and only if $P(X) = \frac{N(X)}{E(X)}$, which is exactly what Step 4 does in Algorithm 13.

13.2.1 Structure of list decoding algorithms for Reed-Solomon

Note that the Welch-Berlekamp Algorithm has the following general structure:

- **Step 1:** (Interpolation Step) Find non-zero $Q(X,Y)$ such that $Q(\alpha_i, y_i) = 0, 1 \leq i \leq n$.
- **Step 2:** (Root Finding Step) If $Y - P(X)$ is a factor of $Q(X,Y)$, then output $P(X)$ (assuming it is close enough to the received word).

In particular, in the Welch-Berlekamp algorithm we have $Q(X,Y) = YE(X) - N(X)$ and hence, **Step 2** is easy to implement.

All the list decoding algorithms that we will consider in this chapter will have the above two-step structure. The algorithms will differ in how exactly **Step 1** is implemented. Before we move on to the details, we make one observation that will effectively "take care of" **Step 2** for us. Note that **Step 2** can be implemented if one can factorize the bivariate polynomial $Q(X,Y)$ (and then only retain the linear factors of the form $Y - P(X)$). Fortunately, it is known that factoring bivariate polynomials can be done in polynomial time (see e.g. [45]). We will not prove this result in the book but will use this fact as a given.

Finally, to ensure the correctness of the two-step algorithm above for Reed-Solomon codes, we will need to ensure the following:

- **Step 1** requires solving for the co-efficients of $Q(X,Y)$. This can be done as long as the number of coefficients is greater than the the number of constraints. (The proof of this fact is left as an exercise.) Also note that this argument is a departure from the corresponding argument for the Welch-Berlekamp algorithm (where the number of coefficients is upper bounded by the number of constraints).

- In **Step 2**, to ensure that for every polynomial $P(X)$ that needs to be output $Y - P(X)$ divides $Q(X,Y)$, we will add restrictions on $Q(X,Y)$. For example, for the Welch-Berlekamp algorithm, the constraint is that $Q(X,Y)$ has to be of the form $YE(X) - N(X)$, where $E(X)$ and $N(X)$ are non-zero polynomials of degree e and at most $e+k-1$ respectively.

Next, we present the first instantiation of the algorithm structure above, which leads us to our first list decoding algorithm for Reed-Solomon codes.

13.2.2 Algorithm 1

The main insight in the list decoding algorithm that we will see is that if we carefully control the degree of the polynomial $Q(X,Y)$, then we can satisfy the required conditions that will allow us to make sure **Step 1** succeeds. Then we will see that the degree restrictions, along with the degree mantra (Proposition 5.2.3) will allow us to show **Step 2** succeeds too. The catch is in defining the correct notion of degree of a polynomial. We do that next.

First, we recall the definition of maximum degree of a variable.

Definition 13.2.1. $\deg_X(Q)$ is the maximum degree of X in $Q(X,Y)$. Similarly, $\deg_Y(Q)$ is the maximum degree of Y in $Q(X,Y)$

For example, for $Q(X,Y) = X^2Y^3 + X^4Y^2$ $\deg_X(Q) = 4$ and $\deg_Y(Q) = 3$. Given $\deg_X(Q) = a$ and $\deg_Y(Q) = b$, we can write

$$Q(X,Y) = \sum_{\substack{0 \le i \le a, \\ 0 \le j \le b}} c_{ij} X^i Y^j,$$

where the coefficients $c_{ij} \in \mathbb{F}_q$. Note that the number of coefficients is equal to $(a+1)(b+1)$.

The main idea in the first list decoding algorithm for Reed-Solomon code is to place bounds on $\deg_X(Q)$ and $\deg_Y(Q)$ for **Step 1**. The bounds are chosen so that there are enough variables to guarantee the existence of a $Q(X,Y)$ with the required properties. We will then use these bound along with the degree mantra (Proposition 5.2.3) to argue that **Step 2** works. Algorithm 14 presents the details. Note that the algorithm generalizes the Welch-Berlekamp algorithm (and follows the two step skeleton outlined above).

Correctness of Algorithm 14. To ensure the correctness of Step 1, we will need to ensure that the number of coefficients for $Q(X,Y)$ (which is $(\ell+1)(n/\ell+1)$) is larger than the number of constraints in (13.6 (which is n). Indeed,

$$(\ell+1)\cdot\left(\frac{n}{\ell}+1\right) > \ell\cdot\frac{n}{\ell} = n.$$

Algorithm 14 The First List Decoding Algorithm for Reed-Solomon Codes

INPUT: $n \geq k \geq 1$, $\ell \geq 1$, $e = n - t$ and n pairs $\{(\alpha_i, y_i)\}_{i=1}^{n}$
OUTPUT: (Possibly empty) list of polynomials $P(X)$ of degree at most $k-1$

1: Find a non-zero $Q(X,Y)$ with $\deg_X(Q) \leq \ell, \deg_Y(Q) \leq \frac{n}{\ell}$ such that

$$Q(\alpha_i, y_i) = 0, 1 \leq i \leq n. \tag{13.6}$$

2: Ł $\leftarrow \emptyset$
3: FOR every factor $Y - P(X)$ of $Q(X,Y)$ DO
4: IF $\Delta(\mathbf{y}, (P(\alpha_i))_{i=1}^{n}) \leq e$ and $\deg(P) \leq k-1$ THEN
5: Add $P(X)$ to Ł.
6: RETURN Ł

To argue that the final Ł in Step 6 contains all the polynomials $P(X)$ that need to be output. In other words, we need to show that if $P(X)$ of degree $\leq k-1$ agrees with Y in at least t positions, then $Y - P(X)$ divides $Q(X,Y)$. Towards this end, we define

$$R(X) \stackrel{\text{def}}{=} Q(X, P(X)).$$

Note that $Y - P(X)$ divides $Q(X,Y)$ if and only if $R(X) \equiv 0$. Thus, we need to show $R(X) \equiv 0$. For the sake of contradiction, assume that $R(X) \not\equiv 0$. Note that

$$\begin{aligned} \deg(R) &\leq \deg_X(Q) + \deg(P) \cdot \deg_Y(Q) && (13.7) \\ &\leq \ell + \frac{n(k-1)}{\ell}. && (13.8) \end{aligned}$$

On the other hand, if $P(\alpha_i) = y_i$ then (13.6) implies that

$$Q(\alpha_i, y_i) = Q(\alpha_i, P(\alpha_i)) = 0.$$

Thus, α_i is a root of $R(X)$. In other words R has at least t roots. Note that the degree mantra (Proposition 5.2.3) this will lead to a contradiction if $t > \deg(R)$, which will be true if

$$t > \ell + \frac{n(k-1)}{\ell}.$$

If we pick $\ell = \sqrt{n(k-1)}$, we will have $t > 2\sqrt{n(k-1)}$. Thus, we have shown

Theorem 13.2.1. *Algorithm 14 can list decode Reed-Solomon codes of rate R from $1 - 2\sqrt{R}$ fraction of errors. Further, the algorithm can be implemented in polynomial time.*

The claim on the efficient run time follows as Step 1 can be implemented by Gaussian elimination and for Step 3, all the factors of $Q(X,Y)$ (and in particular all linear factors of the form $Y - P(X)$) can be computed using e.g. the algorithm from [45].

The bound $1 - 2\sqrt{R}$ is better than the unique decoding bound of $\frac{1-R}{2}$ for $R < 0.07$. This is still far from the $1 - \sqrt{R}$ fraction of errors guaranteed by the Johnson bound. See Figure 13.2.2 for an illustration.

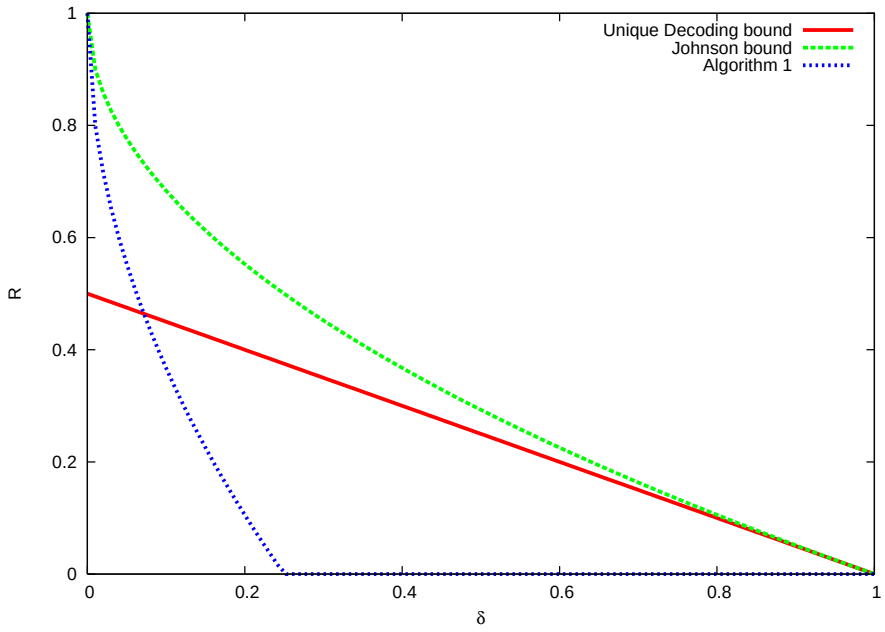


Figure 13.4: The tradeoff between rate R and the fraction of errors that can be corrected by Algorithm 14.

13.2.3 Algorithm 2

To motivate the next algorithm, recall that in Algorithm 14, in order to prove that the root finding step (Steps 3-6 in Algorithm 14) works, we defined a polynomial $R(X) \stackrel{\text{def}}{=} Q(X, P(X))$. In particular, this implied that $\deg(R) \leq \deg_X(Q) + (k-1) \cdot \deg_Y(Q)$ (and we had to select $t > \deg_X(Q) + (k-1) \cdot \deg_Y(Q)$). One shortcoming of this approach is that the maximum degree of X and Y might not occur in the same term. For example, in the polynomial $X^2Y^3 + X^4Y^2$, the maximum X and Y degrees do not occur in the same monomial. The main insight in the new algorithm is to use a more "balanced" notion of degree of $Q(X,Y)$:

Definition 13.2.2. The $(1,w)$ weighted degree of the monomial X^iY^j is $i + wj$. Further, the $(1,w)$-weighted degree of $Q(X,Y)$ (or just its $(1,w)$ degree) is the maximum $(1,w)$ weighted degree of its monomials.

For example, the $(1,2)$-degree of the polynomial $XY^3 + X^4Y$ is $\max(1+3\cdot 2, 4+2\cdot 1) = 7$. Also note that the $(1,1)$-degree of a bivariate polynomial $Q(X,Y)$ is its total degree (or the "usual" definition of degree of a bivariate polynomial). Finally, we will use the following simple lemma (whose proof we leave as an exercise):

Lemma 13.2.2. *Let $Q(X,Y)$ be a bivariate polynomial of $(1,w)$ degree D. Let $P(X)$ be a polynomial such that $\deg(P) \leq w$. Then we have*

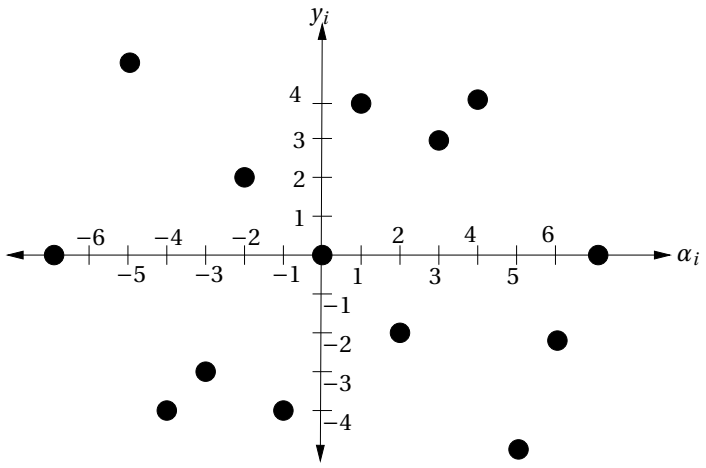
$$\deg(Q(X,P(X))) \leq D.$$

Note that a bivariate polynomial $Q(X,Y)$ of $(1,w)$ degree at most D can be represented as follows:

$$Q(X,Y) \stackrel{\text{def}}{=} \sum_{\substack{i+wj \leq D \\ i,j \geq 0}} c_{i,j} X^i Y^j,$$

where $c_{i,j} \in \mathbb{F}_q$.

The new algorithm is basically the same as Algorithm 14, except that in the interpolation step, we compute a bivariate polynomial of bounded $(1,k-1)$ degree. Before we state the precise algorithm, we will present the algorithm via an example. Consider the received word in Figure 13.5.



$n = 14, k = 2, e = 9$

Figure 13.5: An illustration of a received word for the $[14,2]$ Reed-Solomon code from Figure 13.1 (where again we have implicitly embedded the field $\mathbb{F}_q$ in the set $\{-7,\ldots,7\}$). Here we have considered $e = 9$ errors which is more than what Algorithm 13 can handle. In this case, we are looking for lines that pass through at least 5 points.

Now we want to interpolate a bivariate polynomial $Q(X,Y)$ with a $(1,1)$ degree of 4 that "passes" through all the 2-D points corresponding to the received word from Figure 13.5. Figure 13.6 shows such an example.

Finally, we want to factorize all the linear factors $Y - P(X)$ of the $Q(X,Y)$ from Figure 13.6. Figure 13.7 shows the two polynomials X and $-X$ such that $Y - X$ and $Y + X$ are factors of $Q(X,Y)$ from Figure 13.6.

We now precisely state the new list decoding algorithm in Algorithm 15.

Proof of Correctness of Algorithm 15. As in the case of Algorithm 14, to prove the correctness of Algorithm 15, we need to do the following:

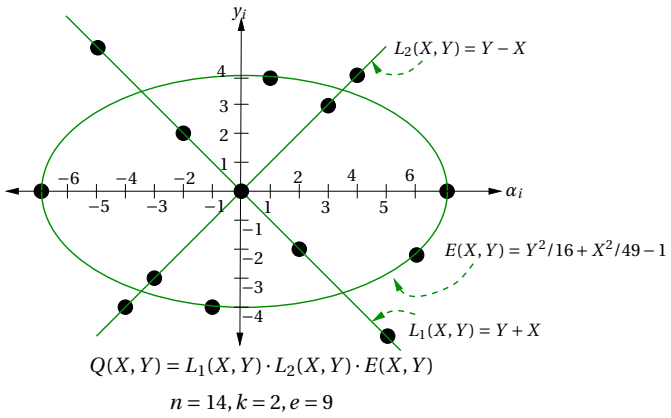


Figure 13.6: An interpolating polynomial $Q(X,Y)$ for the received word in Figure 13.5.

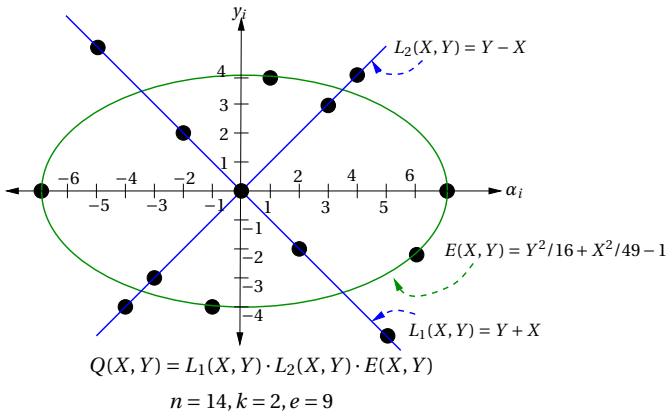


Figure 13.7: The two polynomials that need to be output are shown in blue.

- (Interpolation Step) Ensure that the number of coefficients of $Q(X,Y)$ is strictly greater than n.
- (Root Finding Step) Let $R(X) \stackrel{\text{def}}{=} Q(X, P(X))$. We want to show that if $P(\alpha_i) \geq y_i$ for at least t values of i, then $R(X) \equiv 0$.

To begin with, we argue why we can prove the correctness of the root finding step. Note that since $Q(X,Y)$ has $(1, k-1)$ degree at most D, Lemma 13.2.2 implies that

$$\deg(R) \leq D.$$

Then using the same argument as we used for the correctness of the root finding step of Algorithm 14, we can ensure $R(X) \equiv 0$ if we pick

$$t > D.$$

Algorithm 15 The Second List Decoding Algorithm for Reed-Solomon Codes

INPUT: $n \geq k \geq 1$, $D \geq 1$, $e = n - t$ and n pairs $\{(\alpha_i, y_i)\}_{i=1}^{n}$
OUTPUT: (Possibly empty) list of polynomials $P(X)$ of degree at most $k-1$

1: Find a non-zero $Q(X,Y)$ with $(1, k-1)$ degree at most D, such that

$$Q(\alpha_i, y_i) = 0, 1 \leq i \leq n. \tag{13.9}$$

2: Ł $\leftarrow \emptyset$
3: FOR every factor $Y - P(X)$ of $Q(X,Y)$ DO
4: IF $\Delta(\mathbf{y}, (P(\alpha_i))_{i=1}^{n}) \leq e$ and $\deg(P) \leq k-1$ THEN
5: Add $P(X)$ to Ł.
6: RETURN Ł

Thus, we would like to pick D to be as small as possible. On the other hand, Step 1 will need D to be large enough (so that the number of variables is more than the number of constraints in (13.9). Towards that end, let the number of coefficients of $Q(X,Y)$ be

$$\mathcal{N} = \left|\left\{(i,j) | i + (k-1)j \leq D, i, j \in \mathbb{Z}^+\right\}\right|$$

To bound $\mathcal{N}$, we first note that in the definition above, $j \leq \lfloor \frac{D}{k-1} \rfloor$. (For notational convenience, define $\ell = \lfloor \frac{D}{k-1} \rfloor$.) Consider the following sequence of relationships

$$\begin{aligned}
\mathcal{N} &= \sum_{j=1}^{\ell} \sum_{i=0}^{D-(k-1)j} 1 \\
&= \sum_{j=0}^{\ell} (D - (k-1)j + 1) \\
&= \sum_{j=0}^{\ell} (D+1) - (k-1)\sum_{j=0}^{\ell} j \\
&= (D+1)(\ell+1) - \frac{(k-1)\ell(\ell+1)}{2} \\
&= \frac{\ell+1}{2}(2D + 2 - (k-1)\ell) \\
&\geq \left(\frac{\ell+1}{2}\right)(D+2) \qquad (13.10) \\
&\geq \frac{D(D+2)}{2(k-1)}. \qquad (13.11)
\end{aligned}$$

In the above, (13.10) follows from the fact that $\ell \leq \frac{D}{k-1}$ and (13.11) follows from the fact that $\frac{D}{k-1} - 1 \leq \ell$.

Thus, the interpolation step succeeds (i.e. there exists a non-zero $Q(X,Y)$ with the required properties) if

$$\frac{D(D+2)}{2(k-1)} > n.$$

The choice

$$D = \left\lceil \sqrt{2(k-1)n} \right\rceil$$

suffices by the following argument:

$$\frac{D(D+2)}{2(k-1)} > \frac{D^2}{2(k-1)} \geq \frac{2(k-1)n}{2(k-1)} = n.$$

Thus for the root finding step to work, we need $t > \left\lceil \sqrt{2(k-1)n} \right\rceil$, which implies the following result:

Theorem 13.2.3. *Algorithm* 2 *can list decode Reed-Solomon codes of rate R from up to $1-\sqrt{2R}$ fraction of errors. Further, the algorithm runs in polynomial time.*

Algorithm 2 runs in polynomial time as Step 1 can be implemented using Gaussian elimination (and the fact that the number of coefficients is $O(n)$) while the root finding step can be implemented by any polynomial time algorithm to factorize bivariate polynomials. Further, we note that $1-\sqrt{2R}$ beats the unique decoding bound of $(1-R)/2$ for $R < 1/3$. See Figure 13.2.3 for an illustration.

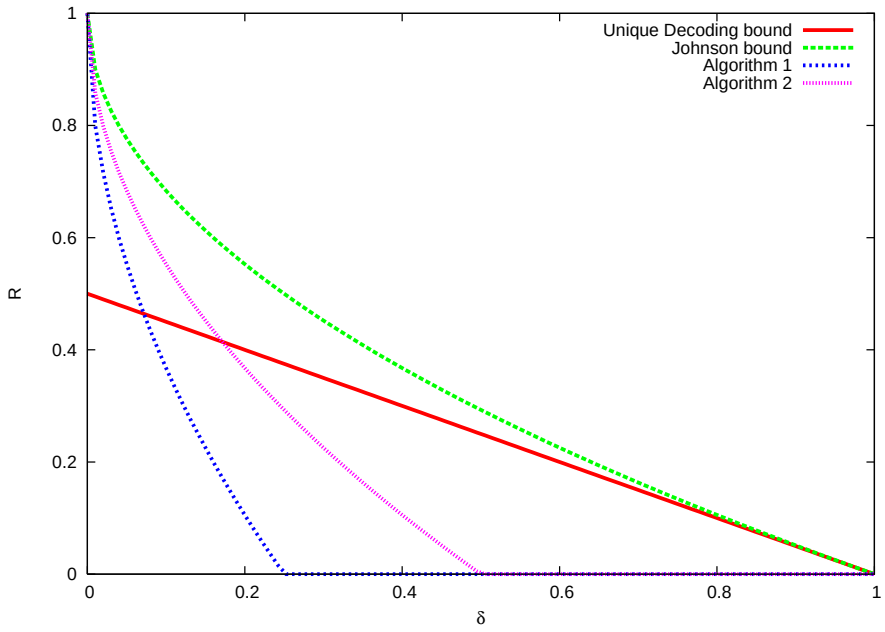


Figure 13.8: The tradeoff between rate R and the fraction of errors that can be corrected by Algorithm 14 and Algorithm 15.

13.2.4 Algorithm 3

Finally, we present the list decoding algorithm for Reed-Solomon codes, which can correct $1 - \sqrt{R}$ fraction of errors. The main idea is to add more restrictions on $Q(X,Y)$ (in addition to its $(1,k-1)$-degree being at most D). This change will have the following implications:

- The number of constraints will increase but the number of coefficients will remain the same. This seems to be bad as this results in an increase in D (which in turn would result in an increase in t).
- However, this change will also increases the number of roots of $R(X)$ and this gain in the number of roots more than compensates for the increase in D.

In particular, the constraint is as follows. For some integer parameter $r \geq 1$, we will insist on $Q(X,Y)$ having r roots at $(\alpha_i, y_i), 1 \leq i \leq n$.

To motivate the definition of multiplicity of a root of a bivariate polynomial, let us consider the following simplified examples. In Figure 13.9 the curve $Q(X,Y) = Y - X$ passes through the

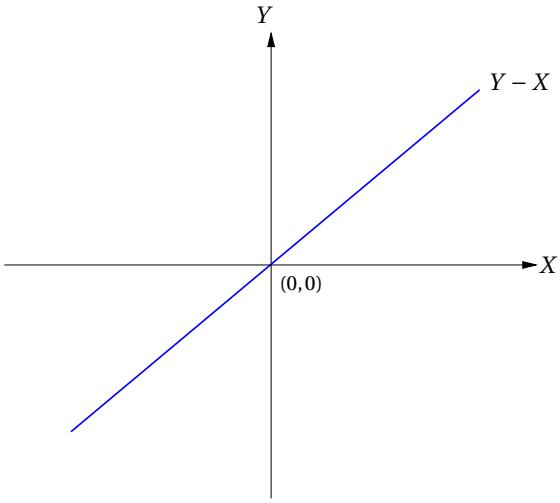


Figure 13.9: Multiplicity of 1

origin once and has no term of degree 0.

In Figure 13.10, the curve $Q(X,Y) = (Y-X)(Y+X)$ passes though the origin twice and has no term with degree at most 1.

In Figure 13.11, the curve $Q(X,Y) = (Y-X)(Y+X)(Y-2X)$ passes through the origin thrice and has no term with degree at most 2. More generally, if r lines pass through the origin, then note that the curve corresponding to their product has no term with degree at most $r-1$. This leads to the following more general definition:

Definition 13.2.3. $Q(X,Y)$ has r roots at $(0,0)$ if $Q(X,Y)$ doesn't have any monomial with degree at most $r-1$.

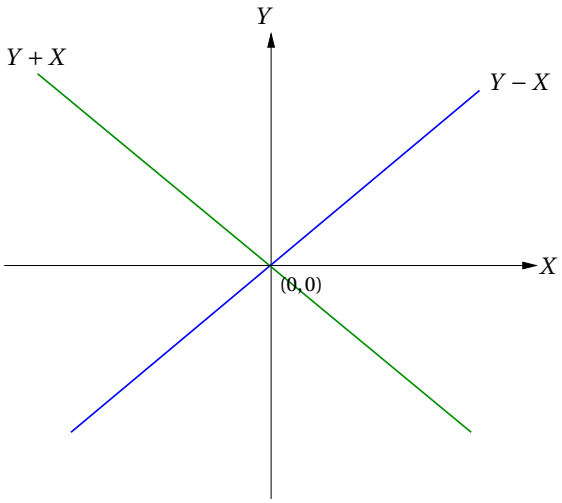


Figure 13.10: Multiplicity of 2

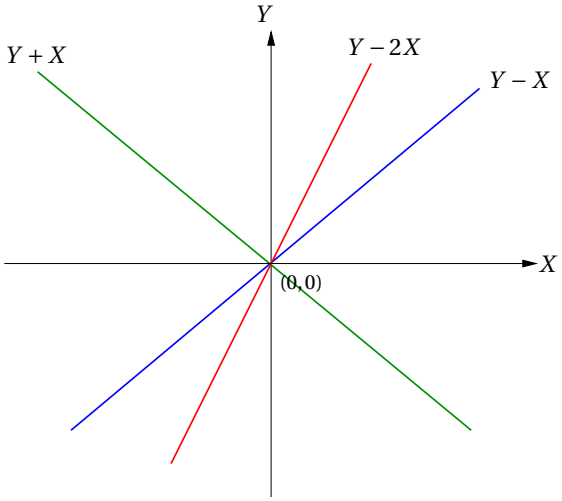


Figure 13.11: Multiplicity of 3

The definition of a root with multiplicity r at a more general point follows from a simple translation:

Definition 13.2.4. $Q(X,Y)$ has r roots at (α,β) if $Q_{\alpha,\beta}(X,Y) \stackrel{\text{def}}{=} Q(x+\alpha, y+\beta)$ has r roots at $(0,0)$.

Before we state the precise algorithm, we will present the algorithm with an example. Consider the received word in Figure 13.12.

Now we want to interpolate a bivariate polynomial $Q(X,Y)$ with $(1,1)$ degree 5 that "passes twice" through all the 2-D points corresponding to the received word from Figure 13.12. Figure 13.13 shows such an example.

Finally, we want to factorize all the linear factors $Y - P(X)$ of the $Q(X,Y)$ from Figure 13.13. Figure 13.14 shows the five polynomials of degree one are factors of $Q(X,Y)$ from Figure 13.13.

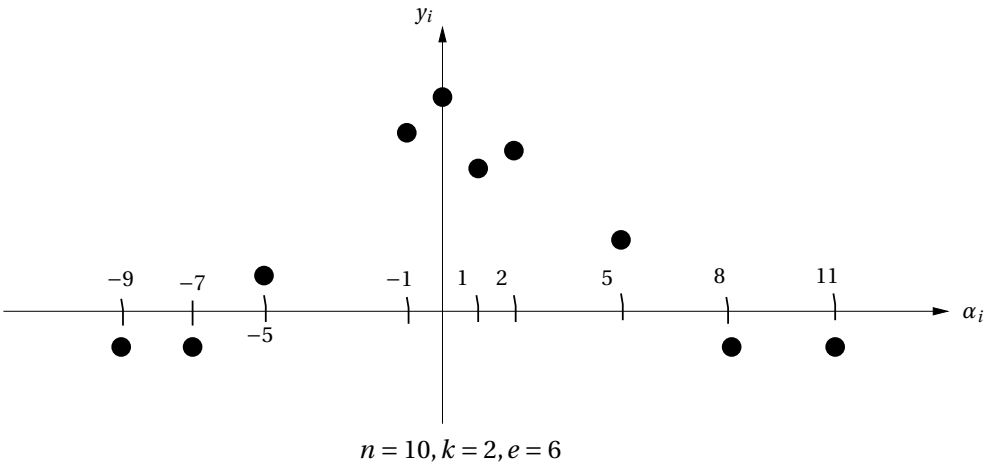


Figure 13.12: An illustration of a received word for the $[10,2]$ Reed-Solomon code (where we have implicitly embedded the field $\mathbb{F}_q$ in the set $\{-9,\ldots,11\}$). Here we have considered $e = 6$ errors which is more than what Algorithm 15 can decode. In this case, we are looking for lines that pass through at least 4 points.

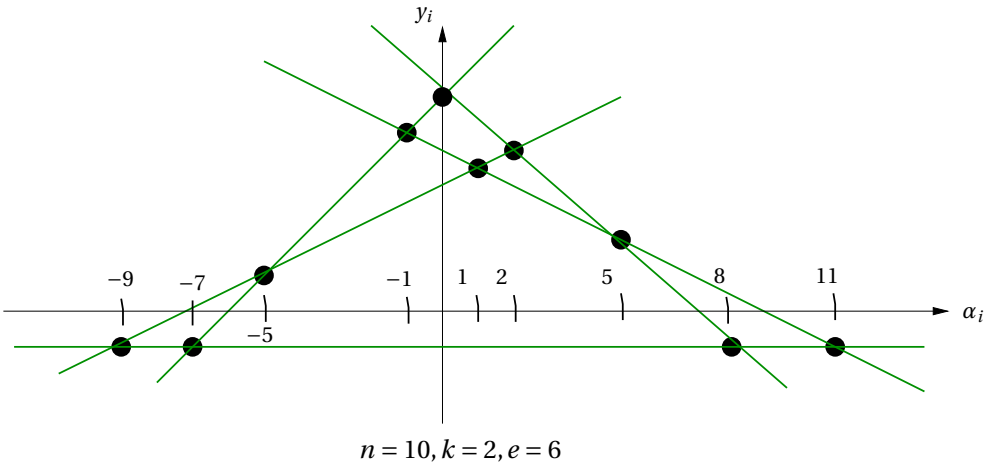


Figure 13.13: An interpolating polynomial $Q(X,Y)$ for the received word in Figure 13.12.

(In fact, $Q(X,Y)$ exactly decomposes into the five lines.)
Algorithm 16 formally states the algorithm.

Correctness of Algorithm 16. To prove the correctness of Algorithm 16, we will need the following two lemmas (we defer the proofs of the lemmas above to Section 13.2.4):

Lemma 13.2.4. *The constraints in (13.12) imply* $\binom{r+1}{2}$ *constraints for each* i *on the coefficients of* $Q(X,Y)$.

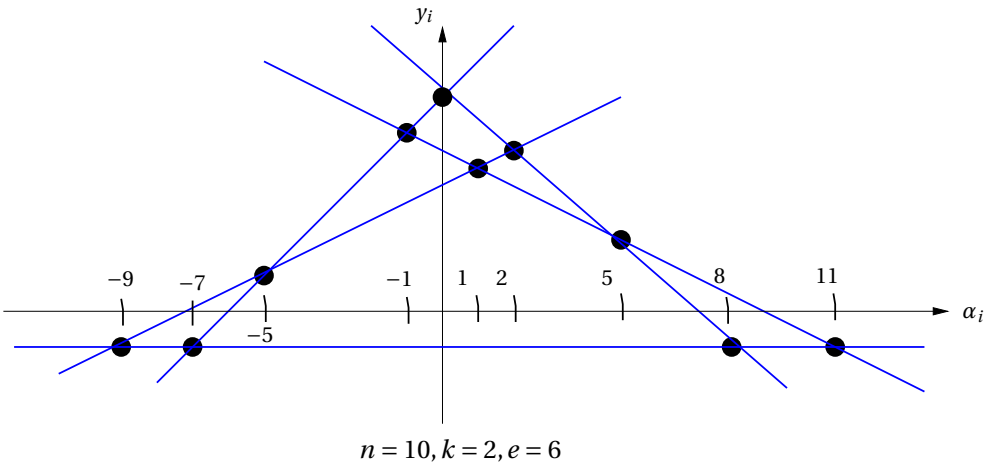


Figure 13.14: The five polynomials that need to be output are shown in blue.

Algorithm 16 The Third List Decoding Algorithm for Reed-Solomon Codes

INPUT: $n \geq k \geq 1$, $D \geq 1$, $r \geq 1$, $e = n - t$ and n pairs $\{(\alpha_i, y_i)\}_{i=1}^{n}$
OUTPUT: (Possibly empty) list of polynomials $P(X)$ of degree at most $k-1$

1: Find a non-zero $Q(X,Y)$ with $(1, k-1)$ degree at most D, such that

$$Q(\alpha_i, y_i) = 0, \text{ with multiplicity } r \text{ for every } 1 \leq i \leq n. \tag{13.12}$$

2: Ł $\leftarrow \emptyset$
3: FOR every factor $Y - P(X)$ of $Q(X,Y)$ DO
4: IF $\Delta(\mathbf{y}, (P(\alpha_i))_{i=1}^{n}) \leq e$ and $\deg(P) \leq k-1$ THEN
5: Add $P(X)$ to Ł.
6: RETURN Ł

Lemma 13.2.5. *$R(X) \stackrel{\text{def}}{=} Q(X, P(X))$ has r roots for every i such that $P(\alpha_i) = y_i$. In other words, $(X - \alpha_i)^r$ divides $R(X)$.*

Using arguments similar to those used for proving the correctness of Algorithm 15, to argue the correctness of the interpolations step we will need

$$\frac{D(D+2)}{2(k-1)} > n\binom{r+1}{2},$$

where the LHS is an upper bound on the number of coefficients of $Q(X,Y)$ as before from (13.11) and the RHS follows from Lemma 13.2.4. We note that the choice

$$D = \left\lceil \sqrt{(k-1)nr(r-1)} \right\rceil$$

works. Thus, we have shown the correctness of Step 1.

For the correctness of the root finding step, we need to show that the number of roots of $R(X)$ (which by Lemma 13.2.5 is at least rt) is strictly bigger than the degree of $R(X)$, which from Lemma 13.2.2 is D. That is we would be fine we if have,

$$tr > D,$$

which is the same as

$$t > \frac{D}{r},$$

which in turn will follow if we pick

$$t = \left\lceil \sqrt{(k-1)n\left(1-\frac{1}{r}\right)} \right\rceil.$$

If we pick $r = 2(k-1)n$, then we will need

$$t > \left\lceil \sqrt{(k-1)n - \frac{1}{2}} \right\rceil > \left\lceil \sqrt{(k-1)n} \right\rceil,$$

where the last inequality follows because of the fact that t is an integer. Thus, we have shown

Theorem 13.2.6. *Algorithm 16 can list decode Reed-Solomon codes of rate R from up to $1-\sqrt{R}$ fraction of errors. Further, the algorithm runs in polynomial time.*

The claim on the run time follows from the same argument that was used to argue the polynomial running time of Algorithm 15. Thus, Theorem 13.2.6 shows that Reed-Solomon codes can be efficiently decoded up to the Johnson bound. For an illustration of fraction of errors correctable by the three list decoding algorithms we have seen, see Figure 13.2.3.

A natural question to ask is if Reed-Solomon codes of rate R can be list decoded beyond $1-\sqrt{R}$ fraction of errors. The answer is still not known:

Open Question 13.2.1. *Given a Reed-Solomon code of rate R, can it be efficiently list decoded beyond $1-\sqrt{R}$ fraction of errors?*

Recall that to complete the proof of Theorem 13.2.6, we still need to prove Lemmas 13.2.4 and 13.2.5, which we do next.

Proof of key lemmas

Proof of Lemma 13.2.4. Let

$$Q(X,Y) = \sum_{\substack{i,j \\ i+(k-1)j \le D}} c_{i,j} X^i Y^j$$

and

$$Q_{\alpha,\beta}(X,Y) = Q(X+\alpha, Y+\beta) = \sum_{i,j} c_{i,j}^{\alpha,\beta} X^i Y^j.$$

We will show that

(i) $c_{i,j}^{\alpha,\beta}$ are homogeneous linear combinations of $c_{i,j}$'s.

(ii) If $Q_{\alpha,\beta}(X,Y)$ has no monomial with degree $< r$, then that implies $\binom{r+1}{2}$ constraints on $c_{i,j}^{\alpha,\beta}$'s.

Note that (i) and (ii) prove the lemma. To prove (i), note that by the definition:

$$Q_{\alpha,\beta}(X,Y) = \sum_{i,j} c_{i,j}^{\alpha,\beta} X^i Y^j \tag{13.13}$$

$$= \sum_{\substack{i',j' \\ i'+(k-1)j' \le D}} c_{i',j'} (X+\alpha)^{i'} (Y+\beta)^{j'} \tag{13.14}$$

Note that, if $i > i'$ or $j > j'$, then $c_{i,j}^{\alpha,\beta}$ doesn't depend on $c^{i',j'}$. By comparing coefficients of $X^i Y^j$ from (13.13) and (13.14), we obtain

$$c_{i,j}^{\alpha,\beta} = \sum_{\substack{i'>i \\ j'>j}} c_{i',j'} \binom{i'}{i} \binom{j'}{j} \alpha^i \beta^j,$$

which proves (i). To prove (ii), recall that by definition $Q_{\alpha,\beta}(X,Y)$ has no monomial of degree $< r$. In other words, we need to have constraints $c_{i,j}^{\alpha,\beta} = 0$ if $i + j \le r-1$. The number of such constraints is

$$|\{(i,j) | i+j \le r-1,\ i,j \in \mathbb{Z}^{\ge 0}\}| = \binom{r+1}{2},$$

where the equality follows from the following argument. Note that for every fixed value of $0 \le j \le r-1$, i can take $r-j$ values. Thus, we have that the number of constraints is

$$\sum_{j=0}^{r-1} r - j = \sum_{\ell=1}^{r} \ell = \binom{r+1}{2},$$

as desired. □

We now re-state Lemma 13.2.5 more precisely and then prove it.

Lemma 13.2.7. *Let $Q(X,Y)$ be computed by Step 1 in Algorithm 16. Let $P(X)$ be a polynomial of degree $\le k-1$, such that $P(\alpha_i) = y_i$ for at least $t > \frac{D}{r}$ many values of i, then $Y - P(X)$ divides $Q(X,Y)$.*

Proof. Define

$$R(X) \stackrel{\text{def}}{=} Q(X, P(X)).$$

As usual, to prove the lemma, we will show that $R(X) \equiv 0$. To do this, we will use the following claim.

Claim 13.2.8. *If $P(\alpha_i) = y_i$, then $(X - \alpha_i)^r$ divides $R(X)$, that is α_i is a root of $R(X)$ with multiplicity r.*

Note that by definition of $Q(X, Y)$ and $P(X)$, $R(X)$ has degree $\leq D$. Assuming the above claim is correct, $R(X)$ has at least $t \cdot r$ roots. Therefore, by the degree mantra (Proposition 5.2.3), $R(X)$ is a zero polynomial as $t \cdot r > D$. We will now prove Claim 13.2.8. Define

$$P_{\alpha_i, y_i}(X) \stackrel{\text{def}}{=} P(X + \alpha_i) - y_i, \tag{13.15}$$

and

$$\begin{aligned} R_{\alpha_i, y_i}(X) &\stackrel{\text{def}}{=} R(X + \alpha_i) && (13.16)\\ &= Q(X + \alpha_i, P(X + \alpha_i)) && (13.17)\\ &= Q(X + \alpha_i, P_{\alpha_i, y_i}(X) + y_i) && (13.18)\\ &= Q_{\alpha_i, y_i}(X, P_{\alpha_i, y_i}(X)), && (13.19) \end{aligned}$$

where (13.17), (13.18) and (13.19) follow from the definitions of $R(X)$, $P_{\alpha_i, y_i}(X)$ and $Q_{\alpha_i, y_i}(X, Y)$ respectively.

By (13.16) if $R_{\alpha_i, y_i}(0) = 0$, then $R(\alpha_i) = 0$. So, if X divides $R_{\alpha_i, y_i}(X)$, then $X - \alpha_i$ divides $R(X)$. (This follows from a similar argument that we used to prove Proposition 5.2.3.) Similarly, if X^r divides $R_{\alpha_i, y_i}(X)$, then $(X - \alpha_i)^r$ divides $R(X)$. Thus, to prove the lemma, we will show that X^r divides $R_{\alpha_i, y_i}(X)$. Since $P(\alpha_i) = y_i$ when α_i agrees with y_i, we have $P_{\alpha_i, y_i}(0) = 0$. Therefore, X is a root of $P_{\alpha_i, y_i}(X)$, that is, $P_{\alpha_i, y_i}(X) = X \cdot g(X)$ for some polynomial $g(X)$ of degree at most $k - 1$. We can rewrite

$$R_{\alpha_i, y_i}(X) = \sum_{i', j'} c_{i', j'}^{\alpha_i, y_i} X^{i'} (P_{\alpha_i, y_i}(X))^{j'} = \sum_{i', j'} c_{i', j'}^{\alpha_i, y_i} X^{i'} (X g(X))^{j'}.$$

Now for every i', j' such that $c_{i', j'}^{\alpha_i, y_i} \neq 0$, we have $i' + j' \geq r$ as $Q_{\alpha_i, y_i}(X, Y)$ has no monomial of degree $< r$. Thus X^r divides $R_{\alpha_i, y_i}(X)$, since $R_{\alpha_i, y_i}(x)$ has no non-zero monomial X^ℓ for any $\ell < r$. □

13.3 Extensions

We now make some observations about Algorithm 16. In particular, the list decoding algorithm is general enough to solve more general problems than just list decoding. In this section, we present an overview of these extensions.

Recall that the constraint (13.12) states that $Q(X,Y)$ has $r \geq 0$ roots at (α_i, y_i), $1 \leq i \leq n$. However, our analysis did not explicitly use the fact that the multiplicity is same for every i. In particular, given non-zero integer multiplicities $w_i \geq 0$, $1 \leq i \leq n$, Algorithm 16 can be generalized to output all polynomials $P(X)$ of degree at most $k-1$, such that

$$\sum_{i:P(\alpha_i)=y_i} w_i > \sqrt{(k-1)n \sum_{i=0}^{n} \binom{w_i+1}{2}}.$$

(We leave the proof as an exercise.) Note that till now we have seen the special case $w_i = r$, $1 \leq i \leq n$.

Further, we claim that the α_i's need not be distinct for the all of the previous arguments to go through. In particular, one can generalize Algorithm 16 even further to prove the following (the proof is left as an exercise):

Theorem 13.3.1. *Given integer weights $w_{i,\alpha}$ for every $1 \leq i \leq n$ and $\alpha \in \mathbb{F}$, in polynomial time one can output all $P(X)$ of degree at most $k-1$ such that*

$$\sum_{i} w_{i,P(\alpha_i)} > \sqrt{(k-1)n \sum_{i=0}^{n} \sum_{\alpha \in \mathbb{F}} \binom{w_{i,\alpha}+1}{2}}.$$

This will be useful to solve the following generalization of list decoding called soft decoding.

Definition 13.3.1. Under soft decoding problem, the decoder is given as input a set of non-negative weights $w_{i,d} (1 \leq i \leq n, \alpha \in \mathbb{F}_q)$ and a threshold $W \geq 0$. The soft decoder needs to output all codewords $(c_1, c_2, \ldots, c_n)$ in q-ary code of block length n that satisfy:

$$\sum_{i=1}^{n} w_{i,c_i} \geq W.$$

Note that Theorem 13.3.1 solve the soft decoding problem with

$$W = \sqrt{(k-1)n \sum_{i=0}^{n} \sum_{\alpha \in \mathbb{F}} \binom{w_{i,\alpha}+1}{2}}.$$

Consider the following special case of soft decoding where $w_{i,y_i} = 1$ and $w_{i,\alpha} = 0$ for $\alpha \in \mathbb{F} \setminus \{y_i\}$ $(1 \leq i \leq n)$. Note that this is exactly the list decoding problem with the received word $(y_1, \ldots, y_n)$. Thus, list decoding is indeed a special case of soft decoding. Soft decoding has practical applications in settings where the channel is analog. In such a situation, the "quantizer" might not be able to pinpoint a received symbol y_i with 100% accuracy. Instead, it can use the weight $w_{i,\alpha}$ to denote its confidence level that ith received symbol was α.

Finally, we consider a special case of soft called list recovery, which has applications in designing list decoding algorithms for concatenated codes.

Definition 13.3.2 (List Recovery)**.** Given $S_i \subseteq \mathbb{F}_q$, $1 \leq i \leq n$ where $|S_i| \leq \ell$, output all $[n,k]_q$ codewords $(c_1,\ldots,c_n)$ such that $c_i \in S_i$ for at least t values of i. If for every valid input the number of such codewords is at most L, then the corresponding code is called $(1-t/n,\ell,L)$-list recoverable.

We leave the proof that list decoding is a special case of soft decoding as an exercise. Finally, we claim that Theorem 13.3.1 implies the following result for list recovery (the proof is left as an exercise):

Theorem 13.3.2. *Given $t > \sqrt{(k-1)\ell n}$, the list recovery problem with agreement parameter t for $[n,k]_q$ Reed-Solomon codes can be solved in polynomial time.*

13.4 Bibliographic Notes

In 1960, before polynomial time complexity was regarded as an acceptable notion of efficiency, Peterson designed an $O(N^3)$ time algorithm for the unique decoding of Reed-Solomon codes [60]. This algorithm was the first efficient algorithm for unique decoding of Reed-Solomon codes. The Berlekamp-Massey algorithm, which used shift registers for multiplication, was even more efficient, achieving a computational complexity of $O\left(N^2\right)$. Currently, an even more efficient algorithm, with a computational complexity of $O\left(N\text{poly}(\log N)\right)$, is known. This algorithm is the topic of one of the research survey reports for the course this semester.

The Welch-Berlekamp algorithm, covered under US Patent [77], has a running time complexity of $O\left(N^3\right)$. We will follow a description of the Welch-Berlekamp algorithm provided by Gemmell and Sudan in [23].

Håstad, Philips and Safra showed that solving a system of quadratic equations (even those without any square terms like we have in (13.1)) over any filed $\mathbb{F}_q$ is NP-hard [41]. (In fact, it is even hard to approximately solve this problem: i.e. where on tries to compute an assignment that satisfies as many equations as possible.) Linearization is a trick that has been used many times in theoretical computer science and cryptography. See this blog post by Dick Lipton for more on this.

Algorithm 15 is due to Sudan [71] and Algorithm 16 is due to Guruswami and Sudan [37].

It is natural to ask whether Theorem 13.3.2 is tight for list recovery, i.e. generalize Open Question 13.2.1 to list recovery. It was shown by Guruswami and Rudra that Theorem 13.3.2 is indeed the best possible list recovery result for Reed-Solomon codes [32]. Thus, any algorithm that answers Open Question 13.2.1 in some sense has to exploit the fact that in the list decoding problem, the α_i's are distinct.

Chapter 14

Efficiently Achieving List Decoding Capacity

In the previous chapters, we have seen these results related to list decoding:

- Reed-Solomon codes of rate $R > 0$ can be list-decoded in polynomial time from $1 - \sqrt{R}$ errors (Theorem 13.2.6). This is the best algorithmic list decoding result we have seen so far.
- There exist codes of rate $R > 0$ that are $\left(1 - R - \varepsilon, O\left(\frac{1}{\varepsilon}\right)\right)$-list decodable for $q \geq 2^{\Omega(\frac{1}{\varepsilon})}$ (and in particular for $q = \text{poly}(n)$) (Theorem 7.4.1 and Proposition 3.3.2). This of course is the best possible combinatorial result.

Note that there is a gap between the algorithmic result and the best possible combinatorial result. This leads to the following natural question:

Question 14.0.1. *Are there explicit codes of rate $R > 0$ that can be list-decoded in polynomial time from $1 - R - \varepsilon$ fraction of errors for $q \leq poly(n)$?*

In this chapter, we will answer Question 14.0.1 in the affirmative.

14.1 Folded Reed-Solomon Codes

We will now introduce a new type of code called the Folded Reed-Solomon codes. These codes are constructed by combining every m consecutive symbols of a regular Reed-Solomon code into one symbol from a larger alphabet. Note that we have already seen such a folding trick when we instantiated the outer code in the concatenated code that allowed us to efficiently achieve the BSC_p capacity (Section 12.4.1). For a Reed-Solomon code that maps $\mathbb{F}_q^k \to \mathbb{F}_q^n$, the corresponding Folded Reed-Solomon code will map $\mathbb{F}_q^k \to \mathbb{F}_{q^m}^{n/m}$. We will analyze Folded Reed-Solomon codes that are derived from Reed-Solomon codes with evaluation $\{1, \gamma, \gamma^2, \gamma^3, \ldots, \gamma^{n-1}\}$, where γ is the generator of $\mathbb{F}_q^*$ and $n \leq q - 1$. Note that in the Reed-Solomon code, a message is encoded as in Figure 14.1.

$f(1)$	$f(\gamma)$	$f(\gamma^2)$	$f(\gamma^3)$	$\cdots$	$f(\gamma^{n-2})$	$f(\gamma^{n-1})$

Figure 14.1: Encoding $f(X)$ of degree $\leq k-1$ and coefficients in $\mathbb{F}_q$ corresponding to the symbols in the message.

For $m = 2$, the conversion from Reed-Solomon to Folded Reed-Solomon can be visualized as in Figure 14.2 (where we assume n is even).

$f(1)$	$f(\gamma)$	$f(\gamma^2)$	$f(\gamma^3)$	$\cdots$	$f(\gamma^{n-2})$	$f(\gamma^{n-1})$

$\Downarrow$

$f(1)$ $f(\gamma)$	$f(\gamma^2)$ $f(\gamma^3)$	$\cdots$	$f(\gamma^{n-2})$ $f(\gamma^{n-1})$

Figure 14.2: Folded Reed-Solomon code for $m = 2$

For general $m \geq 1$, this transformation will be as in Figure 14.3 (where we assume that m divides n).

$f(1)$	$f(\gamma)$	$f(\gamma^2)$	$f(\gamma^3)$	$\cdots$	$f(\gamma^{n-2})$	$f(\gamma^{n-1})$

$\Downarrow$

$f(1)$ $f(\gamma)$ $\vdots$ $f(\gamma^{m-1})$	$f(\gamma^m)$ $f(\gamma^{m+1})$ $\vdots$ $f(\gamma^{2m-1})$	$f(\gamma^{2m})$ $f(\gamma^{2m+1})$ $\vdots$ $f(\gamma^{3m-1})$	$\cdots$	$f(\gamma^{n-m})$ $f(\gamma^{n-m+1})$ $\vdots$ $f(\gamma^{n-1})$

Figure 14.3: Folded Reed-Solomon code for general $m \geq 1$

More formally, here is the definition of folded Reed-Solomon codes:

Definition 14.1.1 (Folded Reed-Solomon Code)**.** The m-folded version of an $[n,k]_q$ Reed-Solomon code C (with evaluation points $\{1,\gamma,\ldots,\gamma^{n-1}\}$), call it C', is a code of block length $N = n/m$ over $\mathbb{F}_{q^m}$, where $n \leq q-1$. The encoding of a message $f(X)$, a polynomial over $\mathbb{F}_q$ of degree at most $k-1$, has as its j'th symbol, for $0 \leq j < n/m$, the m-tuple $\left(f\left(\gamma^{jm}\right), f\left(\gamma^{jm+1}\right), \cdots, f\left(\gamma^{jm+m-1}\right)\right)$. In other words, the codewords of C' are in one-one correspondence with those of the Reed-Solomon code C and are obtained by bundling together consecutive m-tuple of symbols in codewords of C.

14.1.1 The Intuition Behind Folded Reed-Solomon Codes

We first make the simple observation that the folding trick above cannot *decrease* the list decodability of the code. (We have already seen this argument earlier in Section 12.4.1.)

Claim 14.1.1. *If the Reed-Solomon code can be list-decoded from ρ fraction of errors, then the corresponding folded Reed-Solomon code with folding parameter m can also be list-decoded from ρ fraction of errors.*

Proof. The idea is simple: If the Reed-Solomon code can be list decoded from ρ fraction of errors (by say an algorithm $\mathscr{A}$), the Folded Reed-Solomon code can be list decoded by "unfolding" the received word and then running $\mathscr{A}$ on the unfolded received word and returning the resulting set of messages. Algorithm 17 has a more precise statement.

Algorithm 17 Decoding Folded Reed-Solomon Codes by Unfolding

INPUT: $\mathbf{y} = ((y_{1,1},\ldots,y_{1,m}),\ldots,(y_{n/m,1},\ldots,y_{n/m,m})) \in \mathbb{F}_{q^m}^{n/m}$

OUTPUT: A list of messages in $\mathbb{F}_q^k$

1: $\mathbf{y}' \leftarrow (y_{1,1},\ldots,y_{1,m},\ldots,y_{n/m,1},\ldots,y_{n/m,m}) \in \mathbb{F}_q^n$.
2: RETURN $\mathscr{A}(\mathbf{y}')$

The reason why Algorithm 17 works is simple. Let $\mathbf{m} \in \mathbb{F}_q^k$ be a message. Let RS($\mathbf{m}$) and FRS($\mathbf{m}$) be the corresponding Reed-Solomon and folded Reed-Solomon codewords. Now for every $i \in [n/m]$, if $\text{FRS}(\mathbf{m})_i \neq (y_{i,1},\ldots,y_{i,n/m})$ then in the worst-case for every $j \in [n/m]$, $\text{RS}(\mathbf{m})_{(i-1)n/m+j} \neq y_{i,j}$: i.e. one symbol disagreement over $\mathbb{F}_{q^m}$ can lead to at most m disagreements over $\mathbb{F}_q$. See Figure 14.4 for an illustration.

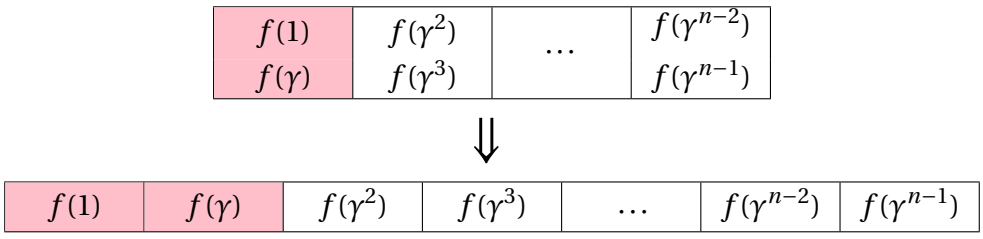


Figure 14.4: Error pattern after unfolding. A pink cell means an error: for the Reed-Solomon code it is for RS($\mathbf{m}$) with $\mathbf{y}'$ and for folded Reed-Solomon code it is for FRS($\mathbf{m}$) with $\mathbf{y}$

This implies that for any $\mathbf{m} \in \mathbb{F}_q^k$ if $\Delta(\mathbf{y}, \text{FRS}(\mathbf{m})) \leq \rho \cdot \frac{n}{m}$, then $\Delta(\mathbf{y}', \text{RS}(\mathbf{m})) \leq m \cdot \rho \cdot \frac{n}{m} = \rho \cdot n$, which by the properties of algorithm $\mathscr{A}$ implies that Step 2 will output $\mathbf{m}$, as desired. □

The intuition for a *strict* improvement by using Folded Reed-Solomon codes is that if the fraction of errors due to folding increases beyond what it can list-decode from, that error pattern does not need to be handled and can be ignored. For example, suppose a Reed-Solomon

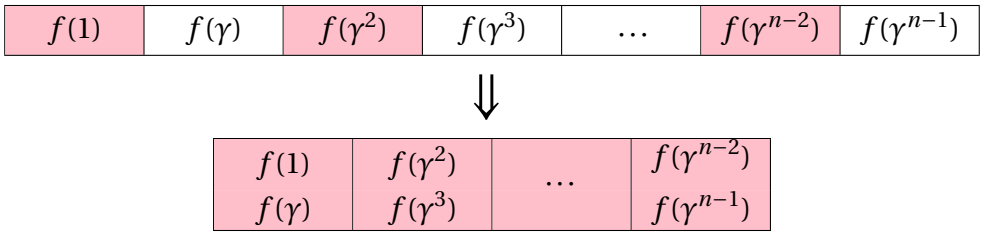


Figure 14.5: An error pattern after folding. The pink cells denotes the location of errors

code that can be list-decoded from up to $\frac{1}{2}$ fraction of errors is folded into a Folded Reed-Solomon code with $m = 2$. Now consider the error pattern in Figure 14.5.

The error pattern for Reed-Solomon code has $\frac{1}{2}$ fraction of errors, so any list decoding algorithm must be able to list-decode from this error pattern. However, for the Folded Reed-Solomon code the error pattern has 1 fraction of errors which is too high for the code to list-decode from. Thus, this "folded" error pattern case can be discarded from the ones that a list decoding algorithm for folded Reed-Solomon code needs to consider. This is of course one example– however, it turns out that this folding operation actually rules out *a lot* of error patterns that a list decoding algorithm for folded Reed-Solomon code needs to handle (even beyond the current best $1 - \sqrt{R}$ bound for Reed-Solomon codes). Put another way, an algorithm for folded Reed-Solomon codes has to solve the list decoding problem for the Reed-Solomon codes where the error patterns are "bunched" together (technically they're called *bursty* errors). Of course, converting this intuition into a theorem takes more work and is the subject of this chapter.

Wait a second... The above argument has a potential hole– what if we take the argument to the extreme and "cheat" by setting $m = n$ where *any* error pattern for the Reed-Solomon code will result in an error pattern with 100% errors for the Folded Reed-Solomon code: thus, we will only need to solve the problem of error detection for Reed-Solomon codes (which we can easily solve for any linear code and in particular for Reed-Solomon codes)? It is a valid concern but we will "close the loophole" by only using a constant m as the folding parameter. This will still keep q to be polynomially large in n and thus, we would still be on track to answer Question 14.0.1. Further, if we insist on smaller list size (e.g. one independent of n), then we can use code concatenation to achieve capacity achieving results for codes over smaller alphabets. (See Section 14.4 for more.)

General Codes. We would like to point out that the folding argument used above is not specific to Reed-Solomon codes. In particular, the argument for the reduction in the number of error patterns holds for any code. In fact, one can prove that for general random codes, with high probability, folding does strictly improve the list decoding capabilities of the original code. (The proof is left as an exercise.)

14.2 List Decoding Folded Reed-Solomon Codes: I

We begin with an algorithm for list decoding folded Reed-Solomon codes that works with agreement $t \sim mRN$. Note that this is a factor m larger than the RN agreement we ultimately want. In the next section, we will see how to knock off the factor of m.

Before we state the algorithm, we formally (re)state the problem we want to solve:

- **Input**: An agreement parameter $0 \leq t \leq N$ and the received word:

$$\mathbf{y} = \begin{pmatrix} y_0 & y_m & & y_{n-m} \\ \vdots & \vdots & \cdots & \vdots \\ y_{m-1} & y_{2m-1} & & y_{n-1} \end{pmatrix} \in \mathbb{F}_q^{m \times N}, \quad N = \frac{n}{m}$$

- **Output**: Return all polynomials $f(X) \in \mathbb{F}_q[X]$ of degree at most $k-1$ such that for at least t values of $0 \leq i < N$

$$\begin{pmatrix} f\left(\gamma^{mi}\right) \\ \vdots \\ f\left(\gamma^{m(i+1)-1}\right) \end{pmatrix} = \begin{pmatrix} y_{mi} \\ \vdots \\ y_{m(i+1)-1} \end{pmatrix} \tag{14.1}$$

The algorithm that we will study is a generalization of the Welch-Berlekamp algorithm (Algorithm 13). However unlike the previous list decoding algorithms for Reed-Solomon codes (Algorithms 14, 15 and 16), this new algorithm has more similarities with the Welch-Berlekamp algorithm. In particular, for $m = 1$, the new algorithm is *exactly* the Welch-Berlekamp algorithm. Here are the new ideas in the algorithm for the two-step framework that we have seen in the previous chapter:

- **Step 1**: We interpolate using $(m+1)$-variate polynomial, $Q(X, Y_1, \ldots, Y_m)$, where degree of each variable Y_i is exactly one. In particular, for $m = 1$, this interpolation polynomial is exactly the one used in the Welch-Berlekamp algorithm.

- **Step 2**: As we have done so far, in this step, we output all "roots" of Q. Two remarks are in order. First, unlike Algorithms 14, 15 and 16, the roots $f(X)$ are no longer simpler linear factors $Y - f(X)$, so one cannot use a factorization algorithm to factorize $Q(X, Y_1, \ldots, Y_m)$. Second, the new insight in this algorithm, is to show that all the roots form an (affine) subspace,[1] which we can use to compute the roots.

Algorithm 18 has the details.

[1] An affine subspace of $\mathbb{F}_q^k$ is a set $\{\mathbf{v} + \mathbf{u} | \mathbf{u} \in S\}$, where $S \subseteq \mathbb{F}_q^k$ is a linear subspace and $\mathbf{v} \in \mathbb{F}_q^k$.

Algorithm 18 The First List Decoding Algorithm for Folded Reed-Solomon Codes

INPUT: An agreement parameter $0 \leq t \leq N$, parameter $D \geq 1$ and the received word:

$$\mathbf{y} = \begin{pmatrix} y_0 & y_m & \cdots & y_{n-m} \\ \vdots & \vdots & & \vdots \\ y_{m-1} & y_{2m-1} & & y_{n-1} \end{pmatrix} \in \mathbb{F}_q^{m \times N}, \quad N = \frac{n}{m}$$

OUTPUT: All polynomials $f(X) \in \mathbb{F}_q[X]$ of degree at most $k-1$ such that for at least t values of $0 \leq i < N$

$$\begin{pmatrix} f\left(\gamma^{mi}\right) \\ \vdots \\ f\left(\gamma^{m(i+1)-1}\right) \end{pmatrix} = \begin{pmatrix} y_{mi} \\ \vdots \\ y_{m(i+1)-1} \end{pmatrix} \tag{14.2}$$

1: Compute a non-zero $Q(X, Y_1, \ldots, Y_m)$ where

$$Q(X, Y_1, \ldots, Y_m) = A_0(X) + A_1(X)\,Y_1 + A_2(X)\,Y_2 + \cdots + A_m(X)\,Y_m$$

with $\deg(A_0) \leq D + k - 1$ and $\deg(A_j) \leq D$ for $1 \leq j \leq m$ such that

$$Q(\gamma^{mi}, y_{mi}, \cdots, y_{m(i+1)-1}) = 0, \ \ \forall 0 \leq i < N \tag{14.3}$$

2: Ł $\leftarrow \emptyset$
3: FOR every $f(X) \in \mathbb{F}_q[X]$ such that $Q(X, f(X), f(\gamma X), f(\gamma^2 X), \ldots, f(\gamma^{m-1} X)) = 0$ DO
4: IF $\deg(f) \leq k-1$ and $f(X)$ satisfies (14.2) for at least t values of i THEN
5: Add $f(X)$ to Ł.
6: RETURN Ł

Correctness of Algorithm 18. In this section, we will only concentrate on the correctness of the algorithm and analyze its error correction capabilities. We will defer the analysis of the algorithm (and in particular, proving a bound on the number of polynomials that are output by Step 6) till the next section.

We first begin with the claim that there always exists a non-zero choice for Q in Step 1 using the same arguments that we have used to prove the correctness of Algorithms 15 and 16:

Claim 14.2.1. *If $(m+1)(D+1) + k - 1 > N$, then there exists a non-zero $Q(X, Y_1, \ldots Y_m)$ that satisfies the required properties of Step* 1.

Proof. As in the proof of correctness of Algorithms 14, 15 and 16, we will think of the constraints in (14.3) as linear equations. The variables are the coefficients of $A_i(X)$ for $0 \leq i \leq m$. With the stipulated degree constraints on the $A_i(X)$'s, note that the number of variables participating in (14.3) is

$$D + k + m(D+1) = (m+1)(D+1) + k - 1.$$

The number of equations is N. Thus, the condition in the claim implies that we have strictly more variables then equations and thus, there exists a non-zero Q with the required properties. □

Next, we argue that the root finding step works (again using an argument very similar to those that we have seen for Algorithms 14, 15 and 16):

Claim 14.2.2. *If $t > D+k-1$, then all polynomial $f(X) \in \mathbb{F}_q[X]$ of degree at most $k-1$ that agree with the received word in at least t positions is returned by Step 6.*

Proof. Define the univariate polynomial

$$R(X) = Q\left(X, f(X), f\left(\gamma X\right), ...f\left(\gamma^{m-1}X\right)\right).$$

Note that due to the degree constraints on the $A_i(X)$'s and $f(X)$, we have

$$\deg(R) \leq D+k-1,$$

since $\deg(f(\gamma^i X)) = \deg(f(X))$. On the other hand, for every $0 \leq i < N$ where (14.1) is satisfied we have

$$R\left(\gamma^{mi}\right) = Q\left(\gamma^{mi}, y_{mi}, \ldots, y_{m(i+1)-1}\right) = 0,$$

where the first equality follows from (14.1), while the second equality follows from (14.3). Thus $R(X)$ has at least t roots. Thus, the condition in the claim implies that $R(X)$ has more roots then its degree and thus, by the degree mantra (Proposition 5.2.3) $R(X) \equiv 0$, as desired. □

Note that Claims 14.2.1 and 14.2.2 prove the correctness of the algorithm. Next we analyze the fraction of errors the algorithm can correct. Note that the condition in Claim 14.2.1 is satisfied if we pick

$$D = \left\lfloor \frac{N-k+1}{m+1} \right\rfloor.$$

This in turn implies that the condition in Claim 14.2.2 is satisfied if

$$t > \frac{N-k+1}{m+1} + k - 1 = \frac{N+m(k-1)}{m+1}.$$

Thus, the above would be satisfied if

$$t \geq \frac{N}{m+1} + \frac{mk}{m+1} = N\left(\frac{1}{m+1} + mR\left(\frac{m}{m+1}\right)\right),$$

where the equality follows from the fact that $k = mRN$.

Note that when $m = 1$, the above bound exactly recovers the bound for the Welch-Berlekamp algorithm (Theorem 13.1.4). Thus, we have shown that

Theorem 14.2.3. *Algorithm 18 can list decode folded Reed-Solomon code with folding parameter $m \geq 1$ and rate R up to $\frac{m}{m+1}(1-mR)$ fraction of errors.*

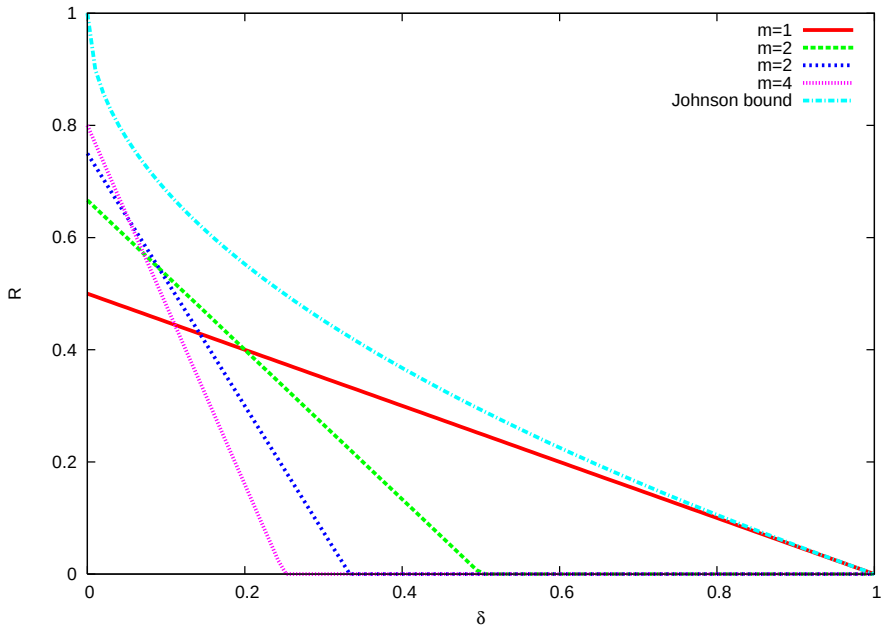


Figure 14.6: The tradeoff between rate R and the fraction of errors that can be corrected by Algorithm 18 for folding parameter $m = 1,2,3$ and 4. The Johnson bound is also plotted for comparison. Also note that the bound for $m = 1$ is the Unique decoding bound achieved by Algorithm 13.

See Figure 14.2 for an illustration of the tradeoff for $m = 1,2,3,4$.

Note that if we can replace the mR factor in the bound from Theorem 14.2.3 by just R then we can approach the list decoding capacity bound of $1-R$. (In particular, we would be able to correct $1-R-\varepsilon$ fraction of errors if we pick $m = O(1/\varepsilon)$.) Further, we need to analyze the number of polynomials output by the root finding step of the algorithm (and then analyze the runtime of the algorithm). In the next section, we show how we can "knock-off" the extra factor m (and we will also bound the list size).

14.3 List Decoding Folded Reed-Solomon Codes: II

In this section, we will present the final version of the algorithm that will allow us to answer Question 14.0.1 in the affirmative. We start off with the new idea that allows us to knock off the factor of m. (It would be helpful to keep the proof of Claim 14.2.2 in mind.)

To illustrate the idea let us consider the folding parameter to be $m = 3$. Let $f(X)$ be a polynomial of degree at most $k-1$ that needs to be output and let $0 \leq i < N$ be a position where it agrees with the received word. (See Figure 14.7 for an illustration.)

The idea is to "exploit" this agreement over *one* $\mathbb{F}_q^3$ symbol and convert it into *two* agreements over $\mathbb{F}_{q^2}$. (See Figure 14.8 for an illustration.)

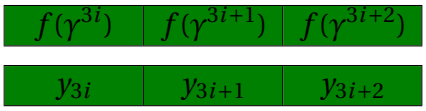


Figure 14.7: An agreement in position i.

Figure 14.8: More agreement with a sliding window of size 2.

Thus, in the proof of Claim 14.2.2, for each agreement we can now get *two* roots for the polynomial $R(X)$. In general for an agreement over one $\mathbb{F}_{q^m}$ symbols translates into $m-s+1$ agreement over $\mathbb{F}_q^s$ for any $1 \le s \le m$ (by "sliding a window" of size s over the m symbols from $\mathbb{F}_q$). Thus, in this new idea the agreement is $m-s+1$ times more than before which leads to the mR term in Theorem 14.2.3 going down to $\frac{mR}{m-s+1}$. Then making s smaller than m but still large enough we can get down the relative agreement to $R+\varepsilon$, as desired. There is another change that needs to be done to make the argument go through: the interpolation polynomial Q now has to be $(s+1)$-variate instead of the earlier $(m+1)$-variate polynomial. Algorithm 19 has the details.

Correctness of Algorithm 19. Next, we analyze the correctness of Algorithm 19 as well as compute its list decoding error bound. We begin with the result showing that there exists a Q with the required properties for Step 1.

Lemma 14.3.1. *If* $D \ge \left\lfloor \frac{N(m-s+1)-k+1}{s+1} \right\rfloor$, *then there exists a non-zero polynomial* $Q(X, Y_1, ..., Y_s)$ *that satisfies* Step 1 *of the above algorithm.*

Proof. Let us consider all coefficients of all polynomials A_i as variables. Then the number of variables will be

$$D + k + s(D+1) = (s+1)(D+1) + k - 1.$$

On the other hand, the number of constraints in (14.5), i.e. the number of equations when all coefficients of all polynomials A_i are considered variables) will be $N(m-s+1)$.

Note that if we have more variables than equations, then there exists a non-zero Q that satisfies the required properties of Step 1. Thus, we would be done if we have:

$$(s+1)(D+1) + k - 1 > N(m-s+1),$$

which is equivalent to:

$$D > \frac{N(m-s+1)-k+1}{s+1} - 1.$$

The choice of D in the statement of the claim satisfies the condition above, which complete the proof. □

Algorithm 19 The Second List Decoding Algorithm for Folded Reed-Solomon Codes

INPUT: An agreement parameter $0 \le t \le N$, parameter $D \ge 1$ and the received word:

$$\mathbf{y} = \begin{pmatrix} y_0 & y_m & \cdots & y_{n-m} \\ \vdots & \vdots & & \vdots \\ y_{m-1} & y_{2m-1} & & y_{n-1} \end{pmatrix} \in \mathbb{F}_q^{m \times N}, \quad N = \frac{n}{m}$$

OUTPUT: All polynomials $f(X) \in \mathbb{F}_q[X]$ of degree at most $k-1$ such that for at least t values of $0 \le i < N$

$$\begin{pmatrix} f\left(\gamma^{mi}\right) \\ \vdots \\ f\left(\gamma^{m(i+1)-1}\right) \end{pmatrix} = \begin{pmatrix} y_{mi} \\ \vdots \\ y_{m(i+1)-1} \end{pmatrix} \tag{14.4}$$

1: Compute non-zero polynomial $Q(X, Y_1, .., Y_s)$ as follows:

$$Q(X, Y_1, .., Y_s) = A_0(X) + A_1(X)Y_1 + A_2(X)Y_2 + .. + A_s(X)Y_s,$$

with $\deg[A_0] \le D + k - 1$ and $\deg[A_i] \le D$ for every $1 \le i \le s$ such that for all $0 \le i < N$ and $0 \le j \le m - s$, we have

$$Q(\gamma^{im+j}, y_{im+j}, ..., y_{im+j+s-1}) = 0. \tag{14.5}$$

2: Ł $\leftarrow \emptyset$

3: FOR every $f(X) \in \mathbb{F}_q[X]$ such that

$$Q\left(X, f(X), f\left(\gamma X\right), f\left(\gamma^2 X\right), \ldots, f\left(\gamma^{s-1} X\right)\right) \equiv 0 \tag{14.6}$$

DO

4: IF $\deg(f) \le k-1$ and $f(X)$ satisfies (14.2) for at least t values of i THEN

5: Add $f(X)$ to Ł.

6: RETURN Ł

Next we argue that the root finding step works.

Lemma 14.3.2. *If $t > \frac{D+k-1}{m-s+1}$, then every polynomial $f(X)$ that needs to be output satisfies (14.6).*

Proof. Consider the polynomial $R(X) = Q\left(X, f(X), f\left(\gamma X\right), ..., f\left(\gamma^{s-1} X\right)\right)$. Because the degree of $f\left(\gamma^{\ell} X\right)$ (for every $0 \le \ell \le s-1$) is at most $k-1$,

$$\deg(R) \le D + k - 1. \tag{14.7}$$

Let f(X) be one of the polynomials of degree at most $k-1$ that needs to be output, and $f(X)$ agrees with the received word at column i for some $0 \le i < N$, that is:

$$\begin{pmatrix} f\left(\gamma^{mi}\right) \\ f\left(\gamma^{mi+1}\right) \\ . \\ . \\ . \\ f\left(\gamma^{m(i+1)-1}\right) \end{pmatrix} = \begin{pmatrix} y_{mi} \\ y_{mi+1} \\ . \\ . \\ . \\ y_{m(i+1)-1} \end{pmatrix},$$

then for all $0 \leq j \leq m-s$, we have:

$$\begin{aligned} R\left(\gamma^{mi+j}\right) &= Q\left(\gamma^{mi+j}, f\left(\gamma^{mi+j}\right), f\left(\gamma^{mi+1+j}\right), ..., f\left(\gamma^{mi+s-1+j}\right)\right) \\ &= Q\left(\gamma^{mi+j}, y_{mi+j}, y_{mi+1+j}, ..., y_{mi+s-1+j}\right) = 0. \end{aligned}$$

In the above, the first equality follows as $f(X)$ agrees with $\mathbf{y}$ in column i while the second equality follows from (14.5). Thus, the number of roots of $R(X)$ is at least

$$t(m-s+1) > D+k-1 \geq \deg(R),$$

where the first inequality follows from the assumption in the claim and the second inequality follows from (14.7). Hence, by the degree mantra $R(X) \equiv 0$, which shows that $f(X)$ satisfies (14.6), as desired. □

14.3.1 Error Correction Capability

Now we analyze the the fraction of errors the algorithm above can handle. (We will come back to the thorny issue of proving a bound on the output list size for the root finding step in Section 14.3.2.)

The argument for the fraction of errors follows the by now standard route. To satisfy the constraint in Lemma 14.3.1, we pick

$$D = \left\lfloor \frac{N(m-s+1)-k+1}{s+1} \right\rfloor.$$

This along with the constraint in Lemma 14.3.2, implies that the algorithm works as long as

$$t > \left\lfloor \frac{D+k-1}{m-s+1} \right\rfloor.$$

The above is satisfied if we choose

$$t > \frac{\frac{N(m-s+1)-k+1}{s+1}+k-1}{m-s+1} = \frac{N(m-s+1)-k+1+(k-1)(s+1)}{(m-s+1)(s+1)} = \frac{N(m-s+1)+s(k-1)}{(s+1)(m-s+1)}.$$

Thus, we would be fine if we pick

$$t > \frac{N}{s+1} + \frac{s}{s+1} \cdot \frac{k}{m-s+1} = N\left(\frac{1}{s+1} + \left(\frac{s}{s+1}\right)\left(\frac{m}{m-s+1}\right) \cdot R\right),$$

where the equality follows from the fact that $k = mRN$. This implies the following result:

Theorem 14.3.3. *Algorithm 19 can list decode folded Reed-Solomon code with folding parameter $m \geq 1$ and rate R up to $\frac{s}{s+1}(1 - mR/(m-s+1))$ fraction of errors.*

See Figure 14.3.1 for an illustration of the bound above.

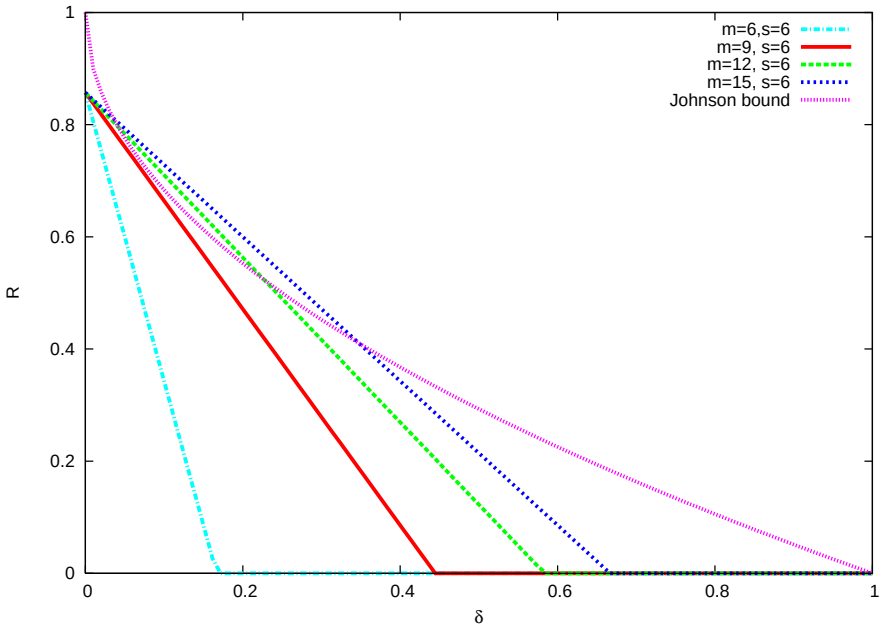


Figure 14.9: The tradeoff between rate R and the fraction of errors that can be corrected by Algorithm 19 for $s = 6$ and folding parameter $m = 6, 9, 12$ and 15. The Johnson bound is also plotted for comparison.

14.3.2 Bounding the Output List Size

We finally address the question of bounding the output list size in the root finding step of the algorithm. We will present a proof that will immediately lead to an algorithm to implement the root finding step. We will show that there are at most q^{s-1} possible solutions for the root finding step.

The main idea is the following: think of the coefficients of the output polynomial $f(X)$ as variables. Then the constraint (14.6) implies $D+k$ linear equations on these k variables. It turns out that if one picks only k out of these $D+k$ constraints, then the corresponding constraint matrix has rank at least $k-s+1$, which leads to the claimed bound. Finally, the claim on the rank of the constraint matrix follows by observing (and this is the crucial insight) that the constraint matrix is upper triangular. Further, the diagonal elements are evaluation of a non-zero polynomial of degree at most $s-1$ in k distinct elements. By the degree mantra (Proposition 5.2.3), this polynomial can have at most $s-1$ roots, which implies that at least $k-s+1$ elements of the

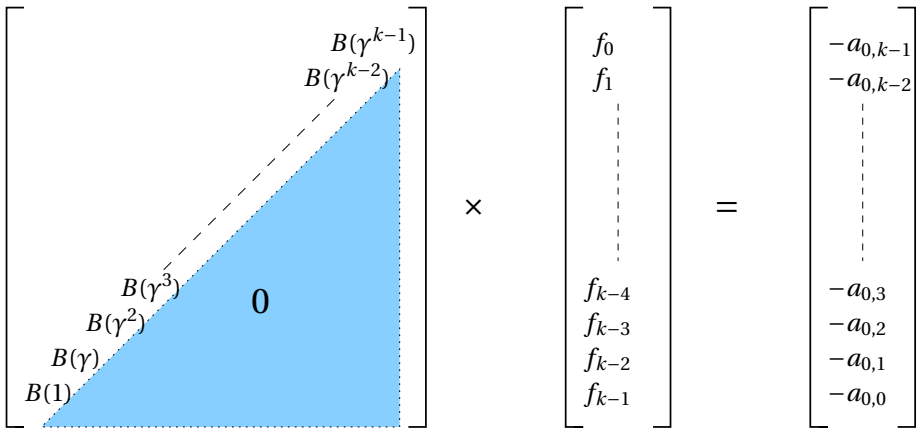


Figure 14.10: The system of linear equations with the variables $f_0, \ldots, f_{k-1}$ forming the coefficients of the polynomial $f(X) = \sum_{i=0}^{k-1} f_i X^i$ that we want to output. The constants $a_{j,0}$ are obtained from the interpolating polynomial from Step 1. $B(X)$ is a non-zero polynomial of degree at most $s-1$.

diagonal are non-zero, which then implies the claim. See Figure 14.10 for an illustration of the upper triangular system of linear equations.

Next, we present the argument above in full detail. (Note that the constraint on (14.8) is the same as the one in (14.6) because of the constraint on the structure of Q imposed by Step 1.)

Lemma 14.3.4. *There are at most q^{s-1} solutions to $f_0, f_1, .., f_{k-1}$ (where $f(X) = f_0 + f_1 X + ... + f_{k-1}X^{k-1}$) to the equations*

$$A_0(X) + A_1(X)f(X) + A_2(X)f\left(\gamma X\right) + ... + A_s(X)f\left(\gamma^{s-1}X\right) \equiv 0 \tag{14.8}$$

Proof. First we assume that X does not divide all of the polynomials $A_0, A_1, ..., A_s$. Then it implies that there exists $i^* > 0$ such that the constant term of the polynomial $A_{i^*}(X)$ is not zero. (Because otherwise, since $X|A_1(X), ..., A_s(X)$, by (14.8), we have X divides $A_0(X)$ and hence X divide all the $A_i(X)$ polynomials, which contradicts the assumption.)

To facilitate the proof, we define few auxiliary variables a_{ij} such that

$$A_i(X) = \sum_{j=0}^{D+k-1} a_{ij}X^j \text{ for every } 0 \le i \le s,$$

and define the following univariate polynomial:

$$B(X) = a_{1,0} + a_{2,0}X + a_{3,0}X^2 + ... + a_{s,0}X^{s-1}. \tag{14.9}$$

Notice that $a_{i^*,0} \neq 0$, so $B(X)$ is non-zero polynomial. And because degree of $B(X)$ is at most $s-1$, by the degree mantra (Proposition 5.2.3), $B(X)$ has at most $s-1$ roots. Next, we claim the following:

Claim 14.3.5. *For every $0 \le j \le k-1$:*

- *If $B(\gamma^j) \neq 0$, then f_j is uniquely determined by $f_{j-1}, f_{j-2}, \ldots, f_0$.*
- *If $B(\gamma^j) = 0$, then f_j is unconstrained, i.e. f_j can take any of the q values in $\mathbb{F}_q$.*

We defer the proof of the claim above for now. Suppose that the above claim is correct. Then as γ is a generator of $\mathbb{F}_q$, $1, \gamma, \gamma^2, \ldots, \gamma^{k-1}$ are distinct (since $k-1 \le q-2$). Further, by the degree mantra (Proposition 5.2.3) at most $s-1$ of these elements are roots of the polynomial $B(X)$. Therefore by Claim 14.3.5, the number of solutions to $f_0, f_1, \ldots, f_{k-1}$ is at most q^{s-1}. [2]

We are almost done except we need to remove our earlier assumption that X does not divide every A_i. Towards this end, we essentially just factor out the largest common power of X from all of the A_i's, and proceed with the reduced polynomial. Let $l \ge 0$ be the largest l such that $A_i(X) = X^l A_i'(X)$ for $0 \le i \le s$; then X does not divide all of $A_i'(X)$ and we have:

$$X^l \left(A_0'(X) + A_1'(X) f(X) + \cdots + A_s'(X) f(\gamma^{s-1} X) \right) \equiv 0.$$

Thus we can do the entire argument above by replacing $A_i(X)$ with $A_i'(X)$ since the above constraint implies that $A_i'(X)$'s also satisfy (14.8). □

Next we prove Claim 14.3.5.

Proof of Claim 14.3.5. Recall that we can assume that X does not divide all of $\{A_0(X), \ldots, A_s(X)\}$.

Let $C(X) = A_0(X) + A_1(X) f(X) + \cdots + A_s f\left(\gamma^{s-1} X\right)$. Recall that we have $C(X) \equiv 0$. If we expand out each polynomial multiplication, we have:

$$\begin{aligned}
C(X) =& a_{0,0} + a_{0,1} X + \cdots + a_{0,D+k-1} X^{D+k-1} \\
&+ \left(a_{1,0} + a_{1,1} X + \cdots + a_{1,D+k-1} X^{D+k-1} \right) \left(f_0 + f_1 X + f_2 X^2 + \cdots + f_{k-1} X^{k-1} \right) \\
&+ \left(a_{2,0} + a_{2,1} X + \cdots + a_{2,D+k-1} X^{D+k-1} \right) \left(f_0 + f_1 \gamma X + f_2 \gamma^2 X^2 + \cdots + f_{k-1} \gamma^{k-1} X^{k-1} \right) \\
&\vdots \\
&+ \left(a_{s,0} + a_{s,1} X + \cdots + a_{s,D+k-1} X^{D+k-1} \right) \left(f_0 + f_1 \gamma^{s-1} X + f_2 \gamma^{2(s-1)} X^2 + \cdots + f_{k-1} \gamma^{(k-1)(s-1)} X^{k-1} \right)
\end{aligned} \tag{14.10}$$

Now if we collect terms of the same degree, we will have a polynomial of the form:

$$C(X) = c_0 + c_1 X + c_2 X^2 + \cdots + c_{D+k-1} X^{D+k-1}.$$

[2] Build a "decision tree" with f_0 as the root and f_j in the jth level: each edge is labeled by the assigned value to the parent node variable. For any internal node in the jth level, if $B(\gamma^j) \neq 0$, then the node has a single child with the edge taking the unique value promised by Claim 14.3.5. Otherwise the node has q children with q different labels from $\mathbb{F}_q$. By Claim 14.3.5, the number of solutions to $f(X)$ is upper bounded by the number of nodes in the kth level in the decision tree, which by the fact that B has at most $s-1$ roots is upper bounded by q^{s-1}.

So we have $D+k$ linear equations in variables $f_0, \ldots, f_{k-1}$, and we are seeking those solutions such that $c_j = 0$ for every $0 \le j \le D+k-1$. We will only consider the $0 \le j \le k-1$ equations. We first look at the equation for $j = 0$: $c_0 = 0$. This implies the following equalities:

$$0 = a_{0,0} + f_0 a_{1,0} + f_0 a_{2,0} + \cdots + f_0 a_{s,0} \tag{14.11}$$

$$0 = a_{0,0} + f_0 \left(a_{1,0} + a_{2,0} + \cdots + a_{s,0}\right) \tag{14.12}$$

$$0 = a_{0,0} + f_0 B(1). \tag{14.13}$$

In the above (14.11) follows from (14.10), (14.12) follows by simple manipulation while (14.13) follows from the definition of $B(X)$ in (14.9).

Now, we have two possible cases:

- **Case 1:** $B(1) \neq 0$. In this case, (14.13) implies that $f_0 = \frac{-a_{0,0}}{B(1)}$. In particular, f_0 is fixed.
- **Case 2:** $B(1) = 0$. In this case f_0 has no constraint (and hence can take on any of the q values in $\mathbb{F}_q$).

Now consider the equation for $j = 1$: $c_1 = 0$. Using the same argument as we did for $j = 0$, we obtain the following sequence of equalities:

$$\begin{aligned}
0 &= a_{0,1} + f_1 a_{1,0} + f_0 a_{1,1} + f_1 a_{2,0}\gamma + f_0 a_{2,1} + \cdots + f_1 a_{s,0}\gamma^{s-1} + f_0 a_{s,1} \\
0 &= a_{0,1} + f_1 \left(a_{1,0} + a_{2,0}\gamma + \cdots + a_{s,0}\gamma^{s-1}\right) + f_0 \left(\sum_{l=1}^{s} a_{l,1}\right) \\
0 &= a_{0,1} + f_1 B(\gamma) + f_0 b_0^{(1)}
\end{aligned} \tag{14.14}$$

where $b_0^{(1)} = \sum_{l=1}^{s} a_{l,1}$ is a constant. We have two possible cases:

- **Case 1:** $B(\gamma) \neq 0$. In this case, by (14.14), we have $f_1 = \frac{-a_{0,1} - f_0 b_0^{(1)}}{B(\gamma)}$ and there is a unique choice for f_1 given fixed f_0.
- **Case 2:** $B(\gamma) = 0$. In this case, f_1 is unconstrained.

Now consider the case of arbitrary j: $c_j = 0$. Again using similar arguments as above, we get:

$$\begin{aligned}
0 = a_{0,j} &+ f_j (a_{1,0} + a_{2,0}\gamma^j + a_{3,0}\gamma^{2j} + \cdots + a_{s,0}\gamma^{j(s-1)}) \\
&+ f_{j-1} (a_{1,1} + a_{2,1}\gamma^{j-1} + a_{3,1}\gamma^{2(j-1)} + \cdots + a_{s,1}\gamma^{(j-1)(s-1)}) \\
&\qquad\qquad \vdots \\
&+ f_1 (a_{1,j-1} + a_{2,j-1}\gamma + a_{3,j-1}\gamma^2 + \cdots + a_{s,j-1}\gamma^{s-1}) \\
&+ f_0 (a_{1,j} + a_{2,j} + a_{3,j} + \cdots + a_{s,j}) \\
0 = a_{0,j} &+ f_j B(\gamma^j) + \sum_{l=0}^{j-1} f_l b_l^{(j)}
\end{aligned} \tag{14.15}$$

where $b_l^{(j)} = \sum_{\iota=1}^{s} a_{\iota,j-l} \cdot \gamma^{l(\iota-1)}$ are constants for $0 \le j \le k-1$.

We have two possible cases:

- **Case 1:** $B(\gamma^j) \neq 0$. In this case, by (14.15), we have

$$f_j = \frac{-a_{0,j} - \sum_{l=0}^{j-1} f_l b_l^{(j)}}{B(\gamma^j)} \tag{14.16}$$

  and there is a unique choice for f_j given fixed $f_0, \ldots, f_{j-1}$.

- **Case 2:** $B(\gamma^j) = 0$. In this case f_j is unconstrained.

This completes the proof. □

We now revisit the proof above and make some algorithmic observations. First, we note that to compute all the tuples $(f_0, \ldots, f_{k-1})$ that satisfy (14.8) one needs to solve the linear equations (14.15) for $j = 0, \ldots, k-1$. One can state this system of linear equation as (see also Figure 14.10)

$$C \cdot \begin{pmatrix} f_0 \\ \vdots \\ f_{k-1} \end{pmatrix} = \begin{pmatrix} -a_{0,k-1} \\ \vdots \\ -a_{0,0} \end{pmatrix},$$

where C is a $k \times k$ upper triangular matrix. Further each entry in C is either a 0 or $B(\gamma^j)$ or $b_l^{(j)}$– each of which can be computed in $O(s \log s)$ operations over $\mathbb{F}_q$. Thus, we can setup this system of equations in $O\left(s \log s k^2\right)$ operations over $\mathbb{F}_q$.

Next, we make the observation that all the solutions to (14.8) form an affine subspace. Let $0 \leq d \leq s-1$ denote the number of roots of $B(X)$ in $\{1, \gamma, \ldots, \gamma^{k-1}\}$. Then since there will be d unconstrained variables among $f_0, \ldots, f_{k-1}$ (one of every j such that $B(\gamma^j) = 0$), it is not too hard to see that all the solutions will be in the set $\left\{M \cdot \mathbf{x} + \mathbf{z} | \mathbf{x} \in \mathbb{F}_q^d\right\}$, for some $k \times d$ matrix M and some $\mathbf{z} \in \mathbb{F}_q^k$. Indeed every $\mathbf{x} \in \mathbb{F}_q^d$ corresponds to an assignment to the d unconstrained variables among $f_0, \ldots, f_j$. The matrix M and the vector $\mathbf{z}$ are determined by the equations in (14.16). Further, since C is upper triangular, both M and $\mathbf{z}$ can be computed with $O\left(k^2\right)$ operations over $\mathbb{F}_q$.

The discussion above implies the following:

Corollary 14.3.6. *The set of solutions to (14.8) are contained in an affine subspace* $\left\{M \cdot \mathbf{x} + \mathbf{z} | \mathbf{x} \in \mathbb{F}_q^d\right\}$ *for some* $0 \leq d \leq s-1$ *and* $M \in \mathbb{F}_q^{k \times d}$ *and* $\mathbf{z} \in \mathbb{F}_q^k$. *Further,* M *and* $\mathbf{z}$ *can be computed from the polynomials* $A_0(X), \ldots, A_s(X)$ *with* $O(s \log s k^2)$ *operations over* $\mathbb{F}_q$.

14.3.3 Algorithm Implementation and Runtime Analysis

In this sub-section, we discuss how both the interpolation and root finding steps of the algorithm can be implemented and analyze the run time of each step.

Step 1 involves solving Nm linear equation in $O(Nm)$ variables and can e.g. be solved by Gaussian elimination in $O((Nm)^3)$ operations over $\mathbb{F}_q$. This is similar to what we have seen for Algorithms 14, 15 and 16. However, the fact that the interpolation polynomial has total degree

of one in the variables $Y_1, \ldots, Y_s$ implies a much faster algorithm. In particular, one can perform the interpolation in $O(Nm\log^2(Nm)\log\log(Nm))$ operations over $\mathbb{F}_q$.

The root finding step involves computing all the "roots" of Q. The proof of Lemma 14.3.4 actually suggests Algorithm 20.

Algorithm 20 The Root Finding Algorithm for Algorithm 19

INPUT: $A_0(X), \ldots, A_s(X)$

OUTPUT: All polynomials $f(X)$ of degree at most $k-1$ that satisfy (14.8)

1: Compute ℓ such that X^ℓ is the largest common power of X among $A_0(X), \ldots, A_s(X)$.
2: FOR every $0 \leq i \leq s$ DO
3: $\quad A_i(X) \leftarrow \frac{A_i(X)}{X^\ell}$.
4: Compute $B(X)$ according to (14.9)
5: Compute d, $\mathbf{z}$ and M such that the solutions to the k linear system of equations in (14.15) lie in the set $\left\{M \cdot \mathbf{x} + \mathbf{z} | \mathbf{x} \in \mathbb{F}_q^d\right\}$.
6: Ł $\leftarrow \emptyset$
7: FOR every $\mathbf{x} \in \mathbb{F}_q^d$ DO
8: $\quad (f_0, \ldots, f_{k-1}) \leftarrow M \cdot \mathbf{x} + \mathbf{z}$.
9: $\quad f(X) \leftarrow \sum_{i=0}^{k-1} f_i \cdot X^i$.
10: $\quad$ IF $f(X)$ satisfies (14.8) THEN
11: $\quad\quad$ Add $f(X)$ to Ł.
12: RETURN Ł

Next, we analyze the run time of the algorithm. Throughout, we will assume that all polynomials are represented in their standard coefficient form.

Step 1 just involves figuring out the smallest power of X in each $A_i(X)$ that has a non-zero coefficient from which one can compute the value of ℓ. This can be done with $O(D + k + s(D + 1)) = O(Nm)$ operations over $\mathbb{F}_q$. Further, given the value of ℓ one just needs to "shift" all the coefficients in each of the $A_i(X)$'s to the right by ℓ, which again can be done with $O(Nm)$ operations over $\mathbb{F}_q$.

Now we move to the root finding step. The run time actually depends on what it means to "solve" the linear system. If one is happy with a succinct description of a set of possible solution that contains the actual output then one can halt Algorithm 20 after Step 5 and Corollary 14.3.6 implies that this step can be implemented in $O\left(s\log sk^2\right) = O\left(s\log s(Nm)^2\right)$ operations over $\mathbb{F}_q$. However, if one wants the actual set of polynomials that need to be output, then the only known option so far is to check all the q^{s-1} potential solutions as in Steps 7-11. (However, we'll see a twist in Section 14.4.) The latter would imply a total of $O(s\log s(Nm)^2) + O(q^{s-1} \cdot (Nm)^2)$ operations over $\mathbb{F}_q$.

Thus, we have the following result:

Lemma 14.3.7. *With $O(s\log s(Nm)^2)$ operations over $\mathbb{F}_q$, the algorithm above can return an affine subspace of dimension $s-1$ that contains all the polynomials of degree at most $k-1$*

that need to be output. Further, the exact set of solution can be computed in with additional $O(q^{s-1} \cdot (Nm)^2)$ *operations over* $\mathbb{F}_q$.

14.3.4 Wrapping Up

By Theorem 14.3.3, we know that we can list decode a folded Reed-Solomon code with folding parameter $m \geq 1$ up to

$$\frac{s}{s+1} \cdot \left(1 - \frac{m}{m-s+1} \cdot R\right) \tag{14.17}$$

fraction of errors for any $1 \leq s \leq m$.

To obtain our desired bound $1 - R - \varepsilon$ fraction of errors, we instantiate the parameter s and m such that

$$\frac{s}{s+1} \geq 1 - \varepsilon \text{ and } \frac{m}{m-s+1} \leq 1 + \varepsilon. \tag{14.18}$$

It is easy to check that one can choose

$$s = \Theta(1/\varepsilon) \text{ and } m = \Theta(1/\varepsilon^2)$$

such that the bounds in (14.18) are satisfied. Using the bounds from (14.18) in (14.17) implies that the algorithm can handle at least

$$(1-\varepsilon)(1-(1+\varepsilon)R) = 1 - \varepsilon - R + \varepsilon^2 R > 1 - R - \varepsilon$$

fraction of errors, as desired.

We are almost done since Lemma 14.3.7 shows that the run time of the algorithm is $q^{O(s)}$. The only thing we need to choose is q: for the final result we pick q to be the smallest power of 2 that is larger than $Nm+1$. Then the discussion above along with Lemma 14.3.7 implies the following result (the claim on strong explicitness follows from the fact that Reed-Solomon codes are strongly explicit):

Theorem 14.3.8. *There exist* strongly explicit *folded Reed-Solomon codes of rate* R *that for large enough block length* N *can be list decoded from* $1 - R - \varepsilon$ *fraction of errors (for any small enough* $\varepsilon > 0$*) in time* $\left(\frac{N}{\varepsilon}\right)^{O(1/\varepsilon)}$. *The worst-case list size is* $\left(\frac{N}{\varepsilon}\right)^{O(1/\varepsilon)}$ *and the alphabet size is* $\left(\frac{N}{\varepsilon}\right)^{O(1/\varepsilon^2)}$.

14.4 Bibliographic Notes and Discussion

There was no improvement to the Guruswami-Sudan result (Theorem 13.2.6) for about seven years till Parvaresh and Vardy showed that "Correlated" Reed-Solomon codes can be list-decoded up to $1 - (mR)^{\frac{1}{m+1}}$ fraction of errors for $m \geq 1$ [58]. Note that for $m = 1$, correlated Reed-Solomon codes are equivalent to Reed-Solomon codes and the result of Parvaresh and Vardy recovers Theorem 13.2.6. Immediately, after that Guruswami and Rudra [33] showed that Folded Reed-Solomon codes can achieve the list-decoding capacity of $1 - R - \varepsilon$ and hence, answer

Question 14.0.1 in the affirmative. Guruswami [28] reproved this result but with a much simpler proof. In this chapter, we studied the proof due to Guruswami. Guruswami in [28] credits Salil Vadhan for the the interpolation step. An algorithm presented in Brander's thesis [4] shows that for the special interpolation in Algorithm 19, one can perform the interpolation in $O(Nm\log^2(Nm)\log\log(Nm))$ operations over $\mathbb{F}_q$. The idea of using the "sliding window" for list decoding Folded Reed-Solomon codes is originally due to Guruswami and Rudra [32].

The bound of q^{s-1} on the list size for folded Reed-Solomon codes was first proven in [32] by roughly the following argument. One reduced the problem of finding roots to finding roots of a *univariate* polynomial related to Q over $\mathbb{F}_{q^k}$. (Note that each polynomial in $\mathbb{F}_q[X]$ of degree at most $k-1$ has a one to one correspondence with elements of $\mathbb{F}_{q^k}$– see e.g. Theorem 11.2.1.) The list size bound follows from the fact that this new univariate polynomial had degree q^{s-1}. Thus, implementing the algorithm entails running a root finding algorithm over a big extension field, which in practice has terrible performance.

Discussion. For constant ε, Theorem 14.3.8 answers Question 14.0.1 in the affirmative. However, from a practical point of view, there are three issues with the result: alphabet, list size and run time. Below we tackle each of these issues.

Large Alphabet. Recall that one only needs an alphabet of size $2^{O(1/\varepsilon)}$ to be able to list decode from $1-R-\varepsilon$ fraction of errors, which is independent of N. It turns out that combining Theorem 14.3.8 along with code concatenation and expanders allows us to construct codes over alphabets of size roughly $2^{O(1/\varepsilon^4)}$ [32]. (The idea of using expanders and code concatenation was not new to [32]: the connection was exploited in earlier work by Guruswami and Indyk [31].)

The above however, does not answer the question of achieving list decoding capacity for *fixed* q, say e.g. $q=2$. We know that there exists binary code of rate R that are $(H^{-1}(1-R-\varepsilon), O(1/\varepsilon))$-list decodable codes (see Theorem 7.4.1). The best known explicit codes with efficient list decoding algorithms are those achieved by concatenating folded Reed-Solomon codes with suitable inner codes achieve the so called *Blokh-Zyablov* bound [34]. However, the tradeoff is far from the list decoding capacity. As one sample point, consider the case when we want to list decode from $\frac{1}{2}-\varepsilon$ fraction of errors. Then the result of [34] gives codes of rate $\Theta(\varepsilon^3)$ while the codes on list decoding capacity has rate $\Omega(\varepsilon^2)$. The following fundamental question is still very much wide open:

Open Question 14.4.1. *Do there exist explicit binary codes with rate R that can be list decoded from $H^{-1}(1-R-\varepsilon)$ fraction of errors with polynomial list decoding algorithms?*

The above question is open even if we drop the requirement on efficient list decoding algorithm or we only ask for a code that can list decode from $1/2-\varepsilon$ fraction of errors with rate $\Omega(\varepsilon^a)$ for some $a<3$. It is known (combinatorially) that concatenated codes can achieve the list decoding capacity but the result is via a souped up random coding argument and does not give much information about an efficient decoding algorithm [35].

List Size. It is natural to wonder if the bound on the list size in Lemma 14.3.4 above can be improved as that would show that folded Reed-Solomon codes can be list decoded up to the list decoding capacity but with a smaller output list size than Theorem 14.3.8. Guruswami showed that in its full generality the bound cannot be improved [28]. In particular, he exhibits explicit polynomials $A_0(X), \ldots, A_s(X)$ such that there are at least q^{s-2} solutions for $f(X)$ that satisfy (14.8). However, these $A_i(X)$'s are not known to be the output for an actual interpolation instance. In other words, the following question is still open:

Open Question 14.4.2. *Can folded Reed-Solomon codes of rate R be list decoded from $1-R-\varepsilon$ fraction of errors with list size $f(1/\varepsilon)\cdot N^c$ for some increasing function $f(\cdot)$ and absolute constant c?*

Even the question above with $N^{(1/\varepsilon)^{o(1)}}$ is still open.

However, if one is willing to consider codes other than folded Reed-Solomon codes in order to answer to achieve list decoding capacity with smaller list size (perhaps with one only dependent on ε), then there is good news. Guruswami in the same paper that presented the algorithm in this chapter also present a *randomized* construction of codes of rate R that are $(1-R-\varepsilon, O(1/\varepsilon^2))$-list decodable codes [28]. This is of course worse than what we know from the probabilistic method. However, the good thing about the construction of Guruswami comes with an $O(N/\varepsilon)^{O(1/\varepsilon)}$-list decoding algorithm.

Next we briefly mention the key ingredient in the result above. To see the potential for improvement consider Corollary 14.3.6. The main observation is that all the potential solutions lie in an affine subspace of dimension $s-1$. The key idea in [28] was use the folded Reed-Solomon encoding for a special subset of the message space $\mathbb{F}_q^k$. Call a subspace $S \subseteq \mathbb{F}_q^k$ to be a (q,k,ε,ℓ,L)-*subspace evasive subset* if

1. $|S| \geq q^{k(1-\varepsilon)}$; and
2. For any (affine) subspace $T \subseteq \mathbb{F}_q^k$ of dimension ℓ, we have $|S \cap T| \leq L$.

The code in [28], applies the folded Reed-Solomon encoding on a $\left(q,k,s,O\left(s^2\right)\right)$-subspace evasive subset (such a subset can be shown to exist via the probabilistic method). The reason why this sub-code of folded Reed-Solomon code works is as follows: Condition (1) ensures that the new code has rate at least $R(1-\varepsilon)$, where R is the rate of the original folded Reed-Solomon code and condition (2) ensures that the number of output polynomial in the root finding step of the algorithm we considered in the last section is at most L. (This is because by Corollary 14.3.6 the output message space is an affine subspace of dimension $s-1$ in $\mathbb{F}_Q^k$. However, in the new code by condition 2, there can be at most $O\left(s^2\right)$ output solutions.)

The result above however, has two shortcomings: (i) the code is no longer explicit and (ii) even though the worst case list size is $O\left(\frac{1}{\varepsilon^2}\right)$, it was not know how to obtain this output without listing all the q^{s-1} possibilities and pruning them against S. The latter meant that the decoding runtime did not improve over the one achieved in Theorem 14.3.8.

Large Runtime. We finally address the question of the high run time of all the list decoding algorithms so far. Dvir and Lovett [16], presented a construction of an *explicit* $(q,k,\varepsilon,s,s^{O(s)})$-subspace evasive subset S^*. More interestingly, given any affine subspace T of dimension at most s, it can compute $S \cap T$ in time proportional to the output size. Thus, this result along with the discussion above implies the following result:

Theorem 14.4.1. *There exist* strongly explicit *codes of rate R that for large enough block length N can be list decoded from $1-R-\varepsilon$ fraction of errors (for any small enough $\varepsilon > 0$) in time $O\left(\left(\frac{N}{\varepsilon^2}\right)^2\right) + \left(\frac{1}{\varepsilon}\right)^{O(1/\varepsilon)}$. The worst-case list size is $\left(\frac{1}{\varepsilon}\right)^{O(1/\varepsilon)}$ and the alphabet size is $\left(\frac{N}{\varepsilon}\right)^{O(1/\varepsilon^2)}$.*

The above answers Question 14.0.1 pretty satisfactorily. However, to obtain a completely satisfactory answer one would have to solve the following open question:

Open Question 14.4.3. *Are there explicit codes of rate $R > 0$ that are $\left(1-R-\varepsilon, (1/\varepsilon)^{O(1)}\right)$-list decodable that can be list-decoded in time* $\mathrm{poly}(N, 1/\varepsilon)$ *over alphabet of size $q \leq poly(n)$?*

The above question, without the requirement of explicitness, has been answered by Guruswami and Xing [38].

Part V

The Applications

Chapter 15

Cutting Data Down to Size: Hashing

In this chapter, we will study hashing, which is a method to compute a small digest of data that can be used as a surrogate to later perform quick checks on the data. We begin with brief descriptions of three practical applications where hashing is useful. We then formally state the definition of hash functions that are needed in these applications (the so called “universal" hash functions). Next, we will show how in some sense good hashing functions and good codes are equivalent. Finally, we will see how hashing can solve a problem motivated by outsourced storage in the “cloud."

15.1 Why Should You Care About Hashing?

Hashing is one of the most widely used objects in computer science. In this section, we outline three practical applications that heavily use hashing. While describing the applications, we will also highlight the properties of hash functions that these applications need.

Before we delve into the applications, let us first formally define a hash function.

Definition 15.1.1 (Hash Function)**.** Given a domain $\mathbb{D}$ and a range Σ, (typically, with $|\Sigma| < |\mathbb{D}|$), a hash function is a map

$$h : \mathbb{D} \to \Sigma.$$

Of course, the definition above is too general and we will later specify properties that will make the definition more interesting.

Integrity Checks on Routers. Routers on the Internet process a lot of packets in a very small amount of time. Among other tasks, router has to perform an “integrity check" on the packet to make sure that the packet it is processing is not corrupted. Since the packet has well defined fields, the router could check if all the field have valid entries. However, it is possible that one of the valid entry could be turned into another valid entry. However, the packet as a whole could still be invalid.

If you have progressed so far in the book, you will recognize that the above is the error detection problem and we know how to do error detection (see e.g., Proposition 2.3.3). However, the

algorithms that we have seen in this book are too slow to implement in routers. Hence, Internet protocols use a hash function on a domain $\mathbb{D}$ that encodes all the information that needs to go into a packet. Thus, given an $\mathbf{x} \in \mathbb{D}$, the packet is the pair $(\mathbf{x}, h(\mathbf{x}))$. The sender sends the packet $(\mathbf{x}, h(\mathbf{x}))$ and the receiver gets $(\mathbf{x}', y)$. In order to check if any errors occurred during transmission, the receiver checks if $h(\mathbf{x}') = y$. If the check fails, the receiver asks for a re-transmission otherwise it assumes there were no errors during transmission. There are two requirements from the hash function: (i) It should be super efficient to compute $h(\mathbf{x})$ given $\mathbf{x}$ and (ii) h should avoid "collisions," i.e. if $\mathbf{x} \neq \mathbf{x}'$, then $h(\mathbf{x}) \neq h(\mathbf{x}')$.[1]

Integrity Checks in Cloud Storage. Say, you (as a client) have data $\mathbf{x} \in \mathbb{D}$ that you want to outsource $\mathbf{x}$ to a cloud storage provider. Of course once you "ship" off $\mathbf{x}$ to the cloud, you do not want to store it locally. However, you do not quite trust the cloud either. If you do not audit the cloud storage server in any way, then nothing stops the storage provider from throwing away $\mathbf{x}$ and send you some other data $\mathbf{x}'$ when you ask for $\mathbf{x}$. The problem of designing an auditing protocol that can verify whether the server has the data $\mathbf{x}$ is called the *data possession* problem.

We consider two scenarios. In the first scenario, you access the data pretty frequently during "normal" operation. In such cases, here is a simple check you can perform. When you ship off $\mathbf{x}$ to the cloud, compute $z = h(\mathbf{x})$ and store it. Later when you access $\mathbf{x}$ and the storage provider send you $\mathbf{x}'$, you compute $h(\mathbf{x}')$ and check if it is the same as the stored $h(\mathbf{x})$. This is exactly the same solution as the one for packet verification mentioned above.

Now consider the scenario, where the cloud is used as an archival storage. In such a case, one needs an "auditing" process to ensure that the server is indeed storing $\mathbf{x}$ (or is storing some massaged version from which it can compute $\mathbf{x}$– e.g. the storage provider can compress $\mathbf{x}$). One can always ask the storage provider to send back $\mathbf{x}$ and then use the scheme above. However, if $\mathbf{x}$ is meant to be archived in the cloud, it would be better to resolve the following question:

Question 15.1.1. *Is there an auditing protocol with small client-server communication*[a]*, which if the server passes then the client should be able to certain (with some high confidence) that the server is indeed storing* $\mathbf{x}$*?*

[a] In particular, we rule out solutions where the server sends $\mathbf{x}$ to the client.

We will see later how this problem can be solved using hashing.

Fast Table Lookup. One of the most common operations on databases is the following. Assume there is a table with entries from $\mathbb{D}$. One would like to decide on a data structure to store

[1] Note that in the above example, one could have $\mathbf{x} \neq \mathbf{x}'$ and $h(\mathbf{x}) \neq h(\mathbf{x}')$ but it is still possible that $y = h(\mathbf{x}')$ and hence the corrupted packet $(\mathbf{x}', y)$ would pass the check above. Our understanding is that such occurrences are rare.

the table so that later on given an element $\mathbf{x} \in \mathbb{D}$, one would quickly like to decide whether $\mathbf{x}$ is in the table or now.

Let us formalize the problem a bit: assume that the table needs to store N values $a_1, \ldots, a_N \in \mathbb{D}$. Then later given $\mathbf{x} \in \mathbb{D}$ one needs to decide if $\mathbf{x} = a_i$ for some i. Here is one simple solution: sort the n elements in an array T and given $\mathbf{x} \in \mathbb{D}$ use binary search to check if $\mathbf{x}$ is in T or not. This solution uses $\Theta(N)$ amounts of storage and searching for $\mathbf{x}$ takes $\Theta(\log N)$ time. Further, the pre-processing time (i.e. time taken to build the array T) is $\Theta(N \log N)$. The space usage of this scheme is of course optimal but one would like the lookup to be faster: ideally we should be able to perform the search in $O(1)$ time. Also it would be nice to get the pre-processing time closer to the optimal $O(N)$. Further, this scheme is very bad for dynamic data: inserting an item to and deleting an item from T takes $\Theta(N)$ time in the worst-case.

Now consider the following solution: build a boolean array B with one entry for each $z \in \mathbb{D}$ and set $B[a_i] = 1$ for every $i \in [N]$ (and every other entry is 0).[2] Then searching for $\mathbf{x}$ is easy: just lookup $B[\mathbf{x}]$ and check if $B[\mathbf{x}] \neq 0$. Further, this data structure can easily handle addition and deletion of elements (by incrementing and decrementing the corresponding entry of B respectively). However, the amount of storage and pre-processing time are both $\Theta(|\mathbb{D}|)$, which can be much much bigger than the optimal $O(N)$. This is definitely true for tables stored in real life databases. This leads to the following question:

Question 15.1.2. *Is there a data structure that supports searching, insertion and deletion in $O(1)$ time but only needs $O(N)$ space and $O(N)$ pre-processing time?*

We will see later how to solve this problem with hashing.

15.2 Avoiding Hash Collisions

One of the properties that we needed in the applications outlined in the previous section was that the hash function $h : \mathbb{D} \to \Sigma$ should avoid collisions. That is, given $\mathbf{x} \neq y \in \mathbb{D}$, we want $h(\mathbf{x}) \neq h(y)$. However, since we have assumed that $|\Sigma| < |\mathbb{D}|$, this is clearly impossible. A simple counting argument shows that there will exist an $\mathbf{x} \neq y \in \mathbb{D}$ such that $h(\mathbf{x}) = h(y)$. There are two ways to overcome this hurdle.

The first is to define a cryptographic collision resistant hash function h, i.e. even though there exists collisions for the hash function h, it is computationally hard for an adversary to compute $\mathbf{x} \neq y$ such that $h(\mathbf{x}) = h(y)$.[3] This approach is out of the scope of this book and hence, we will not pursue this solution.

[2] If one wants to handle duplicates, one could store the number of occurrences of y in $B[y]$.

[3] This is a very informal definition. Typically, an adversary is modeled as a randomized polynomial time algorithm and there are different variants on whether the adversary is given $h(\mathbf{x})$ or $\mathbf{x}$ (or both). Also there are variants where one assumes a distribution on $\mathbf{x}$. Finally, there are no unconditionally collision resistant hash function but there exists provably collision resistant hash function under standard cryptographic assumptions: e.g. factoring is hard.

The second workaround is to define a family of hash functions and then argue that the probability of collision is small for a hash function chosen randomly from the family. More formally, we define a hash family:

Definition 15.2.1 (Hash Family)**.** Given $\mathbb{D}, \Sigma$ and an integer $m \geq 1$, a hash family $\mathscr{H}$ is a set $\{h_1, \ldots, h_m\}$ such that for each $i \in [m]$,

$$h_i : \mathbb{D} \to \Sigma.$$

Next we define the notion of (almost) universal hash function (family).

Definition 15.2.2 (Almost Universal Hash Family)**.** A hash family $\mathscr{H} = \{h_1, \ldots, h_m\}$ defined over the domain $\mathbb{D}$ and range Σ is said to be ε-almost universal hash function (family) for some $0 < \varepsilon \leq 1$ if for every $\mathbf{x} \neq y \in \mathbb{D}$,

$$\Pr_i \left[h_i(\mathbf{x}) = h_i(y) \right] \leq \varepsilon,$$

where in the above i is chosen uniformly at random from $[m]$.

We will show in the next section that ε-almost universal hash functions are equivalent to codes with (large enough) distance. In the rest of the section, we outline how these hash families provides satisfactory solutions to the problems considered in the previous section.

Integrity Checks. For the integrity check problem, one pick random $i \in [m]$ and chooses $h_i \in \mathscr{H}$, where $\mathscr{H}$ is an ε-almost universal hash function. Thus, for any $\mathbf{x} \neq y$, we're guaranteed with probability at least $1 - \varepsilon$ (over the random choice of i) that $h_i(\mathbf{x}) \neq h_i(y)$. Thus, this gives a randomized solution to the integrity checking problem in routers and cloud storage (where we consider the first scenario in which the cloud is asked to return the original data in its entirety).

It is not clear whether such hash functions can present a protocol that answers Question 15.1.1. There is a very natural protocol to consider though. When the client ships off data $\mathbf{x}$ to the cloud, it picks a random hash function $h_i \in \mathscr{H}$, where again $\mathscr{H}$ is an ε-universal hash function, and computes $h_i(\mathbf{x})$. Then it stores $h_i(\mathbf{x})$ and ships off $\mathbf{x}$ to the cloud. Later on, when the client wants to audit, it asks the cloud to send $h_i(\mathbf{x})$ back to it. Then if the cloud returns with z, the client checks if $z = h_i(\mathbf{x})$. If so, it assumes that the storage provider is indeed storing $\mathbf{x}$ and otherwise it concludes that the cloud is not storing $\mathbf{x}$.

Note that it is crucial that the hash function be chosen randomly: if the client picks a deterministic hash function h, then the cloud can store $h(\mathbf{x})$ and throw away $\mathbf{x}$ because it knows that the client is only going to ask for $h(\mathbf{x})$. Intuitively, the above protocol might work since the random index $i \in [m]$ is not known to the cloud till the client asks for $h_i(\mathbf{x})$, it seems "unlikely" that the cloud can compute $h_i(\mathbf{x})$ without storing $\mathbf{x}$. We will see later how the coding view of almost universal hash functions can make this intuition rigorous.

Fast Table Lookup. We now return to Question 15.1.2. The basic idea is simple: we will modify the earlier solution that maintained an entry for each element in the domain $\mathbb{D}$. The new solution will be to keep an entry for all possible hash values (instead of all entries in $\mathbb{D}$).

More formally, let $\mathcal{H} = \{h_1, \ldots, h_m\}$ be an ε-almost hash family with domain $\mathbb{D}$ and range Σ. Next we build an array of link list with one entry in the array for each value $v \in \Sigma$. We pick a random hash function $h_i \in \mathcal{H}$. Then for each a_j ($j \in [N]$) we add it to the link list corresponding to $h_i(a_j)$. Now to determine whether $\mathbf{x} = a_j$ for some j, we scan the link list corresponding to $h_i(\mathbf{x})$ and check if $\mathbf{x}$ is in the list or not. Before we analyze the space and time complexity of this data structure, we point out that insertion and deletion are fairly easy. For inserting an element $\mathbf{x}$, we compute $h_i(\mathbf{x})$ and add $\mathbf{x}$ to the link list corresponding to $h_i(\mathbf{x})$. For deletion, we first perform the search algorithm and then remove $\mathbf{x}$ from the list corresponding to $h_i(\mathbf{x})$, if it is present. It is easy to check that the algorithms are correct.

Next we analyze the space complexity. Note that for a table with N elements, we will use up $O(N)$ space in the linked lists and the array is of size $O(|\Sigma|)$. That is, the total space usage is $O(N + |\Sigma|)$. Thus, if we can pick $|\Sigma| = O(N)$, then we would match the optimal $O(N)$ bound for space.

Now we analyze the time complexity of the various operations. We first note that insertion is $O(1)$ time (assuming computing the hash value takes $O(1)$ time). Note that this also implies that the pre-processing time is $O(N + |\Sigma|)$, which matches the optimal $O(N)$ bound for $|\Sigma| \leq O(N)$. Second, for deletion, the time taken after performing the search algorithm is $O(1)$, assuming the lists as stored as doubly linked lists. (Recall that deleting an item from a doubly linked list if one has a pointer to the entry can be done in $O(1)$ time.)

Finally, we consider the search algorithm. Again assuming that computing a hash value takes $O(1)$ time, the time complexity of the algorithm to search for $\mathbf{x}$ is dominated by size of the list corresponding to $h_i(\mathbf{x})$. In other words, the time complexity is determined by the number of a_j that collide with $\mathbf{x}$, i.e., $h_i(\mathbf{x}) = h_i(a_j)$. We bound this size by the following simple observation.

Claim 15.2.1. *Let $\mathcal{H} = \{h_1, \ldots, h_m\}$ with domain $\mathbb{D}$ and range Σ be an ε-almost universal hash family. Then the following is true for any (distinct) $\mathbf{x}, a_1, a_2, \ldots, a_N \in \mathbb{D}$:*

$$\mathbb{E}_i \left[|\{a_j | h_i(\mathbf{x}) = h_i(a_j)\}| \right] \leq \varepsilon \cdot N,$$

where the expectation is taken over a uniformly random choice of $i \in [m]$.

Proof. Fix a $j \in [N]$. Then by definition of an ε-almost universal hash family, we have that

$$\Pr_i[h_i(\mathbf{x}) = h_i(a_j)] \leq \varepsilon.$$

Note that we want to bound $\mathbb{E}\left[\sum_{j=1}^{N} \mathbb{1}_{h_i(a_j)=h_i(\mathbf{x})}\right]$. The probability bound above along with the linearity of expectation (Proposition 3.1.2) and Lemma 3.1.1 completes the proof. □

The above discussion then implies the following:

Proposition 15.2.2. *Given an $O\left(\frac{1}{N}\right)$-almost universal hash family with domain $\mathbb{D}$ and range Σ such that $|\Sigma| = O(N)$, there exists a randomized data structure that given N elements $a_1, \ldots, a_N \in \mathbb{D}$, supports searching, insertion and deletion in expected $O(1)$ time while using $O(N)$ space in the worst-case.*

Thus, Proposition 15.2.2 answers Question 15.1.2 in the affirmative if we can answer the following question in the affirmative:

Question 15.2.1. *Given a domain $\mathbb{D}$ and an integer $N \geq 1$, does there exist an $O\left(\frac{1}{N}\right)$-almost universal hash function with domain $\mathbb{D}$ and a range Σ such that $|\Sigma| = O(N)$?*

We will answer the question above (spoiler alert!) in the affirmative in the next section.

15.3 Almost Universal Hash Function Families and Codes

In this section, we will present a very strong connection between almost universal hash families and codes with good distance: in fact, we will show that they are in fact *equivalent*.

We first begin by noting that any hash family has a very natural code associated with it and that every code has a very natural hash family associated with it.

Definition 15.3.1. Given a hash family $\mathcal{H} = \{h_1, \ldots, h_n\}$ where for each $i \in [n]$, $h_i : \mathbb{D} \to \Sigma$, consider the following associated code

$$C_{\mathcal{H}} : \mathbb{D} \to \Sigma^n,$$

where for any $\mathbf{x} \in \mathbb{D}$, we have

$$C_{\mathcal{H}}(\mathbf{x}) = (h_1(\mathbf{x}), h_2(\mathbf{x}), \ldots, h_n(\mathbf{x})).$$

The connection also goes the other way. That is, given an $(n,k)_\Sigma$ code C, we call the associated hash family $\mathcal{H}_C = \{h_1, \ldots, h_n)$, where for every $i \in [n]$,

$$h_i : \Sigma^k \to \Sigma$$

such that for every $\mathbf{x} \in \Sigma^k$ and $i \in [n]$,

$$h_i(\mathbf{x}) = C(\mathbf{x})_i.$$

Next we show that an ε-almost universal hash family is equivalent to a code with good distance.

Proposition 15.3.1. *Let $\mathcal{H} = \{h_1, \ldots, h_n\}$ be an ε-almost universal hash function, then the code $C_{\mathcal{H}}$ has distance at least $(1-\varepsilon)n$. On the other hand if C is an $(n,k,\delta n)$-code, then $\mathcal{H}_C$ is a $(1-\delta)$-almost universal hash function.*

Proof. We will only prove the first claim. The proof of the second claim is essentially identical and is left as an exercise.

Let $\mathcal{H} = \{h_1, \ldots, h_n\}$ be an ε-almost universal hash function. Now fix arbitrary $\mathbf{x} \neq \mathbf{y} \in \mathbb{D}$. Then by definition of $C_{\mathcal{H}}$, we have

$$\{i | h_i(\mathbf{x}) = h_i(\mathbf{y})\} = \{i | C_{\mathcal{H}}(\mathbf{x})_i = C_{\mathcal{H}}(\mathbf{y})_i\}.$$

This implies that

$$\Pr_i\left[h_i(\mathbf{x}) = h_i(\mathbf{y})\right] = \frac{|\{i|h_i(\mathbf{x}) = h_i(\mathbf{y})\}|}{n} = \frac{n - \Delta(C_{\mathscr{H}}(\mathbf{x}), C_{\mathscr{H}}(\mathbf{y}))}{n} = 1 - \frac{\Delta(C_{\mathscr{H}}(\mathbf{x}), C_{\mathscr{H}}(\mathbf{y}))}{n},$$

where the second equality follows from the definition of the Hamming distance. By the definition of ε-almost universal hash family the above probability is upper bounded by ε, which implies that

$$\Delta(C_{\mathscr{H}}(\mathbf{x}), C_{\mathscr{H}}(\mathbf{y})) \geq n(1-\varepsilon).$$

Since the choice of $\mathbf{x}$ and $\mathbf{y}$ was arbitrary, this implies that $C_{\mathscr{H}}$ has distance at least $n(1-\varepsilon)$ as desired. □

15.3.1 The Polynomial Hash Function

We now consider the hash family corresponding to a Reed-Solomon code. In particular, let C be a $[q, k, q-k+1]_q$ Reed-Solomon code. By Proposition 15.3.1, the hash family $\mathscr{H}_C$ is an $\frac{k-1}{q}$-almost universal hash family– this hash family in the literature is called the polynomial hash. Thus, if we pick q to be the smallest power of 2 larger than N and pick $k = O(1)$, then this leads to an $O(1/N)$-universal hash family that satisfies all the required properties in Question 15.2.1.

Note that the above implies that $|\mathbb{D}| = N^{O(1)}$. One might wonder if we can get an $O(1/N)$-almost universal hash family with the domain size being $N^{\omega(1)}$. We leave the resolution of this question as an exercise.

15.4 Data Possession Problem

In this section, we return to Question 15.1.1. Next we formalize the protocol for the data possession problem that we outlined in Section 15.2. Algorithm 21 presents the pre-processing step.

Algorithm 21 Pre-Processing for Data Possession Verification

INPUT: Data $\mathbf{x} \in \mathbb{D}$, hash family $\mathscr{H} = \{h_1, \ldots, h_m\}$ over domain $\mathbb{D}$

1: Client computes an index i for $\mathbf{x}$.
2: Client picks a random $j \in [m]$.
3: Client computes $z \leftarrow h_j(\mathbf{x})$ and stores (i, j, z).
4: Client sends $\mathbf{x}$ to the server.

Algorithm 22 formally states the verification protocol. Note that if the server has stored $\mathbf{x}$ (or is able to re-compute $\mathbf{x}$ from what it had stored), then it can pass the protocol by returning $a \leftarrow h_j(\mathbf{x})$. Thus, for the remainder of the section, we will consider the case when the server tries to cheat. We will show that if the server is able to pass the protocol in Algorithm 22 with high enough probability, then the server indeed has stored $\mathbf{x}$.

Algorithm 22 Verification for Data Possession Verification

INPUT: Index i of data $\mathbf{x} \in \mathbb{D}$
OUTPUT: 1 if Server has $\mathbf{x}$ and 0 otherwise

1: Client sends a *challenge* (i, j) to the server.
2: Client receives an answer a.
3: IF $a = z$ THEN
4: RETURN 1
5: ELSE
6: RETURN 0

Before we formally prove the result, we state our assumptions on what the server can and cannot do. We assume that the server follows the following general protocol. First, when the server receives $\mathbf{x}$, it does performs some computation (that terminates) on $\mathbf{x}$ to produce $\mathbf{y}$ and then it stores $\mathbf{y}$. (E.g., the server could store $\mathbf{y} = \mathbf{x}$ or $\mathbf{y}$ could be a compressed version of $\mathbf{x}$.) Then when it receives the challenge (i, j) for $\mathbf{x}$, it uses another algorithm $\mathscr{A}$ and returns the answers $a \leftarrow \mathscr{A}(\mathbf{y}, j)$. We assume that $\mathscr{A}$ always terminates on any input.[4] Note that the server is allowed to use arbitrary (but finite) amount of time to compute its answer. Next, we will prove that under these assumptions, the server cannot cheat with too high a probability.

Theorem 15.4.1. *Assume that the hash family $\mathscr{H}$ is an ε-almost universal hash family. Then if the server passes the protocol in Algorithm 22 with probability $> \frac{1}{2} + \frac{\varepsilon}{2}$, then the server has enough information to recreate $\mathbf{x}$.*

Proof. To prove the claim, we present an algorithm that can compute $\mathbf{x}$ from $\mathbf{y}$. (Note that we do not need this algorithm to be efficient: it just needs to terminate with $\mathbf{x}$.) In particular, consider Algorithm 23.

Algorithm 23 Decompression Algorithm

INPUT: $\mathscr{A}, \mathbf{y}$
OUTPUT: $\mathbf{x}'$

1: $\mathbf{z} \leftarrow \left(\mathscr{A}(\mathbf{y}, j)\right)_{j \in [m]}$.
2: Run the MLD algorithm (Algorithm 1) for $C_{\mathscr{H}}$ on $\mathbf{z}$ and let $C_{\mathscr{H}}(\mathbf{x}')$ be its output.
3: RETURN $\mathbf{x}'$

To complete the proof, we will show that $\mathbf{x}' = \mathbf{x}$. Towards this end we claim that $\Delta(\mathbf{z}, C_{\mathscr{H}}(\mathbf{x})) < \frac{m}{2} \cdot (1 - \varepsilon)$. Assuming this is true, we complete the proof. Note that Proposition 15.3.1 implies that $C_{\mathscr{H}}$ has distance at least $m(1 - \varepsilon)$. Thus, Proposition 1.4.1 (in particular, its proof) implies that Algorithm 1 will return $C_{\mathscr{H}}(\mathbf{x})$ and thus, $\mathbf{x}' = \mathbf{x}$, as desired.

[4] We have stated the algorithm to be independent of $\mathbf{y}$ and j but that need not be the case. However later in the section, we will need the assumption that $\mathscr{A}$ is independent of $\mathbf{y}$ and j, so we will keep it that way.

Finally, we argue that $\Delta(\mathbf{z}, C_{\mathscr{H}}(\mathbf{x})) < m(1-\varepsilon)/2$. To see this note that if the server passes the protocol in Algorithm 22 (i.e. the client outputs 1), then it has to be the case that $z_j \stackrel{\text{def}}{=} \mathscr{A}(\mathbf{y}, j) = h_j(\mathbf{x})$. Recall that by definition of $C_{\mathscr{H}}$, $h_j(\mathbf{x}) = C_{\mathscr{H}}(\mathbf{x})_j$ and that the server passes the protocol with probability $> 1/2+\varepsilon/2$. Since j is chosen uniformly from $[m]$, this implies that for $> m(1/2+\varepsilon/2)$ positions j, $z_j = C_{\mathscr{H}}(\mathbf{x})_j$, which then implies the claim. □

15.4.1 Driving Down the Cheating Probability

One of the unsatisfying aspects of Theorem 15.4.1 is that the probability of catching a "cheating" server is strictly less than 50%.[5] It is of course desirable to drive this up as close to 100% as possible. One way to obtain this would be to "repeat" the protocol: i.e. the client can choose multiple random hashes and store the corresponding values (in Algorithm 21) and (in Algorithm 22) asks the server to send back all the hash values and accepts if and only if all the returned answers match with the stored hash values. This however, comes at a cost: the client has to store more hash values (and also the communication cost between the client and the server goes up accordingly.)

Next we argue using list decoding that the protocol in Algorithm 21 and 22 (without any modification) gives a more powerful guarantee than the one in Theorem 15.4.1. To see why list decoding might buy us something, let us look back at Algorithm 23. In particular, consider Step 2: since we run MLD, we can only guarantee unique decoding up to half the (relative) distance of $C_{\mathscr{H}}$. This in turn leads to the bound of $1/2+\varepsilon/2$ in Theorem 15.4.1. We have seen that list decoding allows us to go beyond half the distance number of errors. So maybe, running a list decoding algorithm instead of MLD in Step 2 of Algorithms 23 would help us get better results. There are two potential issues that we'll need to tackle:

- We will need a general bound that shows that list decoding (arbitrarily) close to 100% is possible for any $C_{\mathscr{H}}$ for an ε-almost universal hash family; and
- Even if the above is possible, what will we do when a list decoding algorithm returns a *list* of possible data?

We will get around the first concern by using the Johnson bound 7.3.1. To get around the second issue we will indirectly use "side information" (like we mentioned in Section 7.2). For the latter, we will need the notion of Kolmogorov complexity, which captures the amount of information content in any given string. For our purposes, the following informal description is enough:

Definition 15.4.1. Given a string $\mathbf{x}$, its Kolmogorov complexity, denoted by $\mathscr{K}(\mathbf{x})$ is the minimum of $|\mathbf{y}|+|\mathscr{D}|$, where $\mathscr{D}$ is a decompression algorithm that on input $\mathbf{y}$ outputs $\mathbf{x}$ (where $|\mathbf{x}|$ and $|\mathscr{D}|$ are the length of $\mathbf{x}$ and (a description of) $\mathscr{D}$ in bits).

Informally, $\mathscr{K}(\mathbf{x})$ is the amount of information that can be obtained algorithmically from $\mathbf{x}$. Kolmogorov complexity is a fascinating topic that it outside the scope of this book. Here we will

[5] Note that Theorem 15.4.1 states that a server that cannot recreate $\mathbf{x}$ can pass the test with probablity at most $1/2+\varepsilon/2$. In other words, the probability that such a server is caught is at most $1/2-\varepsilon/2 < 1/2$.

only need to use the definition of $\mathcal{K}(\mathbf{x})$. We are now ready to prove the following list decoding counterpart of Theorem 15.4.1:

Theorem 15.4.2. *Assume that the hash family $\mathcal{H}$ is an ε-almost universal hash family. Then if the server passes the protocol in Algorithm 22 with probability $> \sqrt{\varepsilon}$, then the amount of information server has stored for $\mathbf{x}$ is at least $\mathcal{K}(\mathbf{x}) - O(\log|\mathbf{x}|)$.*

We note that the above is not a strict generalization of Theorem 15.4.1, as even though probability of catching a cheating server has gone up our guarantee is weaker. Unlike Theorem 15.4.1, where we can guarantee that the server can re-create $\mathbf{x}$, here we can only guarantee "storage enforceability"– i.e. we can only force a server to store close to $\mathcal{K}(\mathbf{x})$ amounts of memory.

Proof of Theorem 15.4.2. Here is the main idea of the proof. We first assume for the sake of contradiction that $|\mathbf{y}| < \mathcal{K}(\mathbf{x}) - O(\log(|\mathbf{x}|))$. Then using we construct a decompression algorithm $\mathcal{D}$ that on given input $\mathbf{y}$ and $O(\log(|\mathbf{x}|))$ extra information (or "advice"), outputs $\mathbf{x}$. Then we will show this overall contradicts the definition of $\mathcal{K}(\mathbf{x})$ (as this gives an overall smaller description of $\mathbf{x}$).

Before we state the decompression algorithm, we recall some facts. First note that $C_{\mathcal{H}}$ by Proposition 15.3.1 is a q-ary code (with $|\Sigma| = q$) with distance $m(1-\varepsilon)$. Further, by the Johnson bound (Theorem 7.3.1), $C_{\mathcal{H}}$ is a $(1-\sqrt{\varepsilon}, L)$-list decodable, where

$$L \leq qm^2. \tag{15.1}$$

Next, in Algorithm 24, we present a decompression algorithm that can compute $\mathbf{x}$ from $\mathbf{y}$ and an advice string $a \in [L]$. (As before, we do not need this algorithm to be efficient: it just needs to terminate with $\mathbf{x}$.)

Algorithm 24 Decompression Algorithm Using List Decoding

INPUT: $\mathscr{A}, \mathbf{y}, a$
OUTPUT: $\mathbf{x}$

1: $\mathbf{z} \leftarrow \left(\mathscr{A}(\mathbf{y}, j)\right)_{j\in[m]}$.
2: $\mathscr{L} \leftarrow \emptyset$.
3: FOR $\mathbf{x}' \in \mathbb{D}$ DO
4: IF $\Delta(C_{\mathcal{H}}(\mathbf{x}'), \mathbf{z}) \leq (1-\sqrt{\varepsilon})m$ THEN
5: Add $\mathbf{x}'$ to $\mathscr{L}$
6: RETURN The ath element in $\mathscr{L}$

To complete the proof, we claim that there exists a choice of $a \in [L]$ such that Algorithm 24 outputs $\mathbf{x}$. Note that this implies that $(\mathbf{y}, a)$ along with Algorithm 24 gives a complete description of $\mathbf{x}$. Now note that Algorithm 24 can be described in $O(1)$ bits. This implies that the size of this description is $|\mathbf{y}| + \log L + O(1)$, which by Definition 15.4.1 has to be at least $\mathcal{K}(\mathbf{x})$. This implies that

$$|\mathbf{y}| \geq \mathcal{K}(\mathbf{x}) - |a| - O(1) = \mathcal{K}(\mathbf{x}) - \log L - O(1) \geq \mathcal{K}(\mathbf{x}) - O(\log|\mathbf{x}|),$$

where the last inequality follows from (15.1).

Next, we argue the existence of an appropriate $a \in [L]$. Towards this end we claim that $\Delta(\mathbf{z}, C_{\mathscr{H}}(\mathbf{x})) < m(1-\sqrt{\varepsilon})$. Note that this implies that $\mathbf{x} \in \mathscr{L}$. Since $|\mathscr{L}| \leq L$, then we can just assign a to be the index of $\mathbf{x}$ in $\mathscr{L}$. Finally, we argue that $\Delta(\mathbf{z}, C_{\mathscr{H}}(\mathbf{x})) < m(1-\sqrt{\varepsilon})$. To see this note that if the server passes the protocol in Algorithm 22 (i.e. the client outputs 1), then it has to be the case that $z_j \stackrel{\text{def}}{=} \mathscr{A}(\mathbf{y}, j) = h_j(\mathbf{x})$. Recall that by definition of $C_{\mathscr{H}}$, $h_j(\mathbf{x}) = C_{\mathscr{H}}(\mathbf{x})_j$ and that the server passes the protocol with probability $> \sqrt{\varepsilon}$. Since j is chosen uniformly from $[m]$, this implies that for $> m\sqrt{\varepsilon}$ positions j, $z_j = C_{\mathscr{H}}(\mathbf{x})_j$, which then implies the claim. □

15.5 Bibliographic Notes

Universal hash functions were defined in the seminal paper of Carter and Wegman [6]. Almost universal hash function family was defined by Stinson [70].

Kolmogorov complexity was defined by Kolmogorov [48]. For a thorough treatment see the textbook by Li and Vitányi [52].

Chapter 16

Securing Your Fingerprints: Fuzzy Vaults

String-based passwords are the dominant mode of authentication in today's information-reliant world. Indeed, all of us use passwords on a regular basis to authenticate ourselves in almost any online activity. Strings have become widespread due to several nice mathematical properties. First, matching two strings (that is, checking if two strings are exactly the same) is computationally very fast (and easy). Second, and more importantly, there exist secure hash functions that map a string x to another string $h(x)$ such that, given $h(x)$, determining x is hard. Furthermore, since $h(x)$ is itself a string, we can check if a claimed password y is the same as the original string x by comparing $h(y)$ and $h(x)$, which (as we just observed) is easy to do. This implies that the server performing the authentification only needs to store the hashes $h(x)$ of the original passwords. Hence, even if the list of hashed passwords were compromised, the passwords themselves would remain secure.

The above scheme is perfect as long as the passwords x are "random enough," and this can be achieved if the passwords were generated randomly by some automated process. However, in real life passwords are generated by humans and are not really random. (One of the most quoted facts is that the most commonly used password is the string "password" itself.) Further, we tend to forget passwords, which has lead to the near ubiquity of "Forgot passwords" links in places where we need to login.

One alternative that has been gaining traction in the last decade or so is to use a user's fingerprint as their password. The big advantage is that it is hard to "forget" one's fingerprint. In this chapter, we will look at the issues in using fingerprints as passwords and see how Reed-Solomon codes can help.

16.1 Some quick background on fingerprints

For the time being let us assume that we can somehow convert a fingerprint (image) somehow to a string f. (We will see more details on this shortly.) Then we have the following naive solution:

Naive Solution. Use any off-the-shelf hash function h for strings and then store $h(f)$ instead of f.

To see the issues with the naive solution, we first need to know a little bit about how fingerprints are stored. The standard way to store a fingerprint is as a collection of triples, called minutia. Each minutia point is located where one ridge splits into two, or where one ridge ends. The i^{th} minutia is the triple (x_i, y_i, θ_i), where x_i and y_i are the x and y coordinates of a point on the finger, and θ_i indicates the direction of the ridge that created the minutia point relative to the plane.

The main issue with our naive solution is that two fingerprint readings will never be exactly the same, even if the same finger is used. For any two fingerprint readings, the following issues may produce errors:

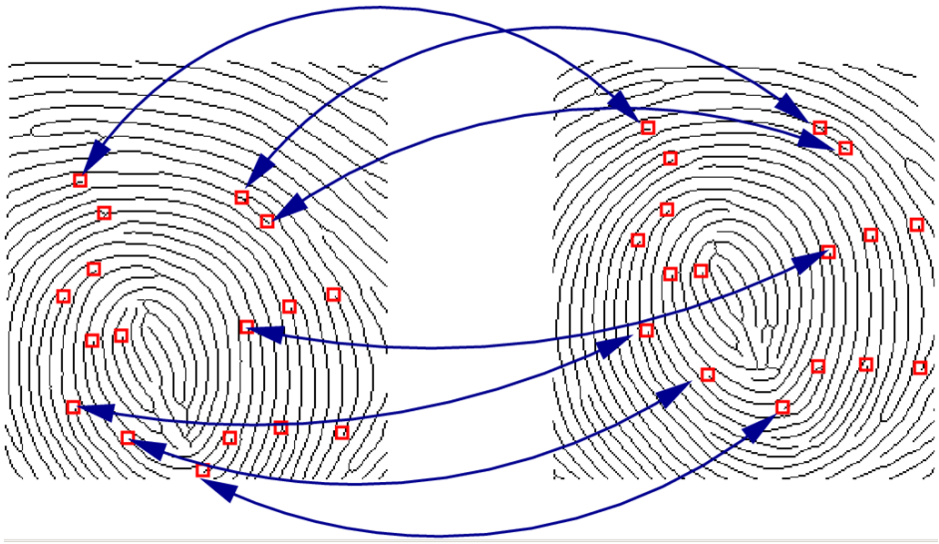
1. Translation and rotation, even when using the same finger.

2. Varying pressure.

3. Each reading may not completely overlap with the ideal fingerprint region (i.e., the finger may be slightly tilted).

4. The minutia are not ordered, so they form a set instead of a vector. Of course one can sort the set to produce a string, but in conjunction with the earlier issue (especially those involving rotation and translation) this implies that the values of (x_i, y_i, θ_i) by themselves are not that important. Furthermore the fact that two readings might not have complete overlap means that we are interested in matching readings that have significant overlap, so it turns out that the set notation is deal to theoretically deal with the issues.

Figure 16.1: The minutia are unordered and form a set, not a vector.



We can now see that the naive solution is inadequate. Even if we could somehow correct the first three issues, existing hash functions for strings require a vector, not a set, so our naive solution will fail.

Remark 16.1.1. The four problems that came up in our naive solution will come up in any solution we propose. Technology has not yet developed to the point where we can securely eliminate these issues, which is why there are no prevalent secure commercial systems that safeguard secrets using fingerprints. (The reason government agencies, such as the police or FBI, use fingerprinting is because there is an inherent trust that the government will keep your data secure, even when it does not apply a good hash function to it.)

Thus, what we are looking for are secure hash functions designed to handle the additional challenges posed by fingerprints. We would like to mention that for fingerprints to replace strings as passwords, the hash function needs to satisfy both of these properties *simultaneously*: (i) we should be able to match hashes from the "same" fingerprint and (ii) an adversary should not be able to "break" the hash function.

16.2 The Fuzzy Vault Problem

We begin with the fuzzy vault problem, which is slightly different from the one we have been studying so far. Say you have a secret string, s, and you want to store it in a secure way. Instead of using a password, you want to use your fingerprint, f, to "lock" the secret s. You want the locked version of your secret to have two properties:

1. You should be able to "unlock" the locked version of s
2. No one else should be able to "unlock" s

We claim that if we can solve the above problem, then we can solve the problem of designing a secure hash function for fingerprints. We leave the details as an exercise. (Hint: pick s at random and then in addition to the locked version of s also store $h(s)$, where h is an off-the-shelf secure hash function for strings.)

We will now formalize the fuzzy vault problem.

16.2.1 Formal Problem Statement

The first thing we need to do is quantize the measurements of the minutia. We cannot be infinitely precise in our measurements anyways, so let us assume that all quantized minutia, (x_i, y_i, θ_i), can be embedded into $\mathbb{F}_q$ for some large enough prime power q. Theoretically, this can also help to correct the first two issues from our naive solution. We could go through all possible values in $\mathbb{F}_q$ to get rid of translation and rotation errors (*e.g.*. for every $(\Delta x, \Delta y, \Delta z) \in \mathbb{F}_q$, we rotate and translate each minutia (x_i, y_i, z_i) to $(x_i + \Delta x, y_i + \Delta y, z_i + \Delta z)$). [1] We can also do some

[1] To be more precise we first perform the translation and rotation over the reals and then quantize and map to the appropriate $\mathbb{F}_q$ value.

local error correction to a quantized value to mitigate the effect of varying pressure. We stress that going over all possible shifts is not a practical solution, but theoretically this can still lead to a polynomial-time solution.

We now formally define our problem, which primarily captures issues 3 and 4. (Below for any integers $t \geq 1$, $\binom{\mathbb{F}_q}{t}$ denotes the set of all subsets of $\mathbb{F}_q$ of size exactly t.) The following are the components of the problem:

- Integers $k \geq 1$, $n \geq t \geq 1$
- Secret $s \in \mathbb{F}_q^k$
- Fingerprint $f \in \binom{\mathbb{F}_q}{t}$
- $\text{LOCK} : \mathbb{F}_q^k \times \binom{\mathbb{F}_q}{t} \to \binom{\mathbb{F}_q}{n}$
- $\text{UNLOCK} : \binom{\mathbb{F}_q}{t} \times \binom{\mathbb{F}_q}{n} \to \mathbb{F}_q^k$

The goal of the problem is to define the functions LOCK and UNLOCK such that they satisfy these two properties (for some $c < t$):

1. (c-completeness.) For any $f, f' \in \binom{\mathbb{F}_q}{t}$ such that $|f - f'| \leq c$, the following should hold:

$$\text{UNLOCK}\left(\text{LOCK}(s, f), f'\right) = s.$$

2. (Soundness.) It should be "hard" for an adversary to get s from $\text{LOCK}(s, f)$. (The notion of "hard" will be more formally defined later.)

Note that the completeness property corresponds to the matching property we need from our hash function, while the soundness property corresponds to the security property of the hash function.

16.2.2 Two Futile Attempts

We begin with two attempts at designing the LOCK and UNLOCK functions, which will not work. However, later we will see how we can combine both to get our final solution.

For this section, unless mentioned otherwise, we will assume that the original fingerprint f is given by the set $\{\alpha_1, \ldots, \alpha_t\}$.

Attempt 1. We begin with a scheme that focuses on the soundness property. A very simple idea, which is what we will start off with, would be to just add $n-t$ random values to f to get our vault. The intuition, which can be made precise, is that an adversary just looking at the vault will just see random points and will not be able to recover f from the random set of points. The catch of course that this scheme has terrible completeness. In particular, if we get a match between a value in the second fingerprint f' and the vault, we have no way to know whether the match is to one of the original values in f or if the match is with one of the random "chaff" points there were added earlier.

Attempt 2. Next, we specify a scheme that has good completeness (but has pretty bad soundness).

We begin with the LOCK function:

$$\text{LOCK}_2(s, f) = \{(\alpha_1, P_s(\alpha_1)), \ldots, (\alpha_t, P_s(\alpha_t))\},$$

where $P_s(X) = \sum_{i=0}^{k-1} s.X^i$ and recall $f = \{\alpha_1, \ldots, \alpha_t\}$. (Note that we have $n = t$.)

The main intuition behind LOCK_2 is the following. Given another fingerprint $f' = \{\beta_1, \ldots, \beta_t\}$ such that it is close enough to f, i.e. $|f \setminus f'| \leq c$, for every value in $f \cap f'$, we will know the corresponding P_s value and thus, we can use the fact that we can decode Reed-Solomon codes from erasures to recover the secret s. We formally present UNLOCK_2 as Algorithm 25.

Algorithm 25 UNLOCK_2

INPUT: Vault $\{(\alpha_1, y_1), \ldots, (\alpha_t, y_t)\} = \text{LOCK}(s, f)$ and another fingerprint $f' = \{\beta_1, \ldots, \beta_t\}$
OUTPUT: s if $|f \setminus f'| \leq c$

1: FOR $i = 1, \ldots, t$ DO
2: IF there exists a $j \in [t]$ such that $\alpha_i = \beta_j$ THEN
3: $z_j \leftarrow y_i$
4: ELSE
5: $z_j \leftarrow ?$
6: $\mathbf{z} \leftarrow (z_1, \ldots, z_t)$
7: Run Algorithm from Theorem 11.2.1 to correct $\mathbf{z}$ from erasures for RS codes with evaluation points $\{\beta_1, \ldots, \beta_t\}$ and output resulting message as s.

The following result is fairly simple to argue.

Lemma 16.2.1. *The pair* $(\text{LOCK}_2, \text{UNLOCK}_2)$ *of functions is* $(t-k)$*-complete. Further, both functions can be implemented in polynomial time.*

Proof. Let us assume $|f \setminus f'| \leq t-k$. Now as both f and f' have exactly t values, this means that $\mathbf{z}$ has at most $t-k$ erasures. Thus, by Theorem 11.2.1, Step 6 will output s and UNLOCK_2 can be implemented in polynomial time. Further, the claim on the polynomial run time of LOCK_2 follows from the fact that one can do encoding of Reed-Solomon code in polynomial time. □

Unfortunately, ($\text{LOCK}_2, \text{UNLOCK}_2$) pair has terrible soundness. This is because the vault $\{(\alpha_1, y_1), \ldots, (\alpha_t, y_t)\}$ has f in the first component in each pair. This an adversary can just read off those values and present $f' = \{\alpha_1, \ldots, \alpha_t\}$, which would imply that $\text{UNLOCK}_2(\text{LOCK}_2(s, f), f') = s$, which means that the vault would be "broken."

16.3 The Final Fuzzy Vault

So far we have seen two attempts: one that (intuitively) has very good soundness but no completeness and another which has good completeness but terrible soundness. It is natural to consider if we can combine both of these attempts and get the best of both worlds. Indeed, it turns we can easily combine both of our previous attempts to get the final fuzzy vault.

Algorithm 26 presents the new LOCK_3 function.

Algorithm 26 LOCK_3

INPUT: Fingerprint $f = \{\alpha_1, \ldots, \alpha_t\}$ and secret $s = (s_0, \ldots, s_{k-1}) \in \mathbb{F}_q^k$
OUTPUT: Vault with f locking s

1: $R, T \leftarrow \emptyset$
2: $P_s(X) \leftarrow \sum_{i=0}^{k-1} s_i \cdot X^i$
3: FOR $i = 1, \ldots, t$ DO
4: $\quad T \leftarrow T \cup \{\alpha_i\}$
5: FOR $i = t+1, \ldots, n$ DO
6: $\quad \alpha_i$ be a random element from $\mathbb{F}_q \setminus T$
7: $\quad T \leftarrow T \cup \{\alpha_i\}$
8: FOR every $\alpha \in T$ DO
9: $\quad \gamma$ be a random element from $\mathbb{F}_q \setminus P_s(\alpha)$
10: $\quad R \leftarrow R \cup \{(\alpha, \gamma)\}$
11: Randomly permute R
12: RETURN R

Algorithm 27 presents the new UNLOCK_3 function.

The following result is a simple generalization of Lemma 16.2.1.

Lemma 16.3.1. *The pair* ($\text{LOCK}_3, \text{UNLOCK}_3$) *of functions is* $(t-k)/2$*-complete. Further, both functions can be implemented in polynomial time.*

Proof. Let us assume $|f \setminus f'| \leq (t-k)/2$. Now as both f and f' have exactly t values, it implies that $|f \cap f'| \geq (t+k)/2$. Further for each $j \in [t]$ such that $\beta_j \in f \cap f'$, we have that $z_j = P_s(\beta_j)$. In other words, this means that $\mathbf{z}$ has at most $(t-k)/2$ errors.[2] Thus, by Theorem 11.2.2, Step 6 will output s and UNLOCK_3 can be implemented in polynomial time. Further, the claim on the polynomial run time of LOCK_3 follows from the fact that one can do encoding of Reed-Solomon code in polynomial time. □

[2] To be more precise if $\mathbf{z}$ has e errors and s erasures w.r.t. the codeword corresponding to s, then $2e + s \leq t - k$.

Algorithm 27 UNLOCK$_2$

INPUT: Vault $\{(\alpha_1, y_1), \ldots, (\alpha_n, y_n)\} = \text{LOCK}(s, f)$ and another fingerprint $f' = \{\beta_1, \ldots, \beta_t\}$
OUTPUT: s if $|f \setminus f'| \leq c$

1: FOR $i = 1, \ldots, t$ DO
2: IF there exists a $j \in [n]$ such that $\alpha_i = \beta_j$ THEN
3: $z_j \leftarrow y_i$
4: ELSE
5: $z_j \leftarrow ?$
6: $\mathbf{z} \leftarrow (z_1, \ldots, z_t)$
7: Run Algorithm from Theorem 11.2.2 to correct $\mathbf{z}$ from errors and erasures for RS codes with evaluation points $\{\beta_1, \ldots, \beta_t\}$ and output resulting message as s.

16.3.1 Soundness

To avoid getting into too much technical details, we will present a high level argument for why the proposed fuzzy vault scheme has good soundness. Given a vault $\{(\alpha_1, y_1), \ldots, (\alpha_n, y_n)\} = \text{LOCK}_3(s, f)$, we know that there are exactly t values (i.e. those $\alpha_j \in f$) such that the polynomial $P_s(X)$ agrees with the vault on exactly those t points. Thus, an intuitive way to argue the soundness of the vault would be to argue that there exists a lot other secrets $s' \in \mathbb{F}_q^k$ such that $P_{s'}(X)$ also agrees with the vault in exactly t positions. (One can formalize this intuition and argue that the vault satisfies a more formal definition of soundness but we will skip those details.)

We will formalize the above argument by proving a slightly different result (and we will leave the final proof as an exercise).

Lemma 16.3.2. *Let $V = \{(x_1, y_1), \ldots, (x_n, y_n)$ be n independent random points from $\mathbb{F}_q \times \mathbb{F}_q$. Then, in expectation, there are at least $\frac{1}{3} \cdot q^k \cdot \left(\frac{n}{qt}\right)^t$ polynomials $P(X)$ of degree at most $k-1$ such that for exactly t values of $j \in [n]$, we have $P(x_j) = y_j$.*

Proof. Consider a fixed polynomial $P(X)$ and a $j \in [n]$. Then for any $x_j \in \mathbb{F}_q$, the probability that for a random $y_j \in F_q$, $P(x_j) = y_j$ is exactly $1/q$. Further, these probabilities are all independent. This implies that the probability that $P(X)$ agrees with V in exactly t positions is given by

$$\binom{n}{t} \cdot \left(\frac{1}{q}\right)^t \cdot \left(1 - \frac{1}{q}\right)^{n-t} \geq \frac{1}{3}\left(\frac{n}{qt}\right)^t.$$

Since there are q^k such polynomials, the claimed result follows. □

We note that there are two aspects of the above lemma that are not satisfactory. (i) The result above is for a vault V with completely random points whereas we would like to prove a similar result but with $V = \text{LOCK}_3(s, f)$ and (ii) Instead of a bound in expectation, we would like to prove a similar exponential lower bound but with high probability. We leave the proof that these can be done as an exercise. (Hint: Use the "Inverse Markov Inequality.")

16.4 Bibliographic Notes

The fuzzy vault presented in this chapter is due to Juels and Sudan [43]. The “inverse Markov inequality" first appeared in Dumer et al. [15].

Chapter 17

Finding Defectives: Group Testing

Consider the following situation that arises very frequently in *biological screens*. Say there are N individuals and the objective of the study is to identify the individuals with a certain "biomarker" that could e.g. indicate that the individual has some specific disease or is at risk for a certain health condition. The naive way to do this would be to test each person individually, that is:

1. Draw sample (e.g. a blood or DNA sample) from a given individual,
2. Perform required tests, then
3. Determine presence or absence of the biomarker.

This method of one test per person will gives us a total of N tests for a total of N individuals. Say we had more than $70-75\%$ of the population infected. At such large numbers, the use of the method of individual testing is reasonable. However, our goal is to achieve effective testing in the more likely scenario where it doesn't make sense to test $100,000$ people to get just (say) 10 positives.

The feasibility of a more effective testing scheme hinges on the following property. We can combine blood samples and test a combined sample together to check if at least one individual has the biomarker.

The main question in group testing is the following: If we have a very large number of items to test, and we know that only certain few will turn out positive, what is a nice and efficient way of performing the tests?

17.1 Formalization of the problem

We now formalize the group testing problem. The input to the problem consists of the following:

- The total number of individuals: N.
- An upper bound on the number of infected individuals d.

- The input can be described as a vector $\mathbf{x} = (x_1, x_2, ..., x_n)$ where $x_i = 1$ if individual i has the biomarker, else $x_i = 0$.

Note that $wt(\mathbf{x}) \leq d$. More importantly, notice that the vector $\mathbf{x}$ is an implicit input since we do not know the positions of 1s in the input. The only way to find out is to run the *test*s. Now, we will formalize the notion of a test.

A query/test S is a subset of $[N]$. The answer to the query $S \subseteq [N]$ is defined as follows:

$$A(S) = \begin{cases} 1 & \text{if } \sum_{i \in S} x_i \geq 1; \\ 0 & \text{otherwise.} \end{cases}$$

Note that the answer to the S is the logical-OR of all bits in S, i.e. $A(S) = \bigvee_{i \in S} x_i$.

The goal of the problem is to compute $\mathbf{x}$ and minimize the number of tests required to determine $\mathbf{x}$.

Testing methods. There is another aspect of the problem that we need specify. In particular, we might need to restrict how the tests interact with each other. Below are two commons ways to carry out tests:

1. *Adaptive group testing* is where we test a given subset of items, get the answer and base our further tests on the outcome of the previous set of tests and their answers.

2. *Non-Adaptive group testing* on the other hand is when all our tests are set even before we perform our first test. That is, all our tests are decided a priori.

Non-adaptive group testing is crucial for many applications. This is because the individuals could be geographically spread pretty far out. Due to this, adaptive group testing will require a very high degree of co-ordination between the different groups. This might actually increase the cost of the process.

Notation. We will also need notation to denote the minimum number of tests needed in group testing. Towards this end, we present the following two definitions.

Definition 17.1.1 ($t(d,N)$)**.** Given a subset of N items with d defects represented as $\mathbf{x} \in \{0,1\}^N$, the minimum number of *non-adaptive* tests that one would have to make is defined as $t(d,N)$.

Definition 17.1.2. $t^a(d,N)$: Given a set of N items with d defects, $t^a(d,N)$ is defined as the number of *adaptive* tests that one would have to make to detect all the defective items.

The obvious questions are to prove bounds on $t(d,N)$ and $t^a(d,N)$:

Question 17.1.1. *Prove asymptotically tight bounds on* $t(d,N)$.

Question 17.1.2. *Prove asymptotically tight bounds on $t^a(d,N)$.*

We begin with some simple bounds:

Proposition 17.1.1. *For every $1 \le d \le N$, we have*

$$1 \le t^a(d,N) \le t(d,N) \le N.$$

Proof. The last inequality follows from the naive scheme of testing all individuals with singleton tests while the first inequality is trivial. The reason for $t^a(d,N) \le t(d,N)$ is due to the fact that any non-adaptive test can be performed by an adaptive test by running all of the tests in the first step of the adaptive test. Adaptive tests can be faster than non-adaptive tests since the test can be changed after certain things are discovered. □

Representing the set of tests as a matrix. It turns out that is is useful to represent a non-adaptive group testing scheme as a matrix. Next, we outline the natural such representation. For, $S \subseteq [N]$, define $\chi_S \in \{0,1\}^N$ such that $i \in S$ if and only if $\chi_S(i) = 1$. Consider a non-adaptive group testing scheme with t test $S_1, \ldots, S_t$. The corresponding matrix M is a $t \times N$ matrix where the ith row is χ_{S_i}. (Note that the trivial solution of testing each individual as a singleton set would just be the $N \times N$ identity matrix.) In other words, $M = \{M_{ij}\}_{i\in[t], j\in[N]}$ such that $M_{ij} = 1$ if $j \in S_i$ and $M_{ij} = 0$ otherwise.

If we assume that multiplication is logical AND ($\bigwedge$) and addition is logical OR ($\bigvee$), then we have $M \times \mathbf{x} = \mathbf{r}$ where $\mathbf{r} \in \{0,1\}^t$ is the vector of the answers to the t tests. We will denote this operation as $M \odot \mathbf{x}$. To think of this in terms of testing, it is helpful to visualize the matrix-vector multiplication. Here, $\mathbf{r}$ will have a 1 in position i if and only if there was a 1 in that position in both M and $\mathbf{x}$ i.e. if that person was tested with that particular group and if he tested out to be positive.

Thus, our goal is to get to compute $\mathbf{x}$ from $M \odot \mathbf{x}$ with as small a t as possible.

17.2 Bounds on $t^a(d,N)$

In this section, we will explore lower and upper bounds on $t^a(d,N)$ with the ultimate objective of answering Question 17.1.2.

We begin with a lower bound that follows from a simple counting argument.

Proposition 17.2.1. *For every $1 \le d \le N$,*

$$t^a(d,N) \ge d \log \frac{N}{d}.$$

Proof. Fix any valid adaptive group testing scheme with t tests. Observe that if $\mathbf{x} \ne \mathbf{y} \in \{0,1\}^N$, with $wt(\mathbf{x}), wt(\mathbf{y}) \le d$ then $\mathbf{r}(\mathbf{x}) \ne \mathbf{r}(\mathbf{y})$, where $\mathbf{r}(\mathbf{x})$ denotes the result vector for running the tests on $\mathbf{x}$ and similarly for $\mathbf{r}(\mathbf{y})$. The reason for this is because two valid inputs cannot give the same

result. If this were the case and the results of the tests gave $\mathbf{r}(\mathbf{x}) = \mathbf{r}(\mathbf{y})$ then it would not be possible to distinguish between $\mathbf{x}$ and $\mathbf{y}$.

The above observation implies that total number of distinct test results is the number distinct binary vectors with Hamming weight at most d, i.e. $Vol_2(d,N)$. On the other hand, the number of possible distinct t-bit vectors is at most 2^t, which with the previous argument implies that

$$2^t \geq Vol_2(d,N)$$

and hence, it implies that

$$t \geq \log Vol_2(d,N).$$

Recall that

$$Vol_2(d,N) \geq \binom{N}{d} \geq \left(\frac{N}{d}\right)^d,$$

where the first inequality follows from (3.23) and the second inequality follows from Lemma B.1.1. So $t \geq d\log(N/d)$, as desired. □

It turns out that $t^a(d,N)$ is also $O\left(d\log\left(\frac{N}{d}\right)\right)$. (See Exercise 17.1.) This along with Proposition 17.2.1 implies that $t^a(d,N) = \Theta\left(d\log\left(\frac{N}{d}\right)\right)$, which answers Question 17.1.2. The upper bound on $t^a(d,N)$ follows from an adaptive group testing scheme and hence does not say anything meaningful for Question 17.1.1. (Indeed, we will see later that $t(d,N)$ cannot be $O(d\log(N/d))$.) Next, we switch gears to talk about non-adaptive group testing.

17.3 Bounds on $t(d,N)$

We begin with the simplest case of $d = 1$. In this case it is instructive to recall our goal. We want to define a matrix M such that given any $\mathbf{x}$ with $wt(\mathbf{x}) \leq 1$, we should be able to compute $\mathbf{x}$ from $M \odot \mathbf{x}$. In particular, let us consider the case when $\mathbf{x} = \mathbf{e}_i$ for some $i \in [N]$. Note that in this case $M \odot \mathbf{x} = M^i$, where M^i is the ith column of M. Hence we should design M such that M^i uniquely defined i. We have already encountered a similar situation before in Section 2.6 when trying to decode the Hamming code. It turns out that is suffices to pick M as the parity check matrix of a Hamming code. In particular, we can prove the following result:

Proposition 17.3.1. $t(1,N) \leq \lceil \log(N+1) \rceil + 1$

Proof. We prove the upper bound by exhibiting a matrix that can handle non adaptive group testing for $d = 1$. The group test matrix M is the parity check matrix for $[2^m - 1, 2^m - m - 1, 3]_2$, i.e. H_m where the i-th column is the binary representation of i (recall Section 2.4). This works because when performing $H_m \odot \mathbf{x} = \mathbf{r}$, if $wt(\mathbf{v}) \leq 1$ then $\mathbf{r}$ will correspond to the binary representation of i. Further, note that if $wt(\mathbf{x}) = 0$, then $\mathbf{r} = \mathbf{0}$, which is exactly $\mathbf{x}$. Hence, $M \odot \mathbf{x}$ uniquely identifies $\mathbf{x}$ when $wt(\mathbf{x}) \leq 1$, as desired.

If $N \neq 2^m - 1$ for any m, the matrix H_m corresponding to the m such that $2^{m-1} - 1 < N < 2^m - 1$ can be used by adding 0s to the end of $\mathbf{x}$. By doing this, decoding is "trivial" for both cases since the binary representation is given for the location. So the number of tests is at most $\lceil \log(N+1) \rceil + 1$, which completes the proof. □

Note that Propositions 17.1.1 and 17.2.1 imply that $t(d,N) \geq \log N$, which with the above result implies that $t(1,N) = \Theta(\log N)$. This answers Question 17.1.1 for the case of $d = 1$. We will see later that such a tight answer is not known for larger d. However, at the very least we should try and extend Proposition 17.3.1 for larger values of d. In particular,

Question 17.3.1. *Prove asymptotic upper bounds on $t(d,N)$ that hold for every $1 < d \leq N$.*

We would like to point out something that was implicitly used in the proof of Proposition 17.3.1. In particular, we used the implicitly understanding that a non-adaptive group testing matrix M should have the property that given any $\mathbf{x} \in \{0,1\}^N$ such that $wt(\mathbf{x}) \leq d$, the result vector $M \odot \mathbf{x}$ should uniquely determine $\mathbf{x}$. This notion is formally captured in the following definition of non-adaptive group testing matrix:

Definition 17.3.1. A $t \times N$ matrix M is *d-separable* if and only if for every $S_1 \neq S_2 \subseteq [N]$ such that $|S_1|, |S_2| \leq d$, we have

$$\bigcup_{j \in S_1} M^j \neq \bigcup_{i \in S_2} M^i.$$

In the above we think of a columns $M^i \in \{0,1\}^t$ as a subset of $[t]$ and for the rest of this chapter we will use both views of the columns of a group testing matrix. Finally, note that the above definition is indeed equivalent to our earlier informal definition since for any $\mathbf{x} \in \{0,1\}^N$ with $wt(\mathbf{x}) \leq d$, the vector $M \odot \mathbf{x}$ when thought of as its equivalent subset of $[t]$ is exactly the set $\cup_{i \in S} M^i$, where S is the support of $\mathbf{x}$, i.e. $S = \{i | x_i = 1\}$.

Like in the coding setting, where we cared about the run time of the decoding algorithm, we also care about the time complexity of the decoding problem (given $M \odot \mathbf{x}$ compute $\mathbf{x}$) for group testing. We will now look at the obvious decoding algorithm for d-separable matrices: just check all possible sets that could form the support of $\mathbf{x}$. Algorithm 28 has the details.

Algorithm 28 Decoder for Separable Matrices

INPUT: Result vector $\mathbf{r}$ and d-separable matrix M
OUTPUT: $\mathbf{x}$ if $\mathbf{r} = M \odot \mathbf{x}$ else `Fail`

1: $R \leftarrow \{i | r_i = 1\}$.
2: FOR Every $T \subseteq [N]$ such that $|T| \leq d$ DO
3: $\quad S_T \leftarrow \cup_{i \in T} M^i$
4: $\quad$ IF $R = S_T$ THEN
5: $\quad\quad \mathbf{x} \leftarrow (x_1, \ldots, x_N) \in \{0,1\}^N$ such that $x_i = 1$ if and only $i \in T$.
6: $\quad\quad$ RETURN $\mathbf{x}$
7: RETURN `Fail`

The correctness of Algorithm 28 follows from Definition 17.3.1. Further, it is easy to check that this algorithm will run in $N^{\Theta(d)}$ time, which is not efficient for even moderately large d. This naturally leads to the following question:

Question 17.3.2. *Do there exists d-separable matrices that can be efficient decoded?*

We would like to remark here that the matrices that we seek in the answer to Question 17.3.2 should have small number of tests (as otherwise the identity matrix answers the question in the affirmative).

17.3.1 Disjunct Matrices

We now define a stronger notion of group testing matrices that have a more efficient decoding algorithm than d-separable matrices.

Definition 17.3.2. A $t \times N$ matrix M is d-disjunct if and only if for every $S \subset [N]$ with $|S| \leq d$ and for every $j \notin S$, there exist an $i \in [t]$ such that $M_{ij} = 1$ but for all $k \in S$, $M_{ik} = 0$. Or equivalently

$$M^j \not\subseteq \bigcup_{k \in S} M^k.$$

For an illustration of the definition, see Figure 17.3.2.

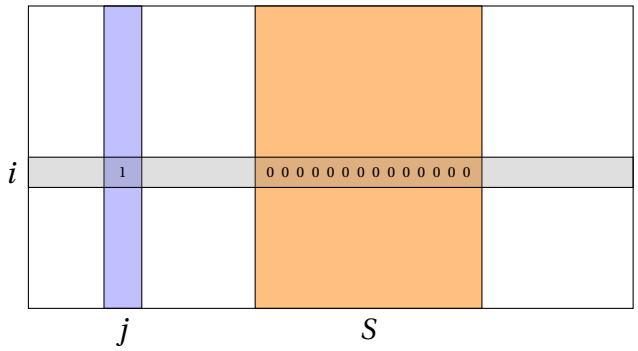


Figure 17.1: Pick a subset S (not necessarily contiguous). Then pick a column j that is not present in S. There will always be i such that row i has a 1 in column j and all zeros in S.

Next we argue that disjunctness is a sufficient condition for separability.

Lemma 17.3.2. *Any d-disjunct matrix is also d-separable.*

Proof. For contradiction, assume M is d-disjunct but not d-separable. Since M is not d-separable, then union of two subset $S \neq T \subset [N]$ of size at most d each are the same; i.e.

$$\bigcup_{k \in S} M^k = \bigcup_{k \in T} M^k.$$

Since $S \neq T$, there exists $j \in T \setminus S$. But we have

$$M^j \subseteq \bigcup_{k \in T} M^k = \bigcup_{k \in S} M^k,$$

where the last equality follows from the previous equality. However, since by definition $j \notin S$, the above contradicts the fact that M is d-disjunct. □

In fact, it turns out that disjunctness is also almost a necessary condition: see Exercise 17.2.

Next, we show the real gain of moving to the notion of disjunctness from the notion of separability.

Lemma 17.3.3. *There exists a $O(tN)$ time decoding algorithm for any $t \times N$ matrix that is d-disjunct.*

Proof. The proof follows from the following two observations.

First, say we have a matrix M and a vector $\mathbf{x}$ and $\mathbf{r} = M \odot \mathbf{x}$ such that $r_i = 1$. Then there exists a column j in matrix that made it possible i.e. if $r_i = 1$, then there exists a j such that $M_{ij} = 1$ and $x_j = 1$.

Second, let T be a subset and j be a column not in T where $T = \{\ell \mid x_\ell = 1\}$ and $|T| \leq d$. Consider the ith row such that T has all zeros in the ith row, then $r_i = 0$. Conversely, if $r_i = 0$, then for every $j \in [N]$ such that $M_{ij} = 1$, it has to be the case that $x_j = 0$. This naturally leads to the decoding algorithm in Algorithm 29.

The correctness of Algorithm 29 follows from the above observation and it can be checked that the algorithm runs in time $O(tN)$– see Exercise 17.3. □

Algorithm 29 Naive Decoder for Disjunct Matrices

INPUT: Result vector $\mathbf{r}$ and d-disjunct matrix M
OUTPUT: $\mathbf{x}$ if $M \odot \mathbf{x} = \mathbf{r}$ else `Fail`

```
1: Initialize x ∈ {0,1}^N to be the all ones vector
2: FOR every i ∈ [t] DO
3:     IF r_i = 0 THEN
4:         FOR Every j ∈ [N] DO
5:             IF M_ij = 1 THEN
6:                 x_j ← 0
7: IF M ⊙ x = r THEN
8:     RETURN x
9: ELSE
10:    RETURN Fail
```

Modulo the task of exhibiting the existence of d-disjunct matrices, Lemmas 17.3.3 and 17.3.2 answer Question 17.3.2 in the affirmative. Next, we will tackle the following question:

Question 17.3.3. *Design d-disjunct matrices with few rows.*

As we will see shortly answering the above question will make connection to coding theory becomes even more explicit.

17.4 Coding Theory and Disjunct Matrices

In this section, we present the connection between coding theory and disjunct matrices with the final goal of answering Question 17.3.3. First, we present a sufficient condition for a matrix to be d-disjunct.

Lemma 17.4.1. *Let $1 \leq d \leq N$ be an integer and M be a $t \times N$ matrix, such that*

(i) For every $j \in [N]$, the jth column has at least $w_{\min}$ ones in it, i.e. $\left|M^j\right| \geq w_{\min}$ and

(ii) For every $i \neq j \in [N]$, the i and j'th columns have at most $a_{\max}$ ones in common, i.e. $\left|M^i \cap M^j\right| \leq a_{\max}$

for some integers $a_{\max} \leq w_{\min} \leq t$. Then M is a $\left\lfloor \frac{w_{\min}-1}{a_{\max}} \right\rfloor$-disjunct.

Proof. For notational convenience, define $d = \left\lfloor \frac{w_{\min}-1}{a_{\max}} \right\rfloor$. Fix an arbitrary $S \subset [N]$ such that $|S| \leq d$ and a $j \notin S$. Note we have to show that

$$M^j \not\subseteq \cup_{i \in S} M^i,$$

or equivalently

$$M^j \not\subseteq \cup_{i \in S} \left(M^i \cap M^j\right).$$

We will prove the above by showing that

$$\left|M^j \setminus \cup_{i \in S} \left(M^i \cap M^j\right)\right| > 0.$$

Indeed, consider the following sequence of relationships:

$$\begin{aligned}
\left|M^j \setminus \cup_{i \in S} \left(M^i \cap M^j\right)\right| &= \left|M^j\right| - \left|\cup_{i \in S} \left(M^i \cap M^j\right)\right| \\
&\geq \left|M^j\right| - \sum_{i \in S} \left|\left(M^i \cap M^j\right)\right| && (17.1) \\
&\geq w_{\min} - |S| \cdot a_{\max} && (17.2) \\
&\geq w_{\min} - d \cdot a_{\max} && (17.3) \\
&\geq w_{\min} - \frac{w_{\min}-1}{a_{\max}} \cdot a_{\max} && (17.4) \\
&= 1.
\end{aligned}$$

In the above, (17.1) follows from the fact that size of the union of sets is at most the sum of their sizes. (17.2) follows from the definitions of $w_{\min}$ and $a_{\max}$. (17.3) follows from the fact that $|S| \leq d$ while (17.4) follows from the definition of d. The proof is complete. □

Next, we present a simple way to convert a code into a matrix. Let $C \subseteq [q]^t$ be a code such that $C = \{\mathbf{c}_1, \ldots, \mathbf{c}_N\}$. Then consider the matrix M_C whose i'th column is $\mathbf{c}_i$, i.e.

$$M_C = \begin{bmatrix} \uparrow & \uparrow & & \uparrow \\ \mathbf{c}_1 & \mathbf{c}_2 & \cdots & \mathbf{c}_n \\ \downarrow & \downarrow & & \downarrow \end{bmatrix}.$$

Thus, by Lemma 17.4.1, to construct an $\left\lfloor \frac{w_{\min}-1}{a_{\max}} \right\rfloor$-disjunct matrix, it is enough to design a binary code $C^* \subseteq \{0,1\}^t$ such that (i) for every $\mathbf{c} \in C^*$, $wt(\mathbf{c}) \geq w_{\min}$ and (ii) for every $\mathbf{c}^1 \neq \mathbf{c}^2 \in C^*$, we have $|\{i | c_i^1 = c_i^2 = 1\}| \leq a_{\max}$. Next, we look at the construction of such a code.

17.4.1 Kautz-Singleton Construction

In this section, we will prove the following result:

Theorem 17.4.2. *For every integer $d \geq 1$ and large enough $N \geq d$, there exists a $t \times N$ matrix is d-disjunct where $t = O\left(d^2 \left(\log_d N\right)^2\right)$.*

Note that the above result answers Question 17.3.3. It turns out that one can do a bit better: see Exercise 17.4.

Towards this end, we will now study a construction of C^* as in the previous section due to Kautz and Singleton. As we have already seen in Chapter 10, concatenated codes are a way to design binary codes. For our construction of C^*, we will also use code concatenation. In particular, we will pick $C^* = C_{\text{out}} \circ C_{\text{in}}$, where C_{out} is a $[q, k, q-k+1]_q$ Reed-Solomon code (see Chapter 5) while the inner code $C_{\text{in}} : \mathbb{F}_q \to \{0,1\}^q$ is defined as follows. For any $i \in \mathbb{F}_q$, define $C_{\text{in}}(i) = \mathbf{e}_i$. Note that $M_{C_{\text{in}}}$ is the identity matrix and that $N = q^k$ and $t = q^2$.

Example 17.4.3. *Let $k = 1$ and $q = 3$. Note that by our choice of $[3,1]_3$ Reed-Solomon codes, we have $C_{\text{out}} = \{(0,0,0), (1,1,1), (2,2,2)\}$. In other words,*

$$M_{C_{\text{out}}} = \begin{bmatrix} 0 & 1 & 2 \\ 0 & 1 & 2 \\ 0 & 1 & 2 \end{bmatrix}.$$

Then the construction of M_{C^*} can be visualized as in Figure 17.4.3.

Next, we instantiate parameters in the Kautz-Singleton construction to prove Theorem 17.4.2.

Proof of Theorem 17.4.2. We first analyze the construction to determine the parameters $w_{\min}$ and $a_{\max}$. Then we pick the parameters q and k in terms of d to complete the proof.

Recall that $N = q^k$ and $t = q^2$. It is easy to check that every column of M_{C^*} has exactly q ones in it. In other words, $w_{\min} = q$. Next, we estimate $a_{\max}$.

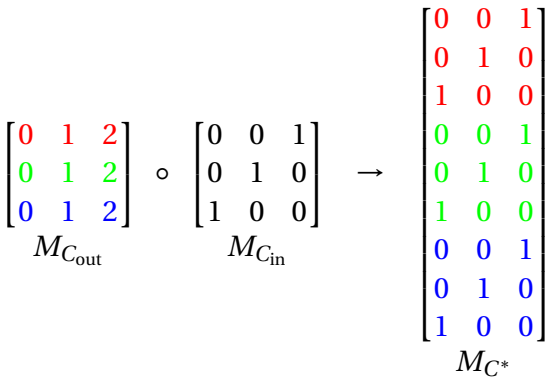


Figure 17.2: Construction of the final matrix M_{C^*} from $M_{C_{\text{out}}}$ and $M_{C_{\text{in}}}$ from Example 17.4.3. The rows in M_{C^*} that correspond to the same row in $M_{C_{\text{out}}}$ have the same color.

Divide the rows into q sized chunks, and index the $t = q^2$ rows by pairs in $[q] \times [q]$. Recall that each column in M_{C^*} corresponds to a codeword in C^*. For notational convenience, we will use M for M_{C^*}. Note that for any row $(i, j) \in [q] \times [q]$ and a column index $\ell \in [N]$, we have $M_{(i,j),\ell} = 1$ if and only if $\mathbf{c}_\ell(j) = j$ (where we use some fixed bijection between $\mathbb{F}_q$ and $[q]$ and $\mathbf{c}_\ell$ is the ℓ'th codeword in C_{out}). In other words, the number of rows where the ℓ_1th and ℓ_2th columns both have a one is exactly the number of positions where $\mathbf{c}_{\ell_1}$ and $\mathbf{c}_{\ell_2}$ agree, which is exactly $q - \Delta(\mathbf{c}_{\ell_1}, \mathbf{c}_{\ell_2})$. Since C_{out} is a $[q, k, q-k+1]_q$ code, the number of rows where any two columns agree is at most $k-1$. In other words, $a_{\max} = k-1$.[1]

Lemma 17.4.1 implies that M_{C^*} is d-disjunct if we pick

$$d = \left\lfloor \frac{q-1}{k-1} \right\rfloor.$$

Thus, we pick q and k such that the above is satisfied. Note that we have $q = O(kd)$. Further, since we have $N = q^k$, we have

$$k = \log_q N.$$

This implies that $q = O(d \cdot \log_q N)$, or $q \log q = O(d \log N)$. In other words we have

$$q = O(d \log_d N).$$

Recalling that $t = q^2$ completes the proof. □

An inspection of the proof of Theorem 17.4.2 shows that we only used the distance of the Reed-Solomon code and in particular, any C_{out} with large enough distance suffices. In particular, if we pick C_{out} to be a random code over an appropriate sized alphabet then one can obtain $t = O(d^2 \log N)$. (See Exercise 17.5 for more on this.) Note that this bound is incomparable to the bound in Theorem 17.4.2. It turns out that these two are the best known upper bounds on $t(d, N)$. In particular,

[1] The equality is obtained due to columns that corresponds to codewords that agree in exactly $k-1$ positions.

Open Question 17.4.1. *Can we beat the upper bound of* $O\left(d^2 \cdot \min\left(\log(N/d), \log_d^2 N\right)\right)$ *on* $t(d,N)$*?*

It turns out that the quadratic dependence on d in the upper bounds is tight. In fact it is known that $t(d,N) \geq \Omega(d^2 \log_d N)$. (See Exercises 17.7 and 17.8.)

Next, we present an application of group testing in the field of data stream algorithms, which in turn will bring out another facet of the connection between coding theory and group testing.

17.5 An Application in Data Stream Algorithms

Let us consider the problem of tracking updates on stock trades. Given a set of trades $(i_1, u_1), \cdots, (i_m, u_m)$, where i_j is the stock id for the j^{th} trade, u_j is the amount of the stocks in the j^{th} trade. The problem is to keep track of the top d stocks. Such a problem is also called hot items/ heavy hitters problem.

Let N be the total number of stocks in the market. This problem could be solved in $O(m) + O(N \log N) \approx O(m)$ time and $O(N)$ space by setting a $O(N)$ size buffer to record the total number of trading for each stock and then sort the buffer later. However, m could be of the order of millions for one minute's trading, e.g. in the first minute of April 19, 2010, there are 8077600 stocks were traded. Taking the huge amount of trades into consideration, such an algorithm is not practical.

A more practical model of efficient algorithms in this context is one of *data stream algorithm*, which is defined as follows.

Definition 17.5.1. A data stream algorithm has four requirements listed below:

1. The algorithm should make one sequential pass over the input.
2. The algorithm should use poly-log space. (In particular, it cannot store the entire input.)
3. The algorithm should have poly-log update time, i.e. whenever a new item appears, the algorithm should be able to update its internal state in poly-log time.
4. The algorithm should have poly-log reporting time, i.e. at any point of time the algorithm should be able to return the required answers in poly-log time.

Thus, ideally we would like to design a data stream algorithm to solve the hot items problem that we discussed earlier. Next, we formally define the hot items problem. We begin with the definition of frequencies of each stock:

Definition 17.5.2. Let f_ℓ denote the total count for the stock id ℓ. Initially $f_\ell = 0$, given (ℓ, u_ℓ), $f_\ell \leftarrow f_\ell + u_\ell$.

Next, we define the hot items problem.

Definition 17.5.3 (Hot Items Problem)**.** Given N different items, for m input pairs of data (i_ℓ, u_ℓ) for $1 \le \ell \le m$, where $i_\ell \in [N]$ indicates the item index and u_ℓ indicates corresponding count. The problem requires updating the count $f_\ell (1 \le \ell \le m)$ for each item, and to output all item indices j such that $f_j > \frac{\sum_{\ell=1}^{N} u_\ell}{d}$. (Any such item is called a *hot item*.)

Note that there can be at most d hot items. In this chapter, we will mostly think of d as $O(\log N)$. Hot items problem is also called heavy hitters problems. We state the result below without proof:

Theorem 17.5.1. *Computing hot items exactly by a deterministic one pass algorithm needs* $\Omega(n)$ *space (even with exponential time).*

This theorem means that we cannot solve the hot items problem in poly-log space as we want. However, we could try to find solutions for problems around this. The first one is to output an approximate solution, which will output a set that contains all hot items and some non-hot items. For this solution, we want to make sure that the size of the output set is not too large (e.g. outputting $[N]$ is not a sensible solution).

Another solution is to make some assumptions on the input. For example, we can assume Zipf-like distribution of the input data, which means only a few items appear frequently. More specifically, we can assume *heavy-tail distribution* on the input data, i.e.:

$$\sum_{\ell:\text{not hot}} f_\ell \le \frac{m}{d}. \tag{17.5}$$

This is reasonable for many applications, such as hot stock finding, where only a few of them have large frequency. Next, we will explore the connection between group testing and hot items problem based on this assumption.

17.5.1 Connection to Group Testing

Let us recall the naive solution that does not lead to a data stream algorithm: for each item $j \in [N]$, we maintain the actual count of number of trades for stock j. In other words, at any point of time, if C_j is the count for stock j, we have $C_j = f_j$. Another way to represent this is if M is the $N \times N$ identity matrix, then we maintain the vector of counts via $M \cdot \mathbf{f}$, where $\mathbf{f}$ is the vector of the frequencies of the items. Paralleling the story in group testing where we replace the identity matrix with a matrix with fewer rows, a natural idea here would be to replace M by matrix with fewer rows that utilizes the fact that there can be at most d hot items. Next, we show that this idea works if the heavy tail distribution holds. In particular, we will reduce the hot items problem to the group testing problem.

We now show how we solve the hot items problem from Definition 17.5.3. Let M be an $t \times N$ matrix that is d-disjunct. We maintain counters $C_1, \ldots, C_t$, where each C_i is the total count of any item that is present in the ith row. We also maintain the total number of items m seen so far. Algorithm 30 and Algorithm 31 present the initialization and update algorithms. The reporting algorithm then needs to solve the following problem: at any point of time, given the counts $C_1, \ldots, C_t$ and m output the at most d hot items.

Algorithm 30 Initialization

OUTPUT: Initialize the counters

1: $m \leftarrow 0$
2: FOR every $j \in [t]$ DO
3: $\quad C_j \leftarrow 0$

Algorithm 31 Update

INPUT: Input pair (i, u), $i \in [N]$ and $u \in \mathbb{Z}$
OUTPUT: Update the Counters

1: $m \leftarrow m + 1$,
2: FOR every $j \in [t]$ DO
3: $\quad$ IF $M_{ij} = 1$ THEN
4: $\quad\quad C_j \leftarrow C_j + u$

Next, we reduce the problem of reporting hot items to the decoding problem of group testing. The reduction essentially follows from the following observations.

Observation 17.5.2. *If j is a hot item and $M_{ij} = 1$, then $C_i > \frac{m}{d}$.*

Proof. Let $i \in [t]$ be such that $M_{ij} = 1$. Then note that at any point of time,

$$C_i = \sum_{k:M_{ik}=1} f_k \geq f_j.^2$$

Since j is a hot item, we have $f_j > \frac{m}{d}$, which completes the proof. □

Observation 17.5.3. *For any $1 \leq i \leq t$, if all j with $M_{ij} = 1$ are not hot items, then we have $C_i \leq \frac{m}{d}$.*

Proof. Fix an $\in [t]$ such that every $j \in [N]$ such that $M_{ij} = 1$ is not a hot item. Then by the same argument as in proof of Observation 17.5.2, we have

$$C_i = \sum_{k:M_{ik}=1} f_k.$$

The proof then follows by the choice of i and (17.5). □

Armed with these observations, we now present the reduction. Define $\mathbf{x} = (x_1, x_2, \ldots, x_N) \in \{0,1\}^N$ with $x_j = 1$ if and only if j is a hot item, and $\mathbf{r} = (r_1, r_2, \ldots, r_t) \in \{0,1\}^t$ with $r_i = 1$ if and

[2]The equality follows e.g. by applying induction on Algorithm 31.

only if $C_i > \frac{m}{d}$, we will have $r_i = \vee_{j:M_{ij}=1} x_j$. The latter claim follows from Observations 17.5.2 and 17.5.3 above. This means we have:

$$M \odot \mathbf{x} = \mathbf{r}. \tag{17.6}$$

Note that by definition, $wt(\mathbf{x}) < d$. Thus reporting the hot items is the same as decoding to compute $\mathbf{x}$ given M and $\mathbf{r}$, which successfully changes the hot items problem into group testing problem. Algorithm 32 has the formal specification of this algorithm.

Algorithm 32 Report Heavy Items

INPUT: Counters m and $C_1, \ldots, C_t$
OUTPUT: Output the heavy items

1: FOR every $j \in [t]$ DO
2: IF $C_t > \frac{m}{d}$ THEN
3: $r_j \leftarrow 1$
4: ELSE
5: $r_j \leftarrow 0$
6: Let $\mathbf{x}$ be the result of decoding (for group testing) $\mathbf{r}$
7: RETURN $\{i | x_i = 1\}$

Next, we will design and analyze the algorithm above and check if the conditions in Definition 17.5.1 are met.

Analysis of the Algorithm

In this part, we will review the requirements on data stream algorithm one by one and check if the algorithm for the hot items problem based on group testing satisfies them. In particular, we will need to pick M and the decoding algorithm. We will pick M to be the d-disjunct matrix from Theorem 17.4.2.

1. *One-pass requirement*
   If we use non-adaptive group testing, the algorithm for the hot items problem above can be implemented in one pass, which means each input is visited only once. (If adaptive group testing is used, the algorithm is no longer one pass, therefore we choose non-adaptive group testing.) We note that by definition, our choice of M satisfies this condition.

2. *Poly-log space requirement*
   In the algorithm, we have to maintain the counters C_i and m. The maximum value for them is mN, thus we can represent each counter in $O(\log N + \log m)$ bits. This means we need $O((\log N + \log m)\, t)$bits to maintain the counters. Theorem 17.4.2 implies that $t = O(d^2 \log_d^2 N)$. Thus, the total space we need to maintain the counters is $O(d^2 \log_d^2 N(\log N + \log m))$.

On the other hand, if we need to store the matrix M, we will need $\Omega(tN)$ space. Therefore, poly-log space requirement can be achieved only if matrix M is not stored directly. (We will tackle this issues in the next point.)

3. *Poly-log update time*
   As mentioned in the previous part, we cannot store the matrix M directly in order to have poly-log space. Since RS code is strongly explicit (see Exercise 6.9), we do not need to explicitly store M (we just need to store the parameters of the code C_{out} and C_{in}, which can be done in poly-log space). In the following, we will argue that the runtime of Algorithm 31 is $O(t \times \text{polylog}\, t)$. It is easy to check the claimed time bound is correct as long as we can perform the check in Step 3 in $\text{polylog}(t)$ time. In particular, we would be done if given $j \in [N]$, we can compute the column M^j in $O(t \times \text{polylog}\, t)$ time. Next, we argue that the latter claim is correct.

   Recall that $M = M_{C^*}$, with $C^* = C_{\text{out}} \circ C_{\text{in}}$, where C_{out} is a $\left[q, k, q-k+1\right]_q$ RS code and C_{in} chosen such that $M_{C_{\text{in}}}$ is the $q \times q$ identity matrix. Recall that codewords of C^* are columns of the matrix M, and we have $n = q^k$, $t = q^2$.

   Since every column of M corresponds to a codeword of C^*, we can think of j equivalently as a message $\mathbf{m} \in {\mathbb{F}_q}^k$. In particular, M^j then corresponds to the codeword $C_{\text{out}}(\mathbf{m})$. On the other hand, the column M^j can be partitioned into q chunks, each chunk is of length q. Notice that $(C_{\text{out}}(\mathbf{m}))_{i_1} = i_2$ if and only if the i_1th chunk has 1 on its i_2th position and 0 on other positions (recall the definition of C_{in}). Therefore, we can compute M^j by computing $C_{\text{out}}(\mathbf{m})$. Because C_{out} is a linear code, $C_{\text{out}}(\mathbf{m})$ can be computed in $O(q^2 \times \text{polylog}\, q)$ time,[3] implies that M^j can be computed in $O(q^2 \times \text{polylog}\, q)$ time. Since we have $t = q^2$, the update process can be finished with $O(t \times \text{polylog}\, t)$ time. (We do not need C_{out} to be strongly explicit: as long as C_{out} is linear the arguments so far work just as well.)

4. *Reporting time*
   It is easy to check that the run time of Algorithm 32 is dominated by Step 6. So far, the only decoding algorithm for M that we have seen is Algorithm 29, which runs in time $\Omega(tN)$, which does not satisfy the required reporting time requirement. In Exercise 17.11, we show that using the fact that C_{out} is the Reed-Solomon code, one can solve the decoding problem in $\text{poly}(t)$.

Thus, we have argued that

Theorem 17.5.4. *There exists a data streaming algorithm that computes d hot items with one pass, $O(t \log N)$ space for $t = O(d^2 \log_d^2 N)$, $O(t\,\text{polylog}\, t)$ update time and $\text{poly}(t)$ reporting time.*

[3] This follows from Proposition 2.3.2 and the fact that C_{out} is strongly explicit

17.6 Summary of best known bounds

We conclude the chapter by collecting the best known bounds on both adaptive and non-adaptive group testing. First, we know the correct bound on the best possible number of adaptive tests:

Theorem 17.6.1.

$$t^a(d,N) = \Theta\left(d\log(N/d)\right).$$

The upper bound follows from Exercise 17.1 while the lower bound follows from Proposition 17.2.1.

There is a gap between the best known upper and lower bound on the number of non-adaptive tests:

Theorem 17.6.2.

$$\Omega\left(d^2\log_d N\right) \le t(d,N) \le O\left(d^2\min\left(\log(N/d),\log_d^2 N\right)\right).$$

The upper bounds follow from Theorem 17.4.2 and Exercise 17.5 while the lower bound follows from Exercise 17.8.

Finally, note that Theorem 17.6.1 and 17.6.2 imply that there is a gap between the minimum number of tests needed for adaptive and non-adaptive group testing:

Corollary 17.6.3.

$$\frac{t(d,N)}{t^a(d,N)} \ge \Omega\left(\frac{d}{\log d}\right).$$

17.7 Exercises

Exercise 17.1 (Upper bound on $t^a(d,N)$). In this problem we will show that $t^a(d,N) = O(d\log(N/d))$. We begin by trying to prove a weaker bound of $O(d\log N)$:

- Show that one can identify at least one i such that $x_i = 1$ (or report none exist) with $O(\log N)$ adaptive tests.

  (*Hint:* Use binary search.)

- Using the scheme above argue that one can compute $\mathbf{x}$ with $O(wt(\mathbf{x})\cdot\log N)$ adaptive tests. Conclude that $t^a(d,N) \le O(d\log N)$.

Next we argue that we can tighten the bound to the optimal bound of $O(d\log(N/d))$:

- Argue that any scheme that computes $\mathbf{x} \in \{0,1\}^N$ with $O(wt(\mathbf{x})\cdot\log N)$ adaptive tests can be used to compute $\mathbf{x}$ with $O(d\log(N/d))$ adaptive tests where $wt(\mathbf{x}) \le d$.

- Conclude that $t^a(d,N) \le O(d\log(N/d))$.

Exercise 17.2. Show that every d-separable matrix is also $(d-1)$-disjunct.

Exercise 17.3. Prove that Algorithm 29 is correct and runs in time $O(tN)$.

Exercise 17.4. For every integer $d \geq 1$ and large enough integer $N \geq d$ show that there exists a d-disjunct matrix with $O(d^2 \log(N/d))$ rows.

(*Hint:* Use the probabilistic method. It might help to pick each of tN bits in the matrix independently at random with the same probability.)

Exercise 17.5. We first begin by generalizing the argument of Theorem 17.4.2:

- Let C_{out} be an $(n,k,D)_q$ code. Let C_{in} be defined such that $M_{C_{\text{in}}}$ is the $q \times q$ identity matrix. Let $M_{C_{\text{out}} \circ C_{\text{in}}}$ be a $t \times N$ matrix that is d-disjunct. Derive the parameters d, t and N.

Next argue that it is enough to pick an outer random code to obtain a d-disjunct matrix with the same parameters obtained in Exercise 17.4:

- Pick $q = \Theta(d)$. Then using the previous part or otherwise show that if C_{out} is a random $[n,k,D]_q$ code, then the resulting $t \times N$ matrix $M_{C_{\text{out}} \circ C_{\text{in}}}$ is d-disjunct with $t = O(d^2 \log N)$ for large enough N.

  (*Hint:* Use Theorem 4.2.1 and Proposition 3.3.5.)

Exercise 17.6. For every integer $d \geq 1$ and large enough $N \geq d$, construct a d-disjunct matrix with $O(d^2 \log N)$ rows in (deterministic) time poly(N).

Hint: Recall Exercise 4.7.

Exercise 17.7 (Lower Bound on $t(d,N)$ due to Bassalygo). In this problem we will show that $t(d,N) \geq \min\left\{\binom{d+2}{2}, N\right\}$. In what follows let M be a $t \times N$ matrix that is d-disjunct.

(a) Argue that if $wt(M^j) < d$ then M^j has a *private row* i.e. there exists a row $i \in [t]$ such that $M_{ij} = 1$ but $M_{ij'} = 0$ for every $j' \neq j$.

(b) Using part (a) or otherwise, argue that if all columns of M have Hamming weight at most $d-1$, then $t \geq N$.

(c) Let M^{-j} for $j \in [N]$ be the matrix M with M^j as well as all rows $i \in [t]$ such that $M_{ij} = 1$ removed. Then argue that M^{-j} is $(d-1)$-disjunct.

(d) Argue that $t(1,N) \geq \min\{3, N\}$.

(e) Using induction with parts (b)-(d) or otherwise, argue that $t \geq \min\left\{\binom{d+2}{2}, N\right\}$.

Exercise 17.8 (Lower Bound on $t(d,N)$ due to Ruszinkó and Alon-Asodi). In this problem, we will show that

$$t(d,N) \geq \Omega\left(\min\left\{d^2 \log_d N, N\right\}\right). \tag{17.7}$$

In what follows let M be a $t \times N$ matrix that is d-disjunct.

(a) Argue that any $j \in [N]$ such that $wt(M^j) < \frac{2t}{d}$ has a *private subset* of size $\lceil 4t/d^2 \rceil$, i.e. there exists a subset $S \subseteq [N]$ with $|S| = \lceil 4t/d^2 \rceil$ such that M^j has ones in all $i \in S$ but for every $j \neq j'$, $M^{j'}$ has at least one row $i' \in S$ such that $M_{i'j'} = 0$.

(b) Using part (a) or otherwise, argue:

$$N - \frac{d}{2} \leq \binom{t}{\lceil 4t/d^2 \rceil}.$$

(c) Using Exercise 17.7 and part (b) or otherwise argue (17.7).

Exercise 17.9. In this exercise and the ones that follow it, we will consider the following equivalent version of the decoding problem: given $\mathbf{r} = M \odot \mathbf{x}$ with $wt(\mathbf{x}) \leq d$, output $\{i | x_i = 1\}$. Now consider the following easier version of the problem. In addition to $\mathbf{r}$ and M assume that one is also given a set S such that $\{i | x_i = 1\} \subseteq S$. Modify Algorithm 29 to design a decoding algorithm that computes $\{i | x_i = 1\}$ in time $O(t \cdot |S|)$.

Exercise 17.10. A $t \times N$ matrix M is called (d, L)-list disjunct if and only if the following holds for every disjoint subsets $S, T \subset [N]$ such that $|S| = d$ and $|T| = L - d$, there is a row in M where all columns in S have a 0 but at least one column in T has a 1.

- What is a $(d, d+1)$-list disjunct matrix?
- Let C_{out} be an $(n,k)_q$ code that is $(0, d, L)$-list recoverable code (recall Definition 13.3.2). Let C_{in} be the inner code such that $M_{C_{\text{in}}}$ is the $q \times q$ identity matrix. Argue that $M_{C_{\text{out}} \circ C_{\text{in}}}$ is (d, L) list disjunct.

Exercise 17.11. Using Exercises 17.9 and 17.10 or otherwise prove the following. Let M_{C^*} be the Kautz-Singleton matrix from Section 17.4.1. Then given $M_{C^*} \odot \mathbf{x}$ with $wt(\mathbf{x}) \leq d$, one can compute $\{i | x_i = 1\}$ in poly(t) time.
(*Hint:* Theorem 13.3.2 could be useful.)

17.8 Bibliographic Notes

Robert Dorfman's paper in 1943 [13] introduced the field of (Combinatorial) Group Testing. It must be noted that though this book covers group testing as an application of coding theory, it took off long before coding theory itself.

The original motivation arose during the Second World War when the United States Public Health service and the Selective Service embarked upon a large scale project. The objective was to weed out all syphilitic men called up for induction. [13].

The connection between group testing and hot items problem considered in Section 17.5 was established by Cormode and Muthukrishnan [12]. More details on data stream algorithms can be bound in the survey by Muthukrishnan [57].

Bibliography

[1] Noga Alon and Joel Spencer. *The Probabilistic Method*. John Wiley, 1992.

[2] P.G.H. Bachmann. *Die analytische Zahlentheorie*. Number v. 2 in Zahlentheorie. Versuch einer Gesammtdarstellung dieser Wissenschaft in ihren Haupttheilen. 2. th. Teubner, 1894.

[3] E. R. Berlekamp. Factoring polynomials over large finite fields. *Mathematics of Computation*, 24:713–735, 1970.

[4] Kristian Brander. *Interpolation and list decoding of algebraic codes*. PhD thesis, Technical University of Denmark, 2010.

[5] P.S. Bullen. *Handbook of Means and Their Inequalities*. Mathematics and Its Applications. Springer Netherlands, 2010.

[6] Larry Carter and Mark N. Wegman. Universal classes of hash functions. *J. Comput. Syst. Sci.*, 18(2):143–154, 1979.

[7] Donald G. Chandler, Eric P. Batterman, and Govind Shah. Hexagonal, information encoding article, process and system. *US Patent Number 4,874,936*, October 1989.

[8] C. L. Chen and M. Y. Hsiao. Error-correcting codes for semiconductor memory applications: A state-of-the-art review. *IBM Journal of Research and Development*, 28(2):124–134, 1984.

[9] Peter M. Chen, Edward K. Lee, Garth A. Gibson, Randy H. Katz, and David A. Patterson. RAID: High-performance, reliable secondary storage. *ACM Computing Surveys*, 26(2):145–185, 1994.

[10] Alan Cobham. The Intrinsic Computational Difficulty of Functions. In Y. Bar-Hillel, editor, *Logic, Methodology and Philosophy of Science, proceedings of the second International Congress, held in Jerusalem, 1964*, Amsterdam, 1965. North-Holland.

[11] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 151–158, 1971.

[12] Graham Cormode and S. Muthukrishnan. What's hot and what's not: tracking most frequent items dynamically. *ACM Trans. Database Syst.*, 30(1):249–278, 2005.

[13] Robert Dorfman. The detection of defective members of large populations. *The Annals of Mathematical Statistics*, 14(4):436–440, December 1943.

[14] Devdatt P. Dubhashi and Alessandro Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, 2009.

[15] Ilya Dumer, Daniele Micciancio, and Madhu Sudan. Hardness of approximating the minimum distance of a linear code. *IEEE Transactions on Information Theory*, 49(1):22–37, 2003.

[16] Zeev Dvir and Shachar Lovett. Subspace evasive sets. *Electronic Colloquium on Computational Complexity (ECCC)*, 18:139, 2011.

[17] Jack Edmonds. Paths, trees, and flowers. In Ira Gessel and Gian-Carlo Rota, editors, *Classic Papers in Combinatorics*, Modern BirkhÃd'user Classics, pages 361–379. BirkhÃd'user Boston, 1987.

[18] Peter Elias. Error-free coding. *IEEE Transactions on Information Theory*, 4(4):29–37, 1954.

[19] Peter Elias. List decoding for noisy channels. *Technical Report 335, Research Laboratory of Electronics, MIT*, 1957.

[20] P. Erdös. On extremal problems of graphs and generalized graphs. *Israel Journal of Mathematics*, 2(3):183–190, 1964.

[21] G. David Forney. *Concatenated Codes*. MIT Press, Cambridge, MA, 1966.

[22] G. David Forney. Generalized Minimum Distance decoding. *IEEE Transactions on Information Theory*, 12:125–131, 1966.

[23] Peter Gemmell and Madhu Sudan. Highly resilient correctors for multivariate polynomials. *Information Processing Letters*, 43(4):169–174, 1992.

[24] E. N. Gilbert. A comparison of signalling alphabets. *Bell System Technical Journal*, 31:504–522, 1952.

[25] M. J. E. Golay. Notes on digital coding. *Proceedings of the IRE*, 37:657, 1949.

[26] Venkatesan Guruswami. Limits to list decodability of linear codes. In *Proceedings of the 34th ACM Symposium on Theory of Computing (STOC)*, pages 802–811, 2002.

[27] Venkatesan Guruswami. *List decoding of error-correcting codes*. Number 3282 in Lecture Notes in Computer Science. Springer, 2004. (Winning Thesis of the 2002 ACM Doctoral Dissertation Competition).

[28] Venkatesan Guruswami. Linear-algebraic list decoding of folded reed-solomon codes. In *Proceedings of the 26th Annual IEEE Conference on Computational Complexity (CCC)*, pages 77–85, 2011.

[29] Venkatesan Guruswami, Johan Håstad, and Swastik Kopparty. On the list-decodability of random linear codes. *IEEE Transactions on Information Theory*, 57(2):718–725, 2011.

[30] Venkatesan Guruswami, Johan Håstad, Madhu Sudan, and David Zuckerman. Combinatorial bounds for list decoding. *IEEE Transactions on Information Theory*, 48(5):1021–1035, 2002.

[31] Venkatesan Guruswami and Piotr Indyk. Linear-time encodable/decodable codes with near-optimal rate. *IEEE Transactions on Information Theory*, 51(10):3393–3400, 2005.

[32] Venkatesan Guruswami and Atri Rudra. Limits to list decoding reed-solomon codes. *IEEE Transactions on Information Theory*, 52(8):3642–3649, August 2006.

[33] Venkatesan Guruswami and Atri Rudra. Explicit codes achieving list decoding capacity: Error-correction with optimal redundancy. *IEEE Transactions on Information Theory*, 54(1):135–150, 2008.

[34] Venkatesan Guruswami and Atri Rudra. Better binary list decodable codes via multilevel concatenation. *IEEE Transactions on Information Theory*, 55(1):19–26, 2009.

[35] Venkatesan Guruswami and Atri Rudra. The existence of concatenated codes list-decodable up to the hamming bound. *IEEE Transactions on Information Theory*, 56(10):5195–5206, 2010.

[36] Venkatesan Guruswami and Igor Shparlinski. Unconditional proof of tightness of johnson bound. In *Proceedgins of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 754–755, 2003.

[37] Venkatesan Guruswami and Madhu Sudan. Improved decoding of Reed-Solomon and algebraic-geometry codes. *IEEE Transactions on Information Theory*, 45(6):1757–1767, 1999.

[38] Venkatesan Guruswami and Chaoping Xing. Folded codes from function field towers and improved optimal rate list decoding. *Electronic Colloquium on Computational Complexity (ECCC)*, 19:36, 2012.

[39] Richard W. Hamming. Error Detecting and Error Correcting Codes. *Bell System Technical Journal*, 29:147–160, April 1950.

[40] G.H. Hardy and J.E. Littlewood. Some problems of diophantine approximation. *Acta Mathematica*, 37(1):193–239, 1914.

[41] Johan Håstad, Steven Phillips, and Shmuel Safra. A well-characterized approximation problem. *Inf. Process. Lett.*, 47(6):301–305, 1993.

[42] Tom Høholdt, J. H. van Lint, and Ruud Pellikaan. Algebraic geometry codes. In W. C. Huffamn V. S. Pless and R. A.Brualdi, editors, *Handbook of Coding Theory*. North Holland, 1998.

[43] Ari Juels and Madhu Sudan. A fuzzy vault scheme. *Des. Codes Cryptography*, 38(2):237–257, 2006.

[44] J. Justesen. Class of constructive asymptotically good algebraic codes. *IEEE Trans. Inform. Theory*, pages 652–656, Sep 1972.

[45] Erich Kaltofen. Polynomial-time reductions from multivariate to bi- and univariate integral polynomial factorization. *SIAM J. Comput.*, 14(2):469–489, 1985.

[46] Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103, 1972.

[47] Donald E. Knuth. Big omicron and big omega and big theta. *SIGACT News*, 8(2):18–24, April 1976.

[48] Andrei N. Kolmogorov. Three Approaches to the Quantitative Definition of Information. *Problems of Information Transmission*, 1(1):1–7, 1965.

[49] E. Landau. *Handbuch der lehre von der verteilung der primzahlen*. Number v. 1 in Handbuch der lehre von der verteilung der primzahlen. B. G. Teubner, 1909.

[50] Amos Lapidoth and P. Narayan. Reliable communication under channel uncertainty. *IEEE Transactions on Information Theory*, 44(6):2148–2177, 1998.

[51] Leonid A Levin. Universal sequential search problems. *Problemy Peredachi Informatsii*, 9(3):115–116, 1973.

[52] Ming Li and Paul M. B. Vitányi. *An Introduction to Kolmogorov Complexity and Its Applications*. Graduate Texts in Computer Science. Springer, New York, NY, USA, third edition, 2008.

[53] Rudolf Lidl and Harald Niederreiter. *Introduction to Finite Fields and their applications*. Cambridge University Press, Cambridge, MA, 1986.

[54] Michael Luby, Michael Mitzenmacher, Mohammad Amin Shokrollahi, and Daniel A. Spielman. Efficient erasure correcting codes. *IEEE Transactions on Information Theory*, 47(2):569–584, 2001.

[55] Robert J. McEliece. On the average list size for the Guruswami-Sudan decoder. In *7th International Symposium on Communications Theory and Applications (ISCTA)*, July 2003.

[56] 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.

[57] S. Muthukrishnan. Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, 1(2), 2005.

[58] Farzad Parvaresh and Alexander Vardy. Correcting errors beyond the guruswami-sudan radius in polynomial time. In *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 285–294, 2005.

[59] Larry L. Peterson and Bruce S. Davis. *Computer Networks: A Systems Approach*. Morgan Kaufmann Publishers, San Francisco, 1996.

[60] W. Wesley Peterson. Encoding and error-correction procedures for Bose-Chaudhuri codes. *IEEE Transactions on Information Theory*, 6:459–470, 1960.

[61] Michael O. Rabin. Probailistic algorithms. In J. F. Traub, editor, *Algorithms and Complexity, Recent Results and New Directions*, pages 21–39, 1976.

[62] Irving S. Reed and Gustav Solomon. Polynomial codes over certain finite fields. *SIAM Journal on Applied Mathematics*, 8(2):300–304, 1960.

[63] Herbert Robbins. A remark on Stirling's formula. *Amer. Math. Monthly*, 62:26–29, 1955.

[64] Atri Rudra and Steve Uurtamo. Two theorems on list decoding. In *Proceedings of the 14th Intl. Workshop on Randomization and Computation (RANDOM)*, pages 696–709, 2010.

[65] Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 623–656, 1948.

[66] Victor Shoup. *A computational introduction to number theory and algebra*. Cambridge University Press, 2006.

[67] R. Singleton. Maximum distance q -nary codes. *Information Theory, IEEE Transactions on*, 10(2):116 – 118, apr 1964.

[68] Michael Sipser. The history and status of the p versus np question. In *Proceedings of the Twenty-fourth Annual ACM Symposium on Theory of Computing*, STOC '92, pages 603–618, New York, NY, USA, 1992. ACM.

[69] Daniel A. Spielman. Linear-time encodable and decodable error-correcting codes. *IEEE Transactions on Information Theory*, 42(6):1723–1731, 1996.

[70] Douglas R. Stinson. Universal hashing and authentication codes. *Des. Codes Cryptography*, 4(4):369–380, 1994.

[71] Madhu Sudan. Decoding of Reed Solomon codes beyond the error-correction bound. *J. Complexity*, 13(1):180–193, 1997.

[72] Madhu Sudan. List decoding: Algorithms and applications. *SIGACT News*, 31:16–27, 2000.

[73] Robert Endre Tarjan. Algorithmic design. *Commun. ACM*, 30(3):204–212, 1987.

[74] Aimo Tietavainen. On the nonexistence theorems for perfect error-correcting codes. *SIAM Journal of Applied Mathematics*, 24(1):88–96, 1973.

[75] Jacobus H. van Lint. Nonexistence theorems for perfect error-correcting codes. In *Proceedings of the Symposium on Computers in Algebra and Number Theory*, pages 89–95, 1970.

[76] R. R. Varshamov. Estimate of the number of signals in error correcting codes. *Doklady Akadamii Nauk*, 117:739–741, 1957.

[77] Lloyd R. Welch and Elwyn R. Berlekamp. Error correction of algebraic block codes. *US Patent Number 4,633,470*, December 1986.

[78] John M. Wozencraft. List Decoding. *Quarterly Progress Report, Research Laboratory of Electronics, MIT*, 48:90–95, 1958.

Appendix A

Notation Table

$\mathbb{R}$	The set of real numbers	
$\neg E$	Negation of the event E	
$\log x$	Logarithm to the base 2	
Σ^m	Vectors of length m with symbols from Σ	
$\mathbf{v}$	A vector	
$\mathbf{0}$	The all zero vector	
$\mathbf{e}_i$	The ith standard vector, i.e. 1 in position i and 0 everywhere else	
$\mathbf{v}_S$	Vector $\mathbf{v}$ projected down to indices in S	
$\langle \mathbf{u}, \mathbf{v} \rangle$	Inner-product of vectors $\mathbf{u}$ and $\mathbf{v}$	
$[a, b]$	$\{x \in \mathbb{R} \mid a \leq x \leq b\}$	
$[x]$	The set $\{1, \ldots, x\}$	Secti
n	Block length of a code	Defin
Σ	Alphabet of a code	Defin
q	$q = \lvert\Sigma\rvert$	Defin
k	Dimension of a code	Defin
R	Rate of a code	Defin
$\Delta(\mathbf{u}, \mathbf{v})$	Hamming distance between $\mathbf{u}$ and $\mathbf{v}$	Defin
d	Minimum distance of a code	Defin
$wt(\mathbf{v})$	Hamming weight of $\mathbf{v}$	Defin
$B(\mathbf{x}, r)$	Hamming ball of radius r centered on $\mathbf{x}$	Defin
$Vol_q(r, n)$	Volume Hamming ball of radius r	Defin
$(n, k, d)_\Sigma$	A code with block length n, dimension k, distance d and alphabet Σ	Defin
$(n, k, d)_q$	A code with block length n, dimension k, distance d and alphabet size q	Defin
$[n, k, d]_q$	A linear $(n, k, d)_q$ code	Defin
$\mathbb{F}_q$	The finite field with q elements (q is a prime power)	Secti
$\mathbb{F}^*$	The set of non-zero elements in the field $\mathbb{F}$	
$\mathbb{F}_q^{m \times N}$	The set of all $m \times N$ matrices where each entry is from $\mathbb{F}_q$	
$\mathbb{F}_q[X_1, \ldots, X_m]$	The set of all m-variate polynomials with coefficients from $\mathbb{F}_q$	
$R(C)$	Rate of a code family C	Defin

$\delta(C)$	Relative distance of a code family C	Defin
$\mathscr{U}$	The uniform distribution	Defin
$\mathbb{E}[V]$	Expectation of a random variable V	Defin
$\mathbb{1}_E$	Indicator variable for event E	Secti
$H_q(x)$	$x\log_q(q-1) - x\log_q x - (1-x)\log_q(1-x)$	Defin
$H_q^{-1}(y)$	Unique $x \in [0, 1-1/q]$ such that $H_q(x) = y$	Secti
$\deg(P)$	Degree of polynomial $P(X)$	Defin
$\mathbb{F}_q[X]$	The set of all univariate polynomials in X over $\mathbb{F}_q$	Secti
$J_q(x)$	$(1-1/q)(1-\sqrt{1-qx/(q-1)})$	Theo
$\binom{S}{t}$	$\{T \subseteq S \mid \|T\| = t\}$	
$M \odot \mathbf{x}$	Binary matrix-vector multiplication where multiplication is AND and addition is OR	

Appendix B

Some Useful Facts

B.1 Some Useful Inequalities

We begin with a simple lower bound on the binomial function:

Lemma B.1.1. *For every integers* $1 \le a \le b$*, we have*

$$\binom{b}{a} \ge \left(\frac{b}{a}\right)^a.$$

Proof. The following sequence of relations completes the proof:

$$\binom{a}{b} = \prod_{i=0}^{a-1} \frac{b-i}{a-i} \ge \prod_{i=0}^{a-1} \frac{b}{a} = \left(\frac{b}{a}\right)^a.$$

In the above, the first equality follows from definition and the inequality is true since $b \ge a$ and $i \ge 0$. □

We state the next set of inequalities without proof (see [63] for a proof):

Lemma B.1.2 (Stirling's Approximation)**.** *For every integer* $n \ge 1$*, we have*

$$\sqrt{2\pi n}\left(\frac{n}{e}\right)^n e^{\lambda_1(n)} < n! < \sqrt{2\pi n}\left(\frac{n}{e}\right)^n e^{\lambda_2(n)},$$

where

$$\lambda_1(n) = \frac{1}{12n+1} \text{ and } \lambda_2(n) = \frac{1}{12n}.$$

We prove another inequality involving Binomial coefficient.

Lemma B.1.3. *For every integers* $1 \le a \le b$*, we have*

$$\binom{b}{a} \le \left(\frac{eb}{a}\right)^a.$$

Proof. First note that

$$\binom{a}{b} = \frac{a(a-1)\cdots(a-b+1)}{b!} \leq \frac{a^b}{b!}.$$

The final bound follows from the fact that

$$b! > \left(\frac{b}{e}\right)^b,$$

which in turns follows from the lower bound in Lemma B.1.2. □

We next state Bernoulli's inequality:

Lemma B.1.4 (Bernoulli's Inequality)**.** *For every real numbers $k \geq 1$ and $x \geq -1$, we have*

$$(1+x)^k \geq 1 + kx.$$

Proof Sketch. We only present the proof for integer k. For the full proof see e.g. [5].

For the base case of $k = 1$, the inequality holds trivially. Assume that the inequality holds for some integer $k \geq 1$ and to complete the proof, we will prove it for $k+1$. Now consider the following inequalities:

$$\begin{aligned}(1+x)^{k+1} &= (1+x)\cdot(1+x)^k \\ &\geq (1+x)\cdot(1+kx) \\ &= 1 + (k+1)x + kx^2 \\ &\geq 1 + (k+1)x,\end{aligned}$$

as desired. In the above, the first inequality follows from the inductive hypothesis and the second inequality follows from the fact that $k \geq 1$. □

Lemma B.1.5. *For $|X|le1$,*

$$\sqrt{1+x} \leq 1 + \frac{x}{2} - \frac{x^2}{16}.$$

Proof. Squaring the RHS we get

$$\left(1 + \frac{x}{2} - \frac{x^2}{16}\right)^2 = 1 + \frac{x^2}{4} + \frac{x^4}{256} + x - \frac{x^2}{16} - \frac{x^3}{32} = 1 + x + \frac{3x^2}{16} - \frac{x^3}{32} + \frac{x^4}{256} \geq 1 + x,$$

as desried. □

B.2 Some Useful Identities and Bounds

We start off with an equivalence between two inequalities.

Lemma B.2.1. *Let $a, b, c, d > 0$. Then $\frac{a}{b} \leq \frac{c}{d}$ if and only if $\frac{a}{a+b} \leq \frac{c}{c+d}$.*

Proof. Note that $\frac{a}{b} \le \frac{c}{d}$ if and only if

$$\frac{b}{a} \ge \frac{d}{c}.$$

The above is true if and only if

$$\frac{b}{a} + 1 \ge \frac{d}{c} + 1,$$

which is same as $\frac{a}{a+b} \le \frac{c}{c+d}$. □

Next, we state some infinite sums that are identical to certain logarithms (the proofs are standard and are omitted).

Lemma B.2.2. *For* $|x| < 1$,

$$\ln(1+x) = x - \frac{x^2}{2!} + \frac{x^3}{3!} - \cdots.$$

We can use the above to prove some bounds on $\ln(1+x)$ (we omit the proof):

Lemma B.2.3. *For* $0 \le x < 1$, *we have*

$$x - x^2/2 \le \ln(1+x) \le x,$$

and for $0 \le x \le 1/2$, *we have*

$$-x - x^2 \le \ln(1-x) \le -x.$$

We can use the above bounds to further prove boounds on the (binary) entropy function:

Lemma B.2.4. *For* $x \le 1/4$, *we have*

$$1 - 5x^2 \le H(1/2 - x) \le 1 - x^2.$$

Proof. By definition $H(1/2-x) = 1 - 1/2\log(1-4x^2) + x\log(1-2x)/(1+2x)$, and using the approximations for $\ln(1+x)$ from Lemma B.2.3, we have, for $x < 1/4$,

$$\begin{aligned} H(1/2-x) &\le 1 + \frac{1}{2\ln 2}\cdot(4x^2 + 16x^4) + \frac{1}{\ln 2}\cdot(-2x^2) - \frac{1}{\ln 2}\cdot(2x^2 - 2x^3) \\ &= 1 - \frac{2}{\ln 2}\cdot x^2 + \frac{2}{\ln 2}\cdot x^3 + \frac{8}{\ln 2}\cdot x^4 \\ &\le 1 - \frac{x^2}{\ln 2} \qquad \text{(B.1)} \\ &\le 1 - x^2. \end{aligned}$$

In the above, (B.1) follows by using our assumption that $x \le 1/4$.

Using the other sides of the approximations we also have:

$$\begin{aligned} H(1/2-x) &\ge 1 + \frac{1}{2\ln 2}\cdot(4x^2) + \frac{1}{\ln 2}\cdot(-2x^2 - 4x^3) - \frac{1}{\ln 2}\cdot(2x^2) \\ &\ge 1 - \frac{3x^2}{\ln 2} \\ &\ge 1 - 5x^2, \end{aligned}$$

where the second inequality uses our assumption that $x \le 1/4$. □

The following fact follows from the well-known fact that $\lim_{x\to\infty}(1+1/x)^x = e$:

Lemma B.2.5. *For every real $x > 0$,*

$$\left(1+\frac{1}{x}\right)^x \leq e.$$

Appendix C

Background on Asymptotic notation, Algorithms and Complexity

In this chapter, we collect relevant background on algorithms and their analysis (as well as their limitations). We begin with notation that we will use to bound various quantities when we do not pay close attention to the constants.

C.1 Asymptotic Notation

Throughout the book, we will encounter situations where we would be interested in how a function $f(N)$ grows as the input parameter N grows. (We will assume that the real valued function f is monotone.) The most common such situation is when we would like to bound the runtime of an algorithm we are analyzing– we will consider this situation in some detail shortly. In particular, we will interested in bounds on $f(N)$ that are "oblivious" to constants. E.g. given that an algorithm takes $24N^2 + 100N$ steps to terminate, we would be interested in the fact that the dominating term is the N^2 (for large enough N). Technically, speaking we are interested in the *asymptotic* growth of $f(N)$. Throughout this chapter, we will assume that all functions are monotone.

The first definition is when we are interested in an upper bound on the function $f(N)$. When talking about numbers, we say b is an upper bound on a if $a \leq b$. We will consider a similar definition for functions that in some sense ignores constants.

Definition C.1.1. We say $f(N)$ is $O(g(N))$ (to be read as $f(N)$ is "Big-Oh" of $g(N)$) if there exists constants $c, N_0 \geq 0$ that are independent of N such that for every large enough $N \geq N_0$:

$$f(N) \leq c \cdot g(N).$$

Alternatively $f(N)$ is $O(g(N))$ if and only if

$$\lim_{N \to \infty} \frac{f(N)}{g(N)} \leq C,$$

for some absolute constant C. (See Exercise C.1.) So for example both $24N^2+100N$ and $N^2/2-N$ are $O(N^2)$ as well as $O(N^3)$. However, neither of them are $O(N)$ or $O(N^{3/2})$.

The second definition is when we are interested in a lower bound on the function $f(N)$. When talking about numbers, we say b is a lower bound on a if $a \geq b$. We will consider a similar definition for functions that in some sense ignores constants.

Definition C.1.2. We say $f(N)$ is $\Omega(g(N))$ (to be read as $f(N)$ is "Big-Omega" of $g(N)$) if there exists constants $\varepsilon, N_0 \geq 0$ that are independent of n such that for every large enough $N \geq N_0$:

$$f(N) \geq \varepsilon \cdot g(N).$$

Alternatively $f(N)$ is $\Omega(g(N))$ if and only if

$$\lim_{N\to\infty} \frac{f(N)}{g(N)} \geq C,$$

for some absolute constant C. (See Exercise C.2.) So for example both $24N^2+100N$ and $N^2/2-N$ are $\Omega(N^2)$ as well as $\Omega(N^{3/2})$. However, neither of them are $\Omega(N^3)$ or $\Omega(N^{5/2})$.

The third definition is when we are interested in a tight bound on the function $f(N)$. When talking about numbers, we say b is same as a if $a = b$. We will consider a similar definition for functions that in some sense ignores constants.

Definition C.1.3. We say $f(N)$ is $\Theta(g(N))$ (to be read as $f(N)$ is "Theta" of $g(N)$) if and only if $f(N)$ is $O(g(N))$ and is also $\Omega(g(N))$.

Alternatively $f(N)$ is $\Theta(g(N))$ if and only if

$$\lim_{N\to\infty} \frac{f(N)}{g(N)} = C,$$

for some absolute constant C. (See Exercise C.3.) So for example both $24N^2+100N$ and $N^2/2-N$ are $\Theta(N^2)$. However, neither of them are $\Theta(N^3)$ or $\Theta(N)$.

The fourth definition is when we are interested in a strict upper bound on the function $f(N)$. When talking about numbers, we say b is a strict upper bound on a if $a < b$. We will consider a similar definition for functions that in some sense ignores constants.

Definition C.1.4. We say $f(N)$ is $o(g(N))$ (to be read as $f(N)$ is "little-oh" of $g(N)$) if $f(N)$ is $O(g(N))$ but $f(N)$ is not $\Omega(g(N))$.

Alternatively $f(N)$ is $o(g(N))$ if and only if

$$\lim_{N\to\infty} \frac{f(N)}{g(N)} = 0.$$

(See Exercise C.4.) So for example both $24N^2+100N$ and $N^2/2-N$ are $o(N^3)$ as well as $o(N^{5/2})$. However, neither of them are $o(N^2)$ or $o(N^{3/2})$.

The final definition is when we are interested in a strict lower bound on the function $f(N)$. When talking about numbers, we say b is a strict lower bound on a if $a > b$. We will consider a similar definition for functions that in some sense ignores constants.

Definition C.1.5. We say $f(N)$ is $\omega(g(N))$ (to be read as $f(N)$ is "little-omega" of $g(N)$) if $f(N)$ is $\Omega(g(N))$ but $f(N)$ is not $O(g(N))$.

Alternatively $f(N)$ is $\omega(g(N))$ if and only if

$$\lim_{N\to\infty} \frac{f(N)}{g(N)} = \infty.$$

(See Exercise C.5.) So for example both $24N^2 + 100N$ and $N^2/2 - N$ are $\omega(N)$ as well as $\omega(N^{3/2})$. However, neither of them are $\omega(N^2)$ or $\omega(N^{5/2})$.

C.1.1 Some Properties

We now collect some properties of asymptotic notation that we will be useful in this book.

First all the notations are transitive:

Lemma C.1.1. *Let $\alpha \in \{O, \Omega, \Theta, o, \omega\}$. Then if $f(N)$ is $\alpha(g(N))$ and $g(N)$ is $\alpha(h(N))$, then $f(N)$ is $\alpha(h(N))$.*

Second, all the notations are additive:

Lemma C.1.2. *Let $\alpha \in \{O, \Omega, \Theta, o, \omega\}$. Then if $f(N)$ is $\alpha(h(N))$ and $g(N)$ is $\alpha(h(N))$, then $f(N) + g(N)$ is $\alpha(h(N))$.*

Finally, all the notations are multiplicative:

Lemma C.1.3. *Let $\alpha \in \{O, \Omega, \Theta, o, \omega\}$. Then if $f(N)$ is $\alpha(h_1(N))$ and $g(N)$ is $\alpha(h_2(N))$, then $f(N) \cdot g(N)$ is $\alpha(h_1(N) \cdot h_2(N))$.*

The proofs of the above properties are left as an exercise (see Exercise C.6).

C.2 Bounding Algorithm run time

Let $\mathscr{A}$ be the algorithm we are trying to analyze. Then we will define $T(N)$ to be the worst-case run-time of $\mathscr{A}$ over all inputs of size N. Slightly more formally, let $t_{\mathscr{A}}(\mathbf{x})$ be the number of steps taken by the algorithm $\mathscr{A}$ on input $\mathbf{x}$. Then

$$T(N) = \max_{\mathbf{x}:\mathbf{x} \text{ is of size } N} t_{\mathscr{A}}(\mathbf{x}). \tag{C.1}$$

In this section, we present two useful strategies to prove statements like $T(N)$ is $O(g(N))$ or $T(N)$ is $\Omega(h(N))$. Then we will analyze the run time of a very simple algorithm. However, before that we digress to clarify the following: (i) For most of the book, we will be interested in deterministic algorithms (i.e. algorithm whose execution is fixed given the input). However, we will consider randomized algorithms (see Section C.3 for more on this). (ii) One needs to clarify what constitutes a "step" in the definition of $T(N)$ above. We do so next.

C.2.1 RAM model

In this book, unless specified otherwise we will assume that the algorithms run on the RAM model. Informally, this computation model is defined as follows. For an input with n items, the memory consists of registers with $O(\log n)$ bits. For simplicity, we can assume that the input and output have separate dedicated registers. Note that the input will have n dedicated registers.

Any (atomic) step an algorithm can take are essentially any basic operations on constant such registers which can be implemented in $O(\log n)$ bit operations. In particular, the following operations are considered to take one step: loading $O(\log n)$ from a register or storing $O(\log n)$ bits in a register, initializing the contents of a register, bit-wise operations among registers, e.g. taking bit-wise XOR of the bits of two registers, adding numbers stored in two registers, incrementing the value stored in a register, comparing the values stored in two registers. Some examples of operations that are *not* single step operations: multiplying numbers or exponentiation (where the operands fit into one register each).

C.2.2 Proving $T(N)$ is $O(f(N))$

We start off with an analogy. Say you wanted prove that given m numbers $a_1, \ldots, a_m$, $\max_i a_i \leq U$. Then how would you go about doing so? One way is to argue that the maximum value is attained at i^* and then show that $a_{i^*} \leq U$. Now this is a perfectly valid way to prove the inequality we are after but note that you will *also* have to prove that the maximum value is attained at i^*. Generally, this is a non-trivial task. However, consider the following strategy:

> Show that for *every* $1 \leq i \leq m$, $a_i \leq U$. Then conclude that $\max_i a_i \leq U$.

Let us consider an example to illustrate the two strategies above. Let us say for whatever reason we are interested in showing that the age of the oldest person in your coding theory lectures is at most 100. Assume there are 98 students registered and the instructor is always present in class. This implies that there are at most $m = 99$ folks in the class. Let us order them somehow and let a_i denote the age of the i'th person. Then we want to show that $\max\{a_1, \ldots, a_{99}\} \leq 100$ (i.e. $U = 100$). The first strategy above would be to first figure out who is the oldest person in room: say that is the i^*'th person (where $1 \leq i^* \leq 99$) and then check if $a_{i^*} \leq 100$. However, this strategy is somewhat invasive: e.g. the oldest person might not want to reveal that he or she is the oldest person in the room. This is where the second strategy works better: we ask every person in the room if their age is ≤ 100: i.e. we check if for every $1 \leq i \leq 99$, $a_i \leq 100$. If everyone says yes, then we have proved that $\max_i a_i \leq 100$ (without necessarily revealing the identity of the oldest person).

Mathematically the above two strategies are the same. However, in "practice," using the second strategy turns out to be much easier. (E.g. this was true in the age example above.) Thus, here is the strategy to prove that $T(N)$ is $O(f(N))$:

> For every large enough N, show that for **every** input $\mathbf{x}$ of size N, $t_{\mathscr{A}}(\mathbf{x})$ is $O(f(N))$.
> Then conclude that $T(N)$ is $O(f(N))$.

C.2.3 Proving $T(N)$ is $\Omega(f(N))$

We start off with the same analogy as in the previous section. Say you wanted prove that given m numbers $a_1, \ldots, a_m$, $\max_i a_i \geq L$. Then how would you go about doing so? Again, one way is to argue that the maximum value is attained at i^* and then show that $a_{i^*} \geq L$. Now this is a perfectly valid way to prove the inequality we are after but note that you will *also* have to prove that the maximum value is attained at i^*. Generally, this is a non-trivial task. However, consider the following strategy:

> Show that there *exists* an $1 \leq i \leq m$, such that $a_i \geq L$. Then conclude that $\max_i a_i \geq L$.

Let us go back to the class room example. Now let us say we are interesting in proving that the oldest person in the room is at least 25 years old. (So $a_1, \ldots, a_m$ is as in Section C.2.2 but now $L = 25$.) Again, the first strategy would be to first figure out the oldest person, say i^* and check if $a_{i^*} \geq 25$. However, as we saw in Section C.2.2, this strategy is somewhat invasive. However, consider the the following implementation of the second strategy above. Say for the sake of mathematics, the instructor comes forward and volunteers the information that her age is at least 25. Since the oldest person's age has to be at least the instructor's age, this proves that $\max_i a_i \geq 25$, as desired.

Mathematically the above two strategies are the same. However, in "practice," using the strategy second turns out to be much easier. (E.g., this was true in the age example above.) Thus, here is the strategy to prove that $T(N)$ is $\Omega(f(N))$:

> For every large enough N, show that there **exists** an input $\mathbf{x}$ of size N, $t_{\mathscr{A}}(\mathbf{x})$ is $\Omega(f(N))$. Then conclude that $T(N)$ is $\Omega(f(N))$.

C.2.4 An Example

Now let us use all the strategies from Section C.2.2 and Section C.2.3 to asymptotically bound the run-time of a simple algorithm. Consider the following simple problem: given $n+1$ numbers $a_1, \ldots, a_n; v$, we should output $1 \leq i \leq n$ if $a_i = v$ (if there are multiple such i's then output any one of them) else output -1. Below is a simple algorithm to solve this problem.

Algorithm 33 Simple Search

INPUT: $a_1, \ldots, a_n; v$
OUTPUT: i if $a_i = v$; -1 otherwise

```
1: FOR every 1 ≤ i ≤ n DO
2:     IF a_i = v THEN RETURN i
3: RETURN −1
```

We will show the following:

Theorem C.2.1. *The Simple Search algorithm 33 has a run time of* $\Theta(n)$.

We will prove Theorem C.2.1 by proving Lemmas C.2.2 and C.2.3.

Lemma C.2.2. *$T(n)$ for Algorithm 33 is $O(n)$.*

Proof. We will use the strategy outlined in Section C.2.2. Let $a_1, \ldots, a_n; v$ be an arbitrary input. Then first note that there are at most n iterations of the for loop in Step 1. Further, each iteration of the for loop (i.e. Step 2) can be implemented in $O(1)$ time (since it involves one comparison and a potential return of the output value). Thus, by Lemma C.1.3, the total times taken overall in Steps 1 and 2 is given by

$$T_{12} \leq O(n \cdot 1) = O(n).$$

Further, since Step 3 is a simple return statement, it takes time $T_3 = O(1)$ time. Thus, we have that

$$t_{\text{Algorithm 33}}(a_1, \ldots, a_n; v) = T_{12} + T_3 \leq O(n) + O(1) \leq O(n),$$

where the last inequality follows from Lemma C.1.2 and the fact that $O(1)$ is also $O(n)$. Since the choice of $a_1, \ldots, a_n; v$ was arbitrary, the proof is complete. □

Lemma C.2.3. *$T(n)$ for Algorithm 33 is $\Omega(n)$.*

Proof. We will follow the strategy laid out in Section C.2.3. For every $n \geq 1$, consider the specific input $a'_i = n+1-i$ (for every $1 \leq i \leq n$) and $v' = 1$. For this specific input, it can be easily checked that the condition in Step 2 is only satisfied when $i = n$. In other words, the for loop runs at least (actually exactly) n times. Further, each iteration of this loop (i.e. Step 2) has to perform at least one comparison, which means that this step takes $\Omega(1)$ time. Since n is $\Omega(n)$, by Lemma C.1.3 (using notation from the proof of Lemma C.2.2), we have

$$T_{12} \geq \Omega(n \cdot 1) = \Omega(n).$$

Thus, we have

$$t_{\text{Algorithm 33}}(a'_1, \ldots, a'_n; v') \geq T_{12} \geq \Omega(n).$$

Since we have shown the existence of one input for each $n \geq 1$ for which the run-time is $\Omega(n)$, the proof is complete. □

A quick remark on the proof of Lemma C.2.3. Since by Section C.2.3, we only need to exhibit only *one* input with runtime $\Omega(n)$, the input instance in the proof of Lemma C.2.3 is only one possibility. One can choose other instances: e.g. we can choose an instance where the output has to be -1 (as a specific instance consider $a_i = i$ and $v = 0$). For this instance one can make a similar argument as in the proof of Lemma C.2.3 to show that $T(n) \geq \Omega(n)$.

C.2.5 The Best-Case Input "Trap"

We now briefly talk about a common mistake that is made when one starts trying to prove $\Omega(\cdot)$ on $T(N)$. Note that in Section C.2.3, it says that one can prove that $T(N)$ to be $\Omega(f(N))$ for every large enough N, one only needs to pick *one* input of size N for which the algorithm takes $\Omega(f(N))$ steps.

The confusing part about the strategy in Section C.2.3 is how does one get a hand on that special input that will prove the $\Omega(f(N))$ bound. There is no mechanical way of finding this input. Generally, speaking you have to look at the algorithm and get a feel for what input might force the algorithm to spend a lot of time. Sometimes, the analysis of the $O(\cdot)$ bound itself gives gives us a clue.

However, one way of picking the "special" input that **almost always never works** *in practice* is to consider (for every large enough N), the "best-case input," i.e. an input of size N on which the algorithm runs very fast. Now such an input will give you a valid lower bound but it would almost never give you a *tight* lower bound.

So for example, let us try to prove Lemma C.2.3 using the best case input. Here is one best case input: $a_i = i$ for every $i \in [n]$ and $v = 1$. Note that in this case the algorithm finds a match in the first iteration and this terminates in constant many steps. Thus, this will prove an $\Omega(1)$ lower bound but that is not tight/good enough.

Another common mistake is to make an argument for a fixed value of N (say $N = 1$). However, note that in this case one can never prove a bound better than $\Omega(1)$ and again, this trick never works in proving any meaningful lower bound.

C.3 Randomized Algorithms

So far the algorithms we have considered are deterministic, i.e. these are algorithm whose behavior is completely determined once the input is fixed. We now consider a generalization of such algorithms to algorithms that have access to random bits. In particular, even when the input is fixed, the behavior of the algorithm might change depending on the actual value of the random bits.[1] For the machine model, it is easy to modify the RAM model from Section C.2.1 to handle randomized algorithms: we can always load a register with independent and uniform random bits in one step.

Typically one considers randomized algorithms due to the following reasons:

- For some problems, it is easier to think of a randomized algorithm. Once one has designed a randomized algorithm, one could then attempt to "derandomize" the randomized algorithm to construct deterministic algorithms.

- In addition to conceptual simplicity, a randomized algorithm might run faster than all corresponding known deterministic algorithms.

- For certain problems, it might be provably impossible to design deterministic algorithms with certain guarantees but it *is* possible to design randomized algorithms with such guarantees. This is a common situation when we might be interested in algorithms that run in sub-linear time.

[1] There are many fundamental and interesting questions regarding how truly random these random bits are and how many such bits can an algorithm access. We will consider the ideal case, where an algorithm has access to as many uniform and independent random bits as it wants.

In this section, we will consider a problem where the third scenario above is applicable. For examples of the first and second scenarios, see Sections 11.3 and **??** respectively.

Before delving into an example problem, we would like to clarify how we determine the run time and correctness of a randomized algorithm. There are multiple natural definitions but we will consider the following ones. The run time of a randomized algorithm will again be the worst-case run time as we defined for deterministic algorithms in Section C.2 for every possible choice of internal random bits that the algorithm might use (with the modification to the RAM model as discussed above). For correctness, the definition for deterministic algorithm was obvious so we did not explicitly state it: for every input, a deterministic algorithm must return the correct output. We call a randomized algorithm correct if on all its inputs, it returns the correct answer with probability bounded away from a $1/2$– to be precise let us say it has to return the correct output with probability at least $2/3$.[2]

We would like to remark on a subtle point in the definition above. In the definition of the correctness of a randomized algorithm above, the probability is taken over the random coin tosses that the algorithm might make. However, note that the guarantee is of the worst-case flavor in the sense that the algorithm has to be correct with high probability for *every* input. This should be contrasted with a scenario where the input might itself be random in which case we might be happy with an average case guarantee where the algorithm is supposed to return the correct output with high probability (over the distribution over the input). In particular, the algorithm is allowed to err on certain inputs as long as the total probability mass on the inputs on which it is incorrect is small: see Chapter 6 where this definition of correctness makes perfect sense. In such situations one can always assume that the algorithm itself is deterministic (see Exercise C.9).

C.3.1 An example problem

In the rest of the section, we will consider the following problem and will attempt to design (deterministic and randomized) algorithms with an eye to illustrate various points that were raised when we defined randomized algorithms.

> Given a vector $\mathbf{x} \in \{0,1\}^n$ determine whether $wt(\mathbf{x}) \leq \frac{n}{3}$ or $wt(\mathbf{x}) \geq \frac{2n}{3}$. For the cases where $wt(\mathbf{x}) \in (n/2, 2n/3)$ the algorithm can have arbitrary behavior.[3]

We will refer to the above as the GAPHAMMING problem.

It is easy to design an $O(n)$ time deterministic algorithm to solve GAPHAMMING: In $O(n)$ one can compute $wt(\mathbf{x})$ and then in $O(1)$ time one can verify if $wt(\mathbf{x}) \leq n/3$ or $wt(\mathbf{x}) \geq 2n/3$. In addition one can show that *any* correct deterministic algorithm will need a run time of $\Omega(n)$: see Exercise C.10.

We will now design a randomized algorithm that solves GAPHAMMING problem. Recall that we only need to determine if $wt(\mathbf{x}) \leq n/3$ or $wt(\mathbf{x}) \geq 2n/3$ (note that we assumed we do not get

[2] The choice of $2/3$ was arbitrary: see Exercise C.8.

[3] Or equivalently one can assume that the algorithm is given the *promise* that it will never encounter an input $\mathbf{x}$ with $wt(\mathbf{x}) \in (n/3, 2n/3)$.

inputs with Hamming weight in $(n/3, 2n/2)$) with high probability. We will present what is called a *sampling* algorithm for this task. To gain intuition, pick a random index $i \in [n]$. Note that then x_i is a random bit. Further, if $wt(\mathbf{x}) \leq n/3$, then $\Pr_i[x_i = 1] \leq 1/3$. On the other hand, if $wt(\mathbf{x}) \geq 2n/3$, then the probability is at least $2/3$. Thus, if we take s samples, with high probability in the first case we expect to see less than $s/3$ ones and in the second case we expect to see at least $2s/3$ ones. To get a constant probability of success we will invoke Chernoff bound to bound the probability of seeing more ones in the first case than the second case. Algorithm 34 for the details.

Algorithm 34 Sampling algorithm for GAPHAMMING

INPUT: $\mathbf{x} \in \{0,1\}^n$

OUTPUT: 0 if $wt(\mathbf{x}) \leq n/3$ and 1 if $wt(\mathbf{x}) \geq 2n/3$ with probability at least $1-\varepsilon$

1: $s \leftarrow 98 \cdot \ln(1/\varepsilon)$
2: $C \leftarrow 0$
3: FOR $j \in [s]$ DO
4: Pick i to be a random index from $[n]$ ▷ The choice of i is independent for each j
5: $C \leftarrow C + x_i$
6: IF $C < s/2$ THEN
7: RETURN 0
8: RETURN 1

It can be checked that Algorithm 34 runs in time $O(\log(1/\varepsilon))$: see Exercise C.11. Next we argue that the algorithm is correct with probability at least $1-\varepsilon$.

Lemma C.3.1. *Algorithm 34 outputs the correct answer with probability at least* $1-\varepsilon$ *for every* $\mathbf{x}$ *(such that* $wt(\mathbf{x}) \notin (n/3, 2n/3)$*).*

Proof. We will prove the lemma for the case when $wt(\mathbf{x}) \leq n/3$ and leave the other case to Exercise C.12.

Fix an arbitrary input $\mathbf{x}$ such that $wt(\mathbf{x}) \leq n/3$. We will argue that at Step 6, we have

$$\Pr\left[C \geq \frac{s}{3} + \frac{s}{7}\right] \leq \varepsilon. \tag{C.2}$$

Note that the above is enough to prove that the algorithm will output 0, as desired.

Towards that end for every $j \in [s]$, let Y_j be the random bit x_i that is picked. Note that $C = \sum_{j=1}^{s} Y_j$. Since each of the Y_j's are independent binary random variables, the additive Chernoff bound (Theorem 3.1.6) implies that

$$\Pr\left[C > \mathbb{E}[C] + \frac{s}{7}\right] \leq e^{-\frac{s}{7^2 \cdot 2}} \leq \varepsilon,$$

where the last inequality follows from our choice of s. As observed earlier for any j, $\Pr[Y_j = 1] \leq 1/3$, which implies that $\mathbb{E}[C] \leq s/3$, which with the above bound implies (C.2), as desired. □

Finally, we consider the average-case version of the GAPHAMMING problem. Our goal is to illustrate the difference between randomized algorithms and average-case algorithms that was alluded to earlier in this section. Recall that in the GAPHAMMING problem, we are trying to distinguish between two classes of inputs: one with Hamming weight at most $n/3$ and the other with Hamming weight at least $2n/3$. We now consider the following natural version where the inputs themselves comes from two distributions and our goal is to distinguish between the two cases.

> Let $\mathbb{D}_p$ denote the distribution on $\{0,1\}^n$, where each bit is picked independently with probability $0 \leq p \leq 1$. Given an $\mathbf{x} \in \{0,1\}^n$ sampled from either $\mathbb{D}_{\frac{1}{3}}$ or $\mathbb{D}_{\frac{2}{3}}$, we need to figure out which distribution $\mathbf{x}$ is sampled from.

The intuition for an algorithm that is correct with high probability (over the corresponding distributions) is same as Algorithm 34, so we directly present the the algorithm for the new version of the problem above.

Algorithm 35 An average-case algorithm for GAPHAMMING

INPUT: $\mathbf{x} \in \{0,1\}^n$ sampled from either $\mathbb{D}_{\frac{1}{3}}$ or $\mathbb{D}_{\frac{2}{3}}$
OUTPUT: 0 if $\mathbf{x}$ was sampled from $\mathbb{D}_{\frac{1}{3}}$ and 1 otherwise with probability at least $1-\varepsilon$

```
s ← 98 · ln(1/ε)
C ← 0
FOR j ∈ [s] DO
    C ← C + x_j
IF C < s/2 THEN
    RETURN 0
RETURN 1
```

Note that unlike Algorithm 34, Algorithm 35 is a *deterministic* algorithm and the algorithm might make an incorrect decision on certain specific inputs $\mathbf{x}$ that it receives.

Using pretty much the same analysis as in the proof of Lemma C.3.1, one can argue that:

Lemma C.3.2. *Let $\mathbf{x}$ be a random sample from $\mathbb{D}_{\frac{1}{3}}$ ($\mathbb{D}_{\frac{2}{3}}$ resp.). Then with probability at least $1-\varepsilon$ (over the choice of $\mathbf{x}$), Algorithm 35 outputs 0 (1 resp.)*

(See Exercise C.13 for a proof.)

C.4 Efficient Algorithms

A major focus of this book is to design algorithms that are efficient. A somewhat smaller focus is to argue that for certain problems efficient algorithms do not exists (maybe with a well accepted assumption that certain computational tasks are hard to accomplish efficiently). In this section, we first begin with the notion of efficient algorithms that will be standard for this book and then

present a peek into how one might argue that a computational task is hard. To illustrate various concepts we will focus on the following problem:

> Given n linear equations over k variables (all over $\mathbb{F}_2$: i.e. all the variables are in $\{0,1\}$ and all arithmetic operations are over the binary field [4] $\mathbb{F}_2$) and an integer $0 \leq s \leq n$, we want to find a solution to the systems of equations that satisfies at least s out of the n equations. We will denote this problem as MAXLINEAREQ(k,n,s). We will drop the arguments when we want to talk about the problem in general.

We choose the non-standard notation of k for number of variables and n for number of equations as they correspond better to problems in coding theory that we would be interested in.

An overwhelming majority of the algorithmic problems considered in this book will have the following property: there are exponentially many possible solutions and we are interested in a solution (or solutions) that satisfy a certain objective. For example, in the MAXLINEAREQ problem, there are 2^k possible solutions and we are interested in a solution that satisfies at least s many linear equations. Note that such problems have a very natural exponential time algorithm: generate all (the exponentially many) potential solutions and check if the current potential solution satisfy the objective. If it does, then the algorithm stops. Otherwise the algorithm continues to the next solution. For example, Algorithm 36 instantiates this general algorithm for the MAXLINEAREQ problem.

Algorithm 36 Exponential time algorithm for MAXLINEAREQ

INPUT: n linear equations over k variables and an integer $0 \leq s \leq n$
OUTPUT: A solution in $\{0,1\}^k$ that satisfies at least s of the equations or fail if none exists

```
1: FOR every x ∈ {0,1}^k DO
2:     Let t be the number of equations the solution x satisfies
3:     IF t ≥ s THEN
4:         RETURN x
5: RETURN fail
```

It is not too hard to argue that Algorithm 36 runs in time $O\left(kn2^k\right)$ (see Exercise C.14). We point out two things that will expand into more detailed discussion on what is an efficient algorithm (and what is not):

1. A run time of $\Omega(2^k)$ is not efficient for even moderate values of k: indeed for $k = 100$, the number of steps of the algorithm exceeds the number of particles in the universe.

2. In the generic exponential time algorithm mentioned earlier, we made the implicit assumption that given a potential solution we can "quickly" verify if the potential solution satisfies the objective or not.

[4] In other words, addition is XOR and multiplication is AND.

Notwithstanding the fact that an exponential run time can become infeasible for moderate input size, one might think that one cannot do better than Algorithm 36 to solve the MAXLINEAREQ problem. In particular, the lack of any extra information other than the fact that we have a system of linear equations on our hands seems to make the possibility of coming up with a faster algorithm slim. However, looks can sometimes be deceptive, as we will see shortly.

Consider the special case of the MAXLINEAREQ problem: MAXLINEAREQ(k,n,n). In other words, we want to see if there exists a solution $\mathbf{x} \in \{0,1\}^n$ that satisfies all the equations. Not only is this is an interesting special case but it is also relevant to this book since this setting corresponds to the error detection problem (Definition 1.3.4) for linear codes (Chapter 2)– see Exercise C.15. It turns out this special setting of parameters makes the problem easy: one can use Gaussian elimination to solve this problem in time $O(kn^2)$ (see Exercise C.16). For the case of $k = \Theta(n)$ (which would be the most important parameter regime for this book), this cubic run time is much faster than the exponential time Algorithm 36. In particular, any run time of the form $O(n^c)$ for some fixed constant c would be much faster than the $2^{\Omega(n)}$ run time of Algorithm 36 (for large enough n). Note that the appealing aspect of a run time of the form $O(n^c)$ is that when the input size doubles, the run time only increases by a constant (though clearly we might be pushing the boundary of a practical definition of a constant for moderately large values of c) as opposed to the exponential run time, where the run time on the new input size is quadratic in the old run time.

In theoretical computer science, the notion of an efficient algorithm is one that runs in time $O(N^c)$ on inputs of size N for some fixed constant $c \geq 0$: such algorithms are said to have *polynomial run time*. In particular, a problem is said to be in the *complexity class* P if it admits a polynomial time algorithm[5]. While one might debate the definition of P as capturing algorithms that are efficient in practice, it clearly seems to capture the difference between problems that "need" exponential time algorithms and problems that have some inherent structure that allows much faster algorithmic solutions (in this case polynomial time algorithm).

C.4.1 Computational Intractability

So far we have talked mainly about problems that admit efficient algorithms. We now consider the issue of how we talk about a problem being hard: e.g. can we somehow formally argue that a certain problem cannot admit efficient solutions? In particular, are there problems where the generic exponential time algorithm discussed earlier is the best possible? To be more precise, let us consider problems where given a potential solution one can in polynomial time determine whether the solution satisfies the objective or not. We call the class of such problems as NP.[6] For example, MAXLINEAREQ(k,n,s) is such a problem because given a potential solution $\mathbf{x}$, one can in time $O(kn)$ verify whether it satisfies at least s out of the n solutions– see Exercise C.17 and hence MAXLINEAREQ $\in \mathsf{NP}$. Like the earlier special case of MAXLINEAREQ(k,n,n),

[5]The technical definition of P is a bit more nuanced: in particular it only considers problems with a binary output but we will ignore this technical issue in this book.

[6]Again for the technical definition we need to only consider problems with binary output but we will ignore this technicality.

the more general problem MAXLINEAREQ(k,n,s) for $s < n$ is also interesting from a coding theory perspective: see Exercise C.18.

Thus, the question of whether there exists a problem where the earlier exponential time algorithm is the best possible is essentially the same as showing $\mathsf{P} \neq \mathsf{NP}$ (see Exercise C.19). While we are nowhere close to answer this fundamental question, we do know a way to identify the "core" of hard problems in NP. Such problems (called NP-complete problems) have the property that if any of them do not have a polynomial time algorithms then none of them do. (Conversely if any of them do have a polynomial time algorithm then $\mathsf{P} = \mathsf{NP}$.) Note that this implies if one assumes that $\mathsf{P} \neq \mathsf{NP}$, then these NP-complete problems are hard problems since they are not in P (which we consider to be the class of "easy" problems).

At first blush, one might wonder how one would go about proving that such problems exist. Proving the existence of such a problem is out of the scope of the book. However, we do want to give an overview of how given one such specific problem that is NP-complete one might argue that another problem is also NP-complete. The way to show such a result is to *reduce* the known NP-complete problem (let us call this problem P_1) to the other problem (let us call this problem P_2). Without going into the technical definition of a reduction, we present an informal definition, which would be sufficient for our purposes. A reduction is a polynomial time algorithm (let us call it $\mathscr{A}_1$) that given an *arbitrary* instance $\mathbf{x}_1$ of P_1 can produce in polynomial time another instance $\mathbf{x}_2$ but this time for the problem P_2 such that given the answer for problem P_2 on $\mathbf{x}_2$, one can in polynomial time exactly determine the answer of P_1 on $\mathbf{x}_1$ (by another algorithm, which let us call $\mathscr{A}_2$). There are two (equivalent) ways to think about such a reduction:

1. A reduction implies that to solve P_1 in polynomial time, it is enough to "only" solve some subset of instances for problem P_2 (in particular, those inputs for P_2 that are generated by $\mathscr{A}_1$ on all possible input instances of P_1). In other words, P_2 is "harder" to solve than P_1. Since the problem P_1 is a hard problem, P_2 is also a hard problem.

2. Let us for the sake of contradiction assume that there exists a polynomial time algorithm $\mathscr{A}_3$ that solves P_2 on all instances (i.e. P_2 is easy). Then one can construct a polynomial time algorithm to solve P_1 as follows. Given an arbitrary input $\mathbf{x}_1$ for P_1, first use $\mathscr{A}_1$ to generate an input $\mathbf{x}_2$ for P_2. Then use $\mathscr{A}_3$ to solve P_2 on $\mathbf{x}_2$ and then convert the answer of P_2 on $\mathbf{x}_2$ to the answer of P_1 on $\mathbf{x}_1$ by using $\mathscr{A}_2$. Note that this is a polynomial time algorithm and is a correct algorithm. Thus, we have proved that P_1 is easy, which contradicts our assumption that P_1 is hard.

To make the concept of reduction a bit less abstract we outline a reduction from a known NP-complete problem to our MAXLINEAREQ problem.[7] In particular, the following problem is known to be NP-complete

> Given a graph $G = (V,E)$ with $|V| = k$ and $|E| = n$ and an integer $0 \leq s \leq n$, does there exist a *cut* of size at least s. In other words, does there exist a subset $S \subset V$ such that the number of edges with one end point in S and the other in $V \setminus$ is at least s? We will call this the MAXCUT(k,n,s) problem.

[7] We assume that the reader is familiar with the mathematical concept of graphs, where we do *not* mean graphs in the sense of plots.

Algorithm 37 is the algorithm $\mathscr{A}_1$ of the reduction from MAXCUT(k, n, s) to MAXLINEAREQ(k, n, s).

Algorithm 37 Reduction from MAXCUT to MAXLINEAREQ

INPUT: An instance for MAXCUT(k, n, s): a graph G and an integer s
OUTPUT: An instance of MAXLINEAREQ(k', n', s')

1: $k' \leftarrow k, n' \leftarrow n, s' \leftarrow s$
2: FOR every vertex $i \in V$ DO
3: Add a variable x_i to the set of variables
4: FOR every edge $(i, j) \in E$ DO
5: Add a linear equation $x_i + x_j = 1$ to the system of equation

Further, the algorithm $\mathscr{A}_2$ is simple: given a solution $(x_1, \ldots, x_k)$ to the instance for MAXLINEAREQ(k, n, s) problem, consider the cut $S = \{i \in V | x_i = 1\}$. It can be checked that this algorithm and Algorithm 37 forms a valid reduction. (See Exercise C.20.)

C.5 Exercises

Exercise C.1. Prove that $f(N)$ is $O(g(N))$ (as per Definition C.1.1) if and only if

$$\lim_{N\to\infty} \frac{f(N)}{g(N)} \leq C,$$

for some absolute constant C.

Exercise C.2. Prove that $f(N)$ is $\Omega(g(N))$ (as per Definition C.1.2) if and only if

$$\lim_{N\to\infty} \frac{f(N)}{g(N)} \geq C,$$

for some absolute constant C.

Exercise C.3. Prove that $f(N)$ is $\Theta(g(N))$ (as per Definition C.1.3) if and only if

$$\lim_{N\to\infty} \frac{f(N)}{g(N)} = C,$$

for some absolute constant C.

Exercise C.4. Prove that $f(N)$ is $o(g(N))$ (as per Definition C.1.4) if and only if

$$\lim_{N\to\infty} \frac{f(N)}{g(N)} = 0.$$

Exercise C.5. Prove that $f(N)$ is $\omega(g(N))$ (as per Definition C.1.5) if and only if

$$\lim_{N\to\infty} \frac{f(N)}{g(N)} = \infty.$$

Exercise C.6. Prove Lemmas C.1.1, C.1.2 and C.1.3.

Exercise C.7. Prove or disprove the following for every $\alpha \in \{O, \Omega, \Theta, o, \omega\}$:

> Let $f(N)$ be $\alpha(h(N))$ and $g(N)$ be $\alpha(h(N))$. Then $f(N) \cdot g(N)$ is $\alpha(h(N))$.

Exercise C.8. Say there exists a randomized algorithm $\mathscr{A}$ that is correct with probability $\frac{1}{2} + \delta$ for some $\delta > 0$ with runtime $T(N)$. Then show that for every $\varepsilon > 0$, there exists another randomized algorithm that is correct with probability $1 - \varepsilon$ with runtime $O\left(\frac{\log(1/\varepsilon)}{\delta} \cdot T(N)\right)$.

Hint: Repeat $\mathscr{A}$ multiple times and pick one among the multiple outputs. For analysis use the Chernoff bound (Theorem 3.1.6).

Exercise C.9. Assume that there is a randomized algorithm $\mathscr{A}$, which one when provided with an input from a distribution $\mathbb{D}$, is correct with probability p (where the probability is taken over both $\mathbb{D}$ and internal randomness of $\mathscr{A}$). Then show that there exists a deterministic algorithm that is correct with probability at least p (where the probability is now only taken over $\mathbb{D}$) with the same run time as $\mathscr{A}$.

Exercise C.10. Argue that any correct deterministic algorithm that solves the GAPHAMMING problem needs a run time of $\Omega(n)$.

Hint: Argue that any correct deterministic algorithm needs to read $\Omega(n)$ bits of the input.

Exercise C.11. Argue that Algorithm 34 runs in time $O(\log(1/\varepsilon))$.

Exercise C.12. Prove that for every $\mathbf{x} \in \{0,1\}^n$ such that $wt(\mathbf{x}) \geq 2n/3$, Algorithm 34 outputs 1 with probability at least $1 - \varepsilon$.

Exercise C.13. Prove Lemma C.3.2 and that Algorithm 35 runs in time $O(\log(1/\varepsilon))$.

Exercise C.14. Argue that Algorithm 36 runs in time $O\left(kn2^k\right)$. Conclude that the algorithm runs in time $2^{O(n)}$.

Exercise C.15. Show that if any MAXLINEAREQ(k, n, n) problem can be solved in time $T(k, n)$, then the error detection for any $[n, k]_2$ code can be solved in $T(k, n)$ time.

Hint: The two problems are in fact *equivalent*.

Exercise C.16. Argue that the problem MAXLINEAREQ(k, n, n) can be solved in time $O(kn^2)$.

Exercise C.17. Show that given a system of n linear equation on k variables over $\mathbb{F}_2$, there exists a $O(kn)$ time algorithm that given a vector $\mathbf{x} \in \{0,1\}^n$ can compute the exact number of equations $\mathbf{x}$ satisfies.

Exercise C.18. Consider the following problem called the BOUNDED DISTANCE DECODING problem. Given a code $C \subseteq \{0,1\}^n$, a vector $\mathbf{y} \in \{0,1\}^n$ and an integer $0 \leq e \leq n$ (called the error radius), output any codeword $\mathbf{c} \in C$ such that $\Delta(\mathbf{c}, \mathbf{y}) \leq e$ (or state that no such codeword exists).

Prove that if any MAXLINEAREQ(k, n, s) problem can be solved in time $T(k, n, s)$, then one can solve the BOUNDED DISTANCE DECODING problem for any $[k, n]_2$ linear code with error radius $n - s$.

Exercise C.19. Argue that showing $\mathsf{P} \neq \mathsf{NP}$ is equivalent to showing that $\mathsf{NP} \setminus \mathsf{P} \neq \emptyset$.

Exercise C.20. Argue that Algorithm 37 and the algorithm $\mathscr{A}_2$ defined just below it are correct and run in polynomial time.

C.6 Bibliographic Notes

The full suite of asymptotic notation in Section C.1 was advocated for analysis of algorithms by Knuth [47]. The Big-Oh notation is credited to Bachmann [2] form a work in 1894 and the little-oh notation was first used by Landau [49] in 1909. A variant of the Big-Omega notation was defined by Hardy and Littlewood [40] in 1914 though the exact definition in Section C.1 seems to be from [47]. The Theta and little omega notation seem to have been defined by Knuth [47] in 1976: Knuth credits Tarjan and Paterson for suggesting the Theta notation to him.

The choice to use worst-case run time as measure of computational efficiency in the RAM model as well as only considering the asymptotic run time (as opposed to more fine grained analysis as advocated by Knuth) seem to have been advocated by Hopcroft and Tarjan: see Tarjan's Turing award lecture for more on this [73].

Cobham [10] and Edmonds [17] are generally credited with making the first forceful case for using P as the notion of efficiently solvable problems. Somewhat interestingly, Peterson's paper on decoding of Reed-Solomon codes [60] that predates these two work explicitly talks about why a polynomial time algorithm is better than an exponential time algorithm (though it does not explicitly define the class P). The notion of NP (along with a proof of the existence of an NP-complete problem) was defined independently by Cook [11] and Levin [51]. This notion really took off when Karp showed that 21 natural problems where NP-complete (including the MAXCUT problem) [46]. For more historical context on P and NP including some relevant historical comments, see the survey by Sipser [68].

The first randomized algorithms is generally credited to Rabin [61]. However, an earlier work of Berlekamp on factoring polynomials presents a randomized algorithm (though it is not stated explicitly as such) [3].

This chapter gave a very brief overview of topics that generally span multiple classes. For further readings, please consult standard textbooks on the subjects of (introductory) algorithms and computational complexity as well as randomized algorithms.