Introduction to Coding Theory Lecture Notes*

Yehuda Lindell
Department of Computer Science
Bar-Ilan University, Israel

January 25, 2010

Abstract

These are lecture notes for an advanced undergraduate (and beginning graduate) course in Coding Theory in the Computer Science Department at Bar-Ilan University. These notes contain the technical material covered but do not include much of the motivation and discussion that is given in the lectures. It is therefore not intended for self study, and is not a replacement for what we cover in class. This is a first draft of the notes and they may therefore contain errors.

*These lecture notes are based on notes taken by Alon Levy in 2008. We thank Alon for his work.

Contents

1 Introduction

The basic problem of coding theory is that of communication over an unreliable channel that results in errors in the transmitted message. It is worthwhile noting that all communication channels have errors, and thus codes are widely used. In fact, they are not just used for network communication, USB channels, satellite communication and so on, but also in disks and other physical media which are also prone to errors.

In addition to their practical application, coding theory has many applications in the theory of computer science. As such it is a topic that is of interest to both practitioners and theoreticians.

Examples:

1. *Parity check:* Consider the following code: For any $x = x_1, \ldots, x_n$ define $C(x) = \oplus_{i=1}^{n} x_i$. This code can detect a single error because any single change will result in the parity check bit being incorrect. The code cannot detect two errors because in such a case one codeword will be mapped to another.

2. *Repetition code:* Let $x = x_1, \ldots, x_n$ be a message and let r be the number of errors that we wish to correct. Then, define $C(x) = x\|x\| \cdots \|x$, where the number of times that x is written in the output is $2r + 1$. Decoding works by taking the n-bit string x that appears a majority of the time. Note that this code corrects r errors because any r errors can change at most r of the x values, and thus the $r+1$ values remain untouched. Thus the original x value is the majority.

The repetition code demonstrates that the coding problem can be solved in principal. However, the problem with this code is that it is extremely wasteful.

The main questions of coding theory:

1. Construct codes that can correct a maximal number of errors while using a minimal amount of redundancy

2. Construct codes (as above) with efficient encoding and decoding procedures

1.1 Basic Definitions

We now proceed to the basic definitions of codes.

Definition 1.1 *Let $A = \{a_1, \ldots, a_q\}$ be an* alphabet*; we call the a_i values* symbols*. A* block code *C of length n over A is a subset of A^n. A vector $c \in C$ is called a* codeword*. The number of elements in C, denoted $|C|$, is called the* size *of the code. A code of length n and size M is called an (n, M)-*code*.*

A code over $A = \{0, 1\}$ is called a binary code *and a code over $A = \{0, 1, 2\}$ is called a* ternary code*.*

Remark 1.2 We will almost exclusively talk about "sending a codeword c" and then finding the codeword c that was originally sent given a vector x obtained by introducing errors into c. This may seem strange at first since we ignore the problem of mapping a message m into a codeword c and then finding m again from c. As we will see later, this is typically not a problem (especially for linear codes) and thus the mapping of original messages to codewords and back is not a concern.

The rate of a code is a measure of its efficiency. Formally:

Definition 1.3 *Let C be an (n, M)-code over an alphabet of size q. Then, the* rate *of C is defined by*

$$\text{rate}(C) = \frac{\log_q M}{n}.$$

Observe that it is possible to specify M messages using $\log_q M$ symbols when there is no redundancy. Thus, the longer that n is, the more wasteful the code (in principal, $\log_q M$ symbols suffice and the $n - \log_q M$ additional symbols are redundancy). Observe that the code $C = A^q$ has an optimal rate of 1, but cannot detect or correct any errors. In contrast, the repetition code has rate of

$$\frac{n}{(2r+1)n} = \frac{1}{2r+1}.$$

Thus, the rate of the repetition code tends to 0 as the number of errors to be corrected increases.

Hamming distance. In general, we will assume that it is more likely to have less errors than more errors. Furthermore, we will assume an upper bound on the number of errors that occur (if we are wrong, then an incorrect message may be received). This "worst case" approach to coding is intuitively appealing within itself, in our opinion. Nevertheless, it is closely connected to a simple probabilistic model where errors are introduced into the message independently for each symbol and with a fixed probability $p < 1/2$. See the textbooks for more on this topic. In order to talk about the "number of errors" we introduce the notion of Hamming distance:

Definition 1.4 *Let $x = x_1, \ldots, x_n$ and $y = y_1, \ldots, y_n$. Then, for every i define*

$$d(x_i, y_i) = \begin{cases} 1 & x_i \neq y_i \\ 0 & x_i = y_i \end{cases}$$

and define

$$d(x, y) = \sum_{i=1}^{n} d(x_i, y_i).$$

We stress that the Hamming distance is not dependent on the actual values of x_i and y_i but only if they are equal to each other or not equal.

Proposition 1.5 *The function d is a metric. That is, for every $x, y, z \in A^n$*

1. $0 \leq d(x, y) \leq n$
2. $d(x, y) = 0$ *if and only if* $x = y$
3. $d(x, y) = d(y, x)$
4. $d(x, z) \leq d(x, y) + d(y, z)$ *(triangle inequality)*

We leave the proof of this as a simple exercise (first prove the proposition for $n = 1$). We now describe the basic rule for decoding:

Definition 1.6 *Let C be a code of length n over an alphabet A. The* nearest neighbor decoding *rule states that every $x \in A^n$ is decoded to $c_x \in C$ that is closest to x. That is, $D(x) = c_x$ where c_x is such that $d(x, c_x) = \min_{c \in C}\{d(x, c)\}$. If there exist more than one c at this minimal distance, then D returns $\perp$.*

Code distance and error detection and correction. Intuitively, a code is better if all the codewords are far apart. We formalize this notion here.

Definition 1.7 *Let C be a code. The* distance *of the code, denoted $d(C)$, is defined by*

$$d(C) = \min\{d(c_1, c_2) \mid c_1, c_2 \in C, c_1 \neq c_2\}$$

An (n, M)-code of distance d is called an (n, M, d)-code. The values n, M, d are called the parameters *of the code.*

Restating what we have discussed above, the aim of coding theory is to construct a code with a short n, and large M and d; equivalently, the aim is to construct a code with a rate that is as close to 1 as possible and with d as large as possible. We now show a connection between the distance of a code and the possibility of detecting and correcting errors.

Definition 1.8 *Let C be a code of length n over alphabet A.*

- *C* detects *u errors if for every codeword $c \in C$ and every $x \in A^n$ with $x \neq c$, it holds that if $d(x,c) \leq u$ then $x \notin C$.*
- *C* corrects *v errors if for every codeword $c \in C$ and every $x \in A^n$ it holds that if $d(x,c) \leq v$ then nearest neighbor decoding of x outputs c.*

The following theorem is easily proven and we leave it as an easy exercise.

Theorem 1.9

- *A code C detects u errors if and only if $d(C) > u$.*
- *A code C corrects v errors if and only if $d(C) \geq 2v + 1$.*

1.2 A Probabilistic Model

The model that we have presented until now is a "worst-case model". Specifically, what interests us is the amount of errors that we can correct and we are not interested in how or where these errors occur. This is the model that we will refer to in most of this course and was the model introduced by Hamming. However, there is another model (introduced by Shannon) which considers probabilistic errors. We will use this model later on (in particular in order to prove Shannon's bounds). In any case, as we will see here, there is a close connection between the two models.

Definition 1.10 *A* communication channel *is comprised of an alphabet $A = \{a_1, \ldots, a_q\}$ and a set of* forward channel probabilities *of the form $\Pr[a_j \text{ received} \mid a_i \text{ was sent}]$ such that for every i:*

$$\sum_{j=1}^{q} \Pr[a_j \text{ received} \mid a_i \text{ was sent}] = 1$$

A communication channel is memoryless *if for all vectors $x = x_1 \ldots x_n$ and $c = c_1 \ldots c_n$ it holds that*

$$\Pr[x \text{ received} \mid c \text{ was sent}] = \prod_{i=1}^{n} \Pr[x_i \text{ received} \mid c_i \text{ was sent}]$$

Note that in a memoryless channel, all errors are independent of each other. This is not a realistic model but is a useful abstraction. We now consider additional simplifications:

Definition 1.11 *A* symmetric channel *is a memoryless communication channel for which there exists a $p < \frac{1}{2}$ such that for every $i, j \in \{0,1\}^n$ with $i \neq j$ it holds that*

$$\sum_{j=1(j\neq i)}^{q} \Pr[a_j \text{ received} \mid a_i \text{ was sent}] = p$$

Note that in a symmetric channel, every symbol has the same probability of error. In addition, if a symbol is received with error, then the probability that it is changed to any given symbol is the same as it being changed to any other symbol.

A *binary symmetric channel* has two probabilities:

$$\begin{aligned} \Pr[1 \text{ received} \mid 0 \text{ was sent}] &= \Pr[0 \text{ received} \mid 1 \text{ was sent}] = p \\ \Pr[1 \text{ received} \mid 1 \text{ was sent}] &= \Pr[0 \text{ received} \mid 0 \text{ was sent}] = 1 - p \end{aligned}$$

The probability p is called the crossover probability.

Maximum likelihood decoding. In this probabilistic model, the decoding rule is also a probabilistic one:

Definition 1.12 *Let C be a code of length n over an alphabet A. The* maximum likelihood decoding *rule states that every $x \in A^n$ is decoded to $c_x \in C$ when*

$$\Pr[x \text{ received} \mid c_x \text{ was sent}] = \max_{c \in C} \Pr[x \text{ received} \mid c \text{ was sent}]$$

If there exist more than one c with this maximum probability, then $\perp$ is returned.

We now show a close connection between maximum likelihood decoding in this probabilistic model, and nearest neighbor decoding.

Theorem 1.13 *In a binary symmetric channel with $p < \frac{1}{2}$, maximum likelihood decoding is equivalent to nearest neighbor decoding.*

Proof: Let C be a code and x the received word. Then for every c and for every i we have that $d(x, c) = i$ *if and only if*

$$\Pr[x \text{ received} \mid c \text{ was sent}] = p^i(1-p)^{n-i}$$

Since $p < \frac{1}{2}$ we have that $\frac{1-p}{p} > 1$. Thus

$$p^i(1-p)^{n-i} = p^{i+1}(1-p)^{n-i-1} \cdot \frac{1-p}{p} > p^{i+1}(1-p)^{n-i-1}.$$

This implies that

$$p^0(1-p)^n > p(1-p)^{n-1} > \ldots > p^n(1-p)^0$$

and so the nearest neighbor yields the codeword that maximizes the required probability. ∎

2 Linear Codes

2.1 Basic Definitions

We denote by $\mathbb{F}_q$ a finite field of size q. Recall that there exists such a finite field for any q that is a power of a prime. In this course, we will just assume that we are given such a field. In linear codes, the alphabet of the code are the elements of some finite field $\mathbb{F}_q$.

Definition 2.1 *A* linear code *with length n over $\mathbb{F}_q$ is a vector subspace of $\mathbb{F}_q^n$.*

The repetition code $C = \{\underbrace{(x, \ldots, x)}_{n} \mid x \in \mathbb{F}_q\}$ is a linear code.

Definition 2.2 *Let C be a linear code over $\mathbb{F}_q^n$. Then*

1. *The* dual code *of C is $C^\perp$ (the orthogonal complement of C in $\mathbb{F}_q^n$).*
2. *The* dimension *of C is the dimension of C as a vector subspace of $\mathbb{F}_q^n$, denoted* $\dim(C)$.

The following theorem is from basic algebra:

Theorem 2.3 *Let C be a linear code of length n over $\mathbb{F}_q$. Then*

1. $|C| = q^{\dim(C)}$
2. *$C^\perp$ is a linear code, and* $\dim(C) + \dim(C^\perp) = n$.
3. $(C^\perp)^\perp = C$

Proof Sketch:

1. This is a basic fact from linear algebra (a subspace with dimension k has q^k elements).
2. It is easy to see that for every set $S \subseteq \mathbb{F}_q^n$ $S^\perp$ is a subspace (exercise). Thus, $C^\perp$ is also a linear code. Furthermore, we know from linear algebra that $\dim(S) + \dim(S^\perp) = n$.
3. From the above, we know that $\dim(C) + \dim(C^\perp) = n$ and $\dim(C^\perp) + \dim(C^{\perp^\perp}) = n$. Thus, $\dim(C) = \dim((C^\perp)^\perp)$ implying that $|C| = |(C^\perp)^\perp|$. It therefore suffices to show that $C \subseteq (C^\perp)^\perp$.

   Let $c \in C$. Then $c \in (C^\perp)^\perp$ if for every $x \in C^\perp$ it holds that $x \cdot c = 0$. But by the definition of the orthogonal complement,

   $$C^\perp = \{x \mid \forall c \in C \; x \cdot c = 0\}$$

   Thus, $x \cdot c = 0$ for all $x \in C^\perp$ and so $c \in (C^\perp)^\perp$.

■

Notation. A linear code of length n and dimension k is denoted an $[n, k]$-code (or an $[n, k, d]$-code when the distance d is specified).

Definition 2.4 *Let C be a linear code. Then*

1. *C is* self orthogonal *if $C \subseteq C^\perp$*
2. *C is* self dual *if $C = C^\perp$*

The following theorem is an immediate corollary of the fact that $\dim(C) + \dim(C^\perp) = n$.

Theorem 2.5

1. *Let C be a self-orthogonal code of length n. Then* $\dim(C) \leq \frac{n}{2}$.
2. *Let C be a self-dual code of length n. Then* $\dim(C) = \frac{n}{2}$.

2.2 Code Weight and Code Distance

Definition 2.6 *Let $x \in \mathbb{F}_q^n$. The* Hamming weight *of x, denoted $\mathsf{wt}(x)$ is defined to be the number of coordinates that are not zero. That is, $\mathsf{wt}(x) \stackrel{\text{def}}{=} d(x,0)$.*

Notation. For $y = (y_1, \ldots, y_n), x = (x_1, \ldots, x_n)$, define $x * y = (x_1 y_1, \ldots, x_n y_n)$

Lemma 2.7 *If $x, y \in \mathbb{F}_2^n$, then $\mathsf{wt}(x+y) = \mathsf{wt}(x) + \mathsf{wt}(y) - 2\mathsf{wt}(x*y)$.*

Proof: Looking at each coordinate separately we have:

$\mathsf{wt}(x+y)$	$\mathsf{wt}(x)$	$\mathsf{wt}(y)$	$\mathsf{wt}(x*y)$
0	0	0	0
1	0	1	0
1	1	0	0
0	1	1	1

The lemma is obtained by summing over all coordinates. ■

Corollary 2.8 *If $x, y \in \mathbb{F}_2^n$ then $\mathsf{wt}(x) + \mathsf{wt}(y) \geq \mathsf{wt}(x+y)$.*

Lemma 2.9 *For every prime power q it holds that for every $x, y \in \mathbb{F}_q^n$*

$$\mathsf{wt}(x) + \mathsf{wt}(y) \geq \mathsf{wt}(x+y) \geq \mathsf{wt}(x) - \mathsf{wt}(y)$$

We leave the proof of this lemma as an exercise.

Definition 2.10 *Let C be a code (not necessarily linear). The* weight *of C, denoted $\mathsf{wt}(C)$, is defined by*

$$\mathsf{wt}(C) = \min_{c \in C; c \neq 0} \{\mathsf{wt}(c)\}$$

The following theorem only holds for linear codes:

Theorem 2.11 *Let C be a linear code over $\mathbb{F}_q^n$. Then $d(C) = \mathsf{wt}(C)$.*

Proof: Let $d = d(C)$. By the definition of the distance of a code, there exist $x', y' \in C$ such that $d(x', y') = d$. Then by linearity we have that $x' - y' \in C$. Now, the weight of the codeword $x' - y'$ is d and so we have found a codeword with weight d implying that $\mathsf{wt}(C) \leq d = d(C)$.

Now, let $w = \mathsf{wt}(C)$. By the definition of weight, there exists a codeword $c \in C$ such that $d(c,0) = \mathsf{wt}(C) = w$. Now, since $0 \in C$ it follows that there exist two codewords in C with distance w from each other. Thus, $d(C) \leq w = \mathsf{wt}(C)$.

We have shown that $\mathsf{wt}(C) \leq d(C)$ and $d(C) \leq \mathsf{wt}(C)$. Thus, $d(C) = \mathsf{wt}(C)$, as required. ■

The above theorem is interesting. In particular, it gives us the first step forward for determining the distance of a code. Previously, in order to calculate the distance of a code, we would have to look at all pairs of codewords and measure their distance (this is quadratic in the size of the code). Using Theorem 2.11 it suffices to look at each codeword in isolation and measure its weight (this is thus linear in the size of the code).

Advantages of Linear Codes

1. A code can be described using its basis. Furthermore, such a basis can be found via Gaussian elimination of a matrix comprised of the codewords as rows.
2. The code's distance equals its weight
3. As we shall see, mapping a message into the code and back is simple.

2.3 Generator and Parity-Check Matrices

Definition 2.12

1. *A* generator matrix *G for a linear code C is a matrix whose rows form a basis for C.*
2. *A* parity check matrix *H for C is a generator matrix for the dual code $C^{\perp}$.*

Remarks:

1. If C is a linear $[n,k]$-code then $G \in \mathbb{F}_q^{k\times n}$ (recall that k denotes the number of rows and n the number of columns), and $H \in \mathbb{F}_q^{(n-k)\times n}$
2. The rows of a generator matrix are linearly independent.
3. In order to show that a k-by-n matrix G is a generator matrix of a code C it suffices to show that the rows of G are codewords in C and that they are linearly independent.

Definition 2.13

1. *A generator matrix is said to be in* standard form *if it is of the form $(I_k \mid X)$, where I_k denotes the k-by-k identity matrix*
2. *A parity check matrix is said to be in* standard form *if it is of the form $(Y \mid I_{n-k})$*

Note that the dimensions of X above are k-by-$(n-k)$, and the dimensions of Y are $(n-k)$-by-k.

Lemma 2.14 *Let C be a linear $[n,k]$-code with generator matrix G. Then for every $v \in \mathbb{F}_q^n$ it holds that $v \in C^{\perp}$ if and only if $v \cdot G^T = 0$. In particular, a matrix $H \in \mathbb{F}_q^{(n-k)\times n}$ is a parity check matrix if and only if its rows are linearly independent and $H \cdot G^T = 0$.*

Proof: Denote the rows of G by $r_1, \ldots, r_k$ (each $r_i \in \mathbb{F}_q^n$). Then for every $c \in C$ we have that

$$c = \sum_{i=1}^{k} \lambda_i r_i$$

for some $\lambda_1, \ldots, \lambda_k \in \mathbb{F}_q$. Now, if $v \in C^{\perp}$ then for every r_i it holds that $v \cdot r_i = 0$ (this holds because each $r_i \in C$). This implies that $v \cdot G^T = 0$, as required.

For the other direction, if $v \cdot G^T = 0$ then for every i it holds that $v \cdot r_i = 0$. Let c be any codeword and let $\lambda_1, \ldots, \lambda_k \in \mathbb{F}_q$ be such that $c = \sum_{i=1}^{k} \lambda_i \cdot r_i$. It follows that

$$v \cdot c = \sum_{i=1}^{k} v \cdot (\lambda_i \cdot r_i) = \sum_{i=1}^{k} \lambda_i \cdot (v \cdot r_i) = \sum_{i=1}^{k} \lambda_i \cdot 0 = 0.$$

This holds for every $c \in C$ and thus $v \in C^{\perp}$.

For the "in particular" part of the lemma, let $H \in \mathbb{F}_q^{(n-k)\times n}$. If H is a parity check matrix then it's rows are linearly independent and in $C^{\perp}$. Thus, by what we have proven it holds that $H \cdot G^T = 0$. For the other direction, if $H \cdot G^T = 0$ then for every row it holds that $v \cdot G^T = 0$ and so every row is in $C^{\perp}$ (by the first part of the proof). Since the rows of the matrix are linearly independent and since the matrix is of the correct dimension, we conclude that H is a parity check matrix for C, as required. ∎

An equivalent formulation: Lemma 2.14 can be equivalently worded as follows.

Let C be a linear $[n,k]$-code with a parity-check matrix H. Then $v \in C$ if and only if $v \cdot H^T = 0$.

This equivalent formulation immediately yields an efficient algorithm for error *detection*.

An additional application of H. The problem of computing the distance of a given code is NP-Hard, even for linear codes. Nevertheless, in certain cases we can use H to compute the code distance.

Theorem 2.15 *Let C be a linear code and let H the parity check matrix for C. Then*

1. *$d(C) \geq d$ if and only if every subset of $d-1$ columns of H are linearly independent.*
2. *$d(C) \leq d$ if and only if there exists a subset of d columns of H that are linearly dependent.*

Proof: The idea behind the proof is that the existence of a word with weight e in C implies linear dependence of e columns of H. In order to see this, recall that $v \in C$ if and only if $v \cdot H^T = 0$. Now, let e be such that $\mathsf{wt}(v) = e$ and let $i_1, \ldots, i_e$ be the non-zero coordinates of v. That is, $v = (0 \cdots 0 v_{i_1} 0 \cdots 0 v_{i_2} 0 \cdots \cdots \cdots 0 v_{i_e} 0 \cdots 0)$. Now

$$vH^T = (v_1 \cdots v_n) \begin{bmatrix} \leftarrow & H_1 & \rightarrow \\ \leftarrow & H_2 & \rightarrow \\ & \vdots & \\ \leftarrow & H_n & \rightarrow \end{bmatrix}$$

and thus the ℓ^{th} coordinate in the output of vH^T equals $\sum_{j=1}^{n} v_j \cdot H_j^\ell$ (where we denote $H_j = H_j^1, \ldots, H_j^{n-k}$). Since only $v_{i_1}, \ldots, v_{i_e}$ are non-zero, we can write that the ℓ^{th} coordinate in the output of vH^T equals $\sum_{j=1}^{e} v_{i_j} \cdot H_{i_j}^\ell$. Recall now that since $v \in C$ it holds that $vH^T = 0$. Thus, for every ℓ it holds that $\sum_{j=1}^{e} v_{i_j} \cdot H_{i_j}^\ell = 0$. This implies that $\sum_{j=1}^{e} v_{i_j} \cdot H_{i_j} = (0, \ldots, 0)$, and so there exist e columns of H that are linearly dependent, as required. Using the same reasoning, any linear combination of e columns of H that yields 0 (which is equivalent to saying that the e columns are linearly dependent) defines a codeword of weight e. Thus, there exists a codeword of weight e *if and only if* e columns of H are linearly dependent.

Now, $d(C) \geq d$ if and only if for every $v \in C$ it holds that $\mathsf{wt}(v) \geq d$. This implies that every $d-1$ columns of H are linearly independent (otherwise, it implies the existence of a word w such that $wH^T = 0$ and so $w \in C$, but $\mathsf{wt}(w) < d$). Likewise, if every $d-1$ columns of H are linearly independent then by what we have seen every word w with $\mathsf{wt}(w) < d$ must fulfill that $wH^T \neq 0$ and so $w \notin C$. Thus, $d(C) \leq d$ if and only if every subset of $d-1$ columns of H are linearly independent.

In addition $d(C) \leq d$ if and only if there exists a $v \in C$ for which $\mathsf{wt}(v) \leq d$ which by what we have seen holds if and only if there exists a subset of d columns of H that are linearly dependent. ∎

Corollary 2.16 *Let C be a linear code and let H be a parity-check matrix for C. Then $d(C) = d$ if and only if every subset of $d-1$ columns in H are linearly independent and there exists a subset of d columns that are dependent in H*

We remark that in general, solving this problem is NP-hard.

Theorem 2.17 *If $G = (I_k \mid X)$ is the generator matrix in standard form for a linear $[n,k]$-code C, then $H = (-X^T \mid I_{n-k})$ is a parity check matrix for C.*

Proof: We have $G \cdot H^T = 0$. In order to see this, we write:

$$G \cdot H^T = \begin{pmatrix} I_k & | & X \end{pmatrix} \begin{pmatrix} -X \\ \\ I_{n-k} \end{pmatrix} = 0$$

In addition, by the I_{n-k} component, we have that the rows of H are linearly independent. Therefore, by Lemma 2.14, H is a parity-check matrix of C. ∎

2.4 Equivalence of Codes

Definition 2.18 *Two (n, M)-codes are* equivalent *if one can be derived from the other by a permutation of the coordinates and multiplication of any specific coordinate by a non-zero scalar.*

Note that permuting the coordinates or multiplying by a non-zero scalar makes no difference to the parameters of the code. In this sense, the codes are therefore equivalent.

Theorem 2.19 *Every linear code C is equivalent to a linear code C' with a generator matrix in standard form.*

Proof: Let G be a generator matrix for C. Then, using Gaussian elimination, find the reduced row echelon form of G. (In this form, every row begins with a one, and there are zeroes below and above it). Given this reduced matrix, we apply a permutation to the columns so that the identity matrix appears in the first k rows. The code generated by the resulting matrix is equivalent to the original one. ∎

2.5 Encoding Messages in Linear Codes

First, we remark that it is possible to always work with standard form matrices. In particular, given a generator G for a linear code, we can efficiently compute G' of standard form using Gaussian elimination. We can then compute H' as we saw previously. Thus, given any generator matrix it is possible to efficiently find its standard-form parity-check matrix (or more exactly the standard-form parity-check matrix of an equivalent code). Note that given this parity-check matrix, it is then possible to compute d by looking for the smallest d for which there are d dependent columns. Unfortunately, we do not have any efficient algorithms for this last task.

Now, let C be a linear $[n, k]$-code over $\mathbb{F}_q$, and let $v_1, \ldots, v_k$ be a basis for it. This implies that for every $c \in C$ there are unique $\lambda_1, \ldots, \lambda_k \in \mathbb{F}_q$ such that $\sum_{i=1}^{k} \lambda_i v_i = c$. In addition, for every $\lambda_1, \ldots, \lambda_k \in \mathbb{F}_q$ it follows that $\sum_{i=1}^{k} \lambda_i v_i \in C$. Therefore, the mapping

$$(\lambda_1, \ldots, \lambda_k) \to \sum_{i=1}^{k} \lambda_i v_i$$

is a 1–1 and onto mapping $\mathbb{F}_q^k$ to the code C. This is true for every basis, and in particular for the generator matrix. We therefore define an encoding procedure E_C as follows. For every $\lambda \in \mathbb{F}_q^k$, define

$$E_C(\lambda) = \lambda \cdot G$$

Observe that if G is in standard form, then $E_C(\lambda) = \lambda \cdot (I_k \mid X) = (\lambda, \lambda \cdot X)$. Thus, it is trivial to map a codeword $E_C(\lambda)$ back to its original message λ (just take its first k coordinates). Specifically, if we are only interested in error detection, then its possible to first compute xH^T; if the result equals 0 then just output the first k coordinates.

The above justifies why we are only interested in the following **decoding** problem:

Given a vector $x \in \mathbb{F}_q^n$ find the closest codeword $c \in C$ to x.

The problems of encoding an original message into a codeword and retrieving it back from the codeword are trivial (at least for linear codes).

2.6 Decoding Linear Codes

Cosets – background. We recall the concept of cosets from algebra.

Definition 2.20 *Let C be a linear code of length n over $\mathbb{F}_q$ and let $u \in \mathbb{F}_q^n$. Then the* coset of C determined by u *is defined to be the set*

$$C + u = \{c + u \mid c \in C\}$$

Example. Let $C = \{000, 101, 010, 111\}$ be a binary linear code. Then,

$$C + 000 = C, \quad C + 010 = \{010, 111, 000, 101\} = C, \quad \text{and} \quad C + 001 = \{001, 100, 011, 110\}$$

Note that $C \cup (C + 001) = \mathbb{F}_2^3$.

Theorem 2.21 *Let C be a linear $[n, k]$-code over $\mathbb{F}_q$. Then*

1. *For every $u \in \mathbb{F}_q^n$ there exists a coset of C that contains u.*
2. *For every $u \in \mathbb{F}_q^n$ we have that $|C + u| = |C| = q^k$*
3. *For every $u, v \in \mathbb{F}_q^n$, $u \in C + v$ implies that $C + u = C + v$*
4. *For every $u, v \in \mathbb{F}_q^n$: either $C + u = C + v$ or $(C + u) \cap (C + v) = \varnothing$*
5. *There are q^{n-k} different cosets for C.*
6. *For every $u, v \in \mathbb{F}_q^n$ it holds that $u - v \in C$ if and only if u and v are in the same coset.*

Proof:

1. The coset $C + u$ contains u.
2. By definition $|C| = q^k$. In addition, $c + u = c' + u$ if and only if $c = c'$. Thus, $|C + u| = |C| = q^k$.
3. Let $u \in C + v$. Then, there exists a $c \in C$ such that $u = c + v$. Let $x \in C + u$; likewise, there exists a $c' \in C$ such that $x = c' + u$. This implies that $x = c' + c + v$. Since $c' + c \in C$ we have that $x \in C + v$ and thus $C + u \subseteq C + v$. Since $|C + u| = |C + v|$ from (2), we conclude that $C + u = C + v$.
4. Let $C + u$ and $C + v$ be cosets. If there exists an x such that $x \in (C + u) \cap (C + v)$ then from (3) it holds that $C + u = C + x$ and $C + v = C + x$ and thus $C + u = C + v$. Thus, either $C + u = C + v$ or they are disjoint.
5. From what we have seen so far, each coset is of size q^k, every vector in $\mathbb{F}_q^n$ is in some coset, and all cosets are disjoint. Thus there are $q^n/q^k = q^{n-k}$ different cosets.
6. Assume that $u - v \in C$ and denote $c = u - v$. Then, $u = c + v \in C + v$. Furthermore, as we have seen $v \in C + v$. Thus, u and v are in the same coset. For the other direction, if u and v are in the same coset $C + x$ then $u = c + x$ and $v = c' + x$ for some $c, c' \in C$. Thus, $u - v = c - c' \in C$ as required.

■

Remark. The above theorem shows that the cosets of C constitution a *partitioning* of the vector space $\mathbb{F}_q^n$. Furthermore, item (6) hints at a decoding procedure: Given u, find v from the same coset and decode to the codeword $u - v$. The question remaining is which v should be taken?

Definition 2.22 *The* leader *of a coset is defined to be the word with the smallest hamming weight in the coset.*

Nearest Neighbor Decoding

The above yields a simple algorithm. Let C be a linear code. Assume that the code word v was sent and the word w received. The error word is $e = w - v \in C + w$. Therefore, given the vector w, we search for the word of the smallest weight in $C + w$. Stated differently, given w we find the leader e of the coset $C + w$ and output $v = w - e \in C$.

The problem with this method is that it requires building and storing an array of all the cosets, and this is very expensive.

Example: Let $C = \{0000, 1011, 0101, 1110\}$. Note that $d = 2$.

$C + 0000:$	0000	1011	0101	1110
$C + 0001:$	**0001**	1010	**0100**	1111
$C + 0010:$	**0010**	1001	0111	1100
$C + 1000:$	**1000**	0011	1101	0110

Observe that since $d = 2$, in at least one of the cosets the leader cannot be unique (otherwise we could correct one error).

Syndrome Decoding

Although for general linear codes we will only see exponential-time algorithms, the algorithm above can be improved using syndromes.

Definition 2.23 *Let C be a linear $[n, k, d]$-code over $\mathbb{F}_q$ and let H be a parity-check matrix for C. Then for every $w \in \mathbb{F}_q^n$ the* syndrome of w determined by H *is defined to be the word*

$$S(w) = wH^T \in \mathbb{F}_q^{n-k}$$

We remark that the syndrome of w depends also on H, so it should actually be denoted $S_H(w)$; for simplicity of notation we will denote it $S(w)$ only.

Theorem 2.24 *Let C be a linear $[n, k, d]$-code and let H be the parity-check matrix of C. Then for every $u, v \in \mathbb{F}_q^n$ it holds that:*

1. $S(u + v) = S(u) + S(v)$
2. $u \in C$ *if and only if* $S(u) = 0$
3. $S(u) = S(v)$ *if and only if u and v are in the same coset*

Proof:

1. Follows immediately from the definition.
2. $S(u) = 0$ if and only if $uH^T = 0$ if and only if $u \in C$
3. $S(u) = S(v)$ if and only if $uH^T = vH^T$ if and only if $(u-v)H^T = 0$ which holds if and only if $u - v \in C$ and equivalently if and only if u and v are in the same coset.

■

The above demonstrates that all the words of the same coset have the same syndrome, and in particular they all have the same syndrome as the coset leader.

Definition 2.25 *A table containing all the pairs $(e, S(e))$ where e is the leader of a coset is called a* syndrome lookup table *or* standard decoding array (SDA).

SDA decoding procedure. Given a vector $v \in \mathbb{F}_q^n$, first compute $S(v)$ and then find the coset leader by looking up $S(v)$ in the SDA. Finally, output $v - e$. By sorting the SDA, this procedure takes time $O(n - k)$. (Recall however that the memory is still exponential in $n - k$ because there are q^{n-k} cosets.)

Constructing an SDA. Naively, an SDA can be built in time q^n by traversing over all the cosets and computing the leader and its syndrome. A faster procedure for small d is as follows:

1. For every e for which $\mathsf{wt}(e) \leq \left\lfloor \frac{d-1}{2} \right\rfloor$ define e to be the leader of a coset (it doesn't matter what the coset is).
2. Store $(e, S(e))$ for every such e

The complexity of this algorithm is linear in the number of words of weight at most $(d-1)/2$. Thus, it takes time and memory

$$\sum_{i=1}^{\lceil \frac{d-1}{2} \rceil} \binom{n}{i} (q-1)^i.$$

This can also be upper bound by

$$\binom{n}{\lceil \frac{d-1}{2} \rceil} q^{\lceil \frac{d-1}{2} \rceil}$$

because you write all possible combinations of symbols in all possible $\lceil \frac{d-1}{2} \rceil$ places. Importantly, if d is a constant, then this procedure runs in *polynomial time*.[1]

In order to justify that this suffices, we remark that every coset has at most one leader with weight less than or equal to $\left\lfloor \frac{d-1}{2} \right\rfloor$. (Otherwise we can take the difference of the two vectors, which must be in C since they are in the same coset. However, by their assumed weight, the weight of the difference must be smaller than d in contradiction to the assumption regarding the distance of C). Thus, no coset leader could have been missed.

On the other hand, if there is a coset with a leader of weight larger than $\left\lfloor \frac{d-1}{2} \right\rfloor$ then that number of errors cannot anyway be corrected, and so there is no reason to store it in the table.

2.7 Summary

A linear code is a vector subspace. Each such code has a generator and parity-check matrix which can be used for encoding messages, computing error detection, and computing the distance of the code. There also exists an algorithm for decoding that is, unfortunately, not polynomial-time in general. However, for d that is constant, it does run in polynomial time. As we proceed in the course, we will see specific linear codes that have efficient decoding procedures.

[1] We remark that this SDA is smaller than the previous one. This is possible because not every coset is necessarily relevant when we consider only $(d-1)/2$ errors.

3 Bounds

For an (n, M, d)-code, the larger the value of M the more efficient the code. We now turn to study bounds on the size of M. In this context, a *lower bound* on M states that it is possible to construct codes that are at least as good as given in the bound, whereas an upper bound states that no code with M this large exists (for some given n and d). Observe that lower bounds and upper bounds here have a reverse meaning as in algorithms (here a lower bound is "good news" whereas an upper bound is "bad news"). Our aim is to find an optimal balance between parameters. We note that sometimes there are different goals that yield different optimality criteria.

3.1 The Main Question of Coding Theory

Recall that the rate of the code is defined to be $R(C) = \frac{\log_q M}{n}$; for a linear $[n, k]$-code we can equivalently write $R(C) = \frac{k}{n}$. We now define a similar notion that combines the distance and length:

Definition 3.1 *For a code C over $\mathbb{F}_q$ with parameters (n, M, d), the* relative distance *of C is defined to be*

$$\delta(C) = \frac{d-1}{n}$$

We remark that relative distance is often defined as d/n; however taking $(d-1)/n$ makes some of the calculations simpler.

Examples:

1. For the trivial code $C = \mathbb{F}_q^n$ we have $d(C) = 1$ and $\delta(C) = 0$
2. Define the repetition code to be the $[n, 1, n]$-code $C = \{0^n, 1^n\}$. We have that $\delta(C) = \frac{n-1}{n} \to 1$, whereas $R(C) = \frac{1}{n} \to 0$.

Definition 3.2 *Let A be an alphabet of size $q > 1$ and fix n, d. We define*

$$A_q(n, d) = \max\{M \mid \text{ there exists an } (n, M, d)\text{-code over } A\}$$

An (n, M, d)-code for which $M = A_q(n, d)$ is called an optimal code.

Definition 3.3 *Let $q > 1$ be a prime power and fix n, d. We define*

$$B_q(n, d) = \max\{q^k \mid \text{ there exists a linear } [n, k, d]\text{-code over } \mathbb{F}_q^n\}$$

A linear $[n, k, d]$-code for which $q^k = B_q(n, d)$ is called an optimal linear code.

We remark that $A_q(n, d)$ and $B_q(n, d)$ depend only on the size q of the alphabet, and not on the alphabet itself.

Theorem 3.4 *Let $q \geq 2$ be a prime power. Then, for every n,*

1. *For every $1 \leq d \leq n$ it holds that $B_q(n, d) \leq A_q(n, d) \leq q^n$*
2. *$B_q(n, 1) = A_q(n, 1) = q^n$*
3. *$B_q(n, n) = A_q(n, n) = q$*

Proof:

1. Directly from the definition and from the fact that every code is a subset of A^n and so $M \leq q^n$.

2. Noting that $\mathbb{F}_q^n$ is a linear $[n, n, 1]$-code and A^n is a $(n, q^n, 1)$-code, we have that $A_q(n, 1), B_q(n, 1) \geq q^n$ (since we have found specific examples of codes this size). From (1) we obtain equality.

3. Since the codewords are all of length n, and the distance must also be n, each different pair of codewords must differ in every coordinate. Therefore there are at most q different words in the code (since the alphabet contains q letters). This implies that $B_q(n, n) \leq A_q(n, n) \leq q$. The repetition code achieves this exact bound and is a linear code. Thus, $B_q(n, n) = A_q(n, n) = q$.

■

Corollary 3.5 *The following codes are optimal:*

1. $C = \mathbb{F}_q^n$

2. *The repetition code* $C = \{a^n \mid a \in \mathbb{F}_q\}$

These codes and any code for which $|C| = 1$ *are called* trivial optimal codes.

We call any code for which $|C| = 1$ a trivial optimal code in order to rule it out when considering "interesting" codes.

3.2 The Sphere-Covering Lower Bound

Definition 3.6 *Let A be an alphabet of size q with $q > 1$. Then for every $u \in A^n$ and every $r \in \mathbb{N}$ $(r \geq 0)$, a sphere with center u and radius r, denoted $S_A(u, r)$, is defined to be the set*

$$\{v \in A^n \mid d(u, v) \leq r\}$$

The volume *of a sphere as above, denoted $V_q^n(r)$, is defined to be $|S_A(u, r)|$.*

Observe that the volume of a sphere depends only on the alphabet size. Thus, only q is specified in the notation. We now compute the volume of a sphere.

Lemma 3.7 *For every natural number $r \geq 0$ and alphabet A of size $q > 1$, and for every $u \in A^n$ we have*

$$V_q^n(r) = \begin{cases} \sum_{i=0}^{r} \binom{n}{i} (q-1)^i & 0 \leq r \leq n \\ q^n & r > n \end{cases}$$

Proof: Let $u \in A^n$. We count the number of vectors $v \in A^n$ with distance exactly m from u. There are $\binom{n}{m}$ ways to choose m coordinates for which v differs from u, and for each one there are $q - 1$ possible choices for the different value. Thus, there are exactly $\binom{n}{m}(q-1)^m$ vectors of distance m from u. Summing over all possible m's we obtain the result for $0 \leq r \leq n$. On the other hand, when $r > n$ then all vectors in the space are within distance r and so the sphere contains the entire space. ■

Observe that in the case of a binary alphabet, we have:

$$V_2^n(r) = \begin{cases} \sum_{i=0}^{r} \binom{n}{i} & 0 \leq r \leq n \\ 2^n & r > n \end{cases}$$

Motivation for the bound. Let C be a code and draw a sphere of radius $d-1$ around every codeword. In an optimal code, it must be the case that the spheres include all of A^n; otherwise, there exists a word that can be added to the code that would be of distance d from all existing codewords. This yields a larger code, in contradiction to the assumed optimality. Thus, the number of codewords in an optimal code is at least the size of the number of spheres that it takes to cover the entire space A^n. Formally:

Theorem 3.8 (sphere-covering bound): *For every natural number $q > 1$ and every $n, d \in \mathbb{N}$ such that $1 \leq d \leq n$ it holds that*

$$A_q(n,d) \geq \frac{q^n}{V_q^n(d-1)}$$

Proof: Let $C = \{c_1, \ldots, c_M\}$ be an *optimal* (n, M, d)-code over an alphabet of size q. That is, $M = A_q(n,d)$. Since C is optimal, there does not exist any word in A^n of distance at least d from every $c_i \in C$ (otherwise, we could add this word to the code without reducing the distance, in contradiction to the optimality of the code). Thus, for every $x \in A^n$ there exists at least one $c_i \in C$ such that $x \in S_A(c_i, d-1)$. This implies that

$$A^n \subseteq \bigcup_{i=1}^{M} S_A(c_i, d-1)$$

and so

$$q^n \leq \sum_{i=1}^{M} |S_A(c_i, d-1)| = M \cdot V_q^n(d-1)$$

Since C is optimal we have $M = A_q(n,d)$ and hence $q^n \leq A_q(n,d) \cdot V_q^n(d-1)$, implying that

$$A_q(n,d) \geq \frac{q^n}{V_q^n(d-1)}$$

■

3.3 The Hamming (Sphere Packing) Upper Bound

We now prove an upper bound, limiting the maximum possible size of any code. The idea behind the upper bound is that if we place spheres of radius $\lfloor \frac{d-1}{2} \rfloor$ around every codeword, then the spheres must be disjoint (otherwise there exists a word that is at distance at most $\lfloor \frac{d-1}{2} \rfloor$ from two codewords and by the triangle inequality there are two codewords at distance at most $d-1$ from each other). The bound is thus derived by computing how many disjoint spheres of this size can be "packed" into the space.

Theorem 3.9 (sphere-packing bound): *For every natural number $q > 1$ and $n, d \in \mathbb{N}$ such that $1 \leq d \leq n$ it holds that*

$$A_q(n,d) \leq \frac{q^n}{V_q^n\left(\lfloor \frac{d-1}{2} \rfloor\right)}$$

Proof: Let $C = \{c_1, \ldots, c_M\}$ be an optimal code with $|A| = q$, and let $e = \lfloor \frac{d-1}{2} \rfloor$. Since $d(C) = d$ the spheres $S_A(c_i, e)$ are all disjoint. Therefore

$$\bigcup_{i=1}^{M} S_A(c_i, e) \subseteq A^n$$

where the union is a disjoint one. Therefore:

$$M \cdot V_q^n\left(\left\lfloor \frac{d-1}{2} \right\rfloor\right) \leq q^n.$$

Using now the fact that $M = A_q(n,d)$ we conclude that

$$A_q(n,d) \leq \frac{q^n}{V_q^n\left(\lfloor \frac{d-1}{2} \rfloor\right)}.$$

■

We stress that it is impossible to prove the existence of a code in this way. This is due to the fact that a word that is not in any of the spheres (and so is at distance greater than $(d-1)/2$ from all codewords) cannot necessarily be added to the code.

Corollary 3.10 *For every natural number $q > 1$ and $n, d \in \mathbb{N}$ such that $1 \leq d \leq n$ it holds that*

$$\frac{q^n}{V_q^n(d-1)} \leq A_q(n,d) \leq \frac{q^n}{V_q^n\left(\lfloor \frac{d-1}{2} \rfloor\right)}$$

Note that there is a huge gap between these two bounds.

3.4 Perfect Codes

We now show that there exist codes that achieve the Hamming (sphere-packing) upper bound. Unfortunately, the codes that we show do not exist for all parameters.

Definition 3.11 *A code C over an alphabet of size q with parameters (n, M, d) is called a* perfect code *if*

$$M = \frac{q^n}{V_q^n\left(\lfloor \frac{d-1}{2} \rfloor\right)}$$

We remark that every perfect code is an optimal code, but not necessarily the other way around.

3.4.1 The Binary Hamming Code

Definition 3.12 *Let $r \geq 2$ and let C be a binary linear code with $n = 2^r - 1$ whose parity-check matrix H is such that the columns are all of the non-zero vectors in $\mathbb{F}_2^r$. This code C is called a* binary Hamming code *of length $2^r - 1$, denoted* $\mathrm{Ham}(r,2)$.

The above definition does not specify the order of the columns and thus there are many Hamming codes. Before proceeding we remark that the matrix H specified in the definition is "legal" because it contains all of the r vectors of weight 1. Thus, H contains I_r and so its rows are linearly independent(since the columns are vectors in $\mathbb{F}_2^r$, the matrix H has r rows).

Example. We write the parity-check matrix for $\mathrm{Ham}(r,2)$.

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

As can be seen, for $\mathrm{Ham}(r,2)$ we have $n = 7$, $k = 4$ and $H \in \mathbb{F}_2^{7\times 3}$.

Proposition 3.13

1. *All binary Hamming codes of a given length are equivalent.*
2. *For every $r \in \mathbb{N}$, the dimension of* $\mathrm{Ham}(r,2)$ *is $k = 2^r - 1 - r$.*
3. *For every $r \in \mathbb{N}$, the distance of* $\mathrm{Ham}(r,2)$ *is $d = 3$ and so the code can correct exactly one error.*

4. Hamming binary codes are perfect codes.

Proof:

1. The only difference between the codes is the order of the columns.
2. H is a matrix of dimension $r \times (2^r - 1)$ and hence $n = 2^r - 1$ and $n - k = r$ implying that $k = n - r = 2^r - 1 - r$.
3. Since all the columns of H are different, every pair of columns are linearly independent. Furthermore, there exist three linearly dependent columns (for instance 001,010,011), Thus, by Corollary 2.16, we have that $d = 3$.
4. For a binary Hamming code: $q = 2$, $n = 2^r - 1$, $d = 3$ and $M = 2^k = 2^{2^r - 1 - r}$. Furthermore
$$V_2^n(1) = \binom{n}{0} + \binom{n}{1} = 1 + n = 2^r.$$
We therefore have that
$$\frac{q^n}{V_q^n(\lfloor \frac{d-1}{2} \rfloor)} = \frac{2^n}{2^r} = 2^{n-r} = 2^{2^r - 1 - r} = M$$

and so the code is perfect, as required. ■

Corollary 3.14 *There exist non-trivial perfect codes, and* $\mathrm{Ham}(r, 2)$ *is a perfect linear code with parameters* $[2^r - 1, 2^r - 1 - r, 3]$.

Discussion. To better understand the parameters of $\mathrm{Ham}(r, 2)$, notice that $r = O(\log n)$ and thus we use $\log n$ bits to fix a single error. Is it really necessary to waste so "many" bits in order to fix just a single error? The answer is yes! This is because the sphere-packing bound says that

$$A_2(n, 3) \leq \frac{2^n}{1 + n} \approx 2^{n - \log n}$$

meaning that in order to transmit $n - \log n$ bits we need to send n bits, and so $\log n$ bits of redundancy are necessary.

3.4.2 Decoding the Binary Hamming Code

Since the Hamming code corrects only a single error, it is possible to decode a given word x by just computing $x'H^T$ for every x' of distance at most one from x, and output x' which gives $x'H^T = 0$. The complexity of this decoding procedure is n multiplications of a vector with the matrix H^T of size $n \log n$. Thus, the overall complexity is $O(n^2 \log n)$.

Hamming himself was troubled by this complexity and presented a better solution. His solution was based on the fact that the coset leaders of $\mathrm{Ham}(r, 2)$ are all the vectors of length n with weight at most one. Now, let e_j be the vector of length $n = 2^r - 1$ with a 1 at the j^{th} coordinate and zeroes in all other coordinates. Then,

$$S(e_j) = e_j \cdot H^T = \text{the } j^{\text{th}} \text{ column of } H$$

Hence if the columns of H are ordered lexicographically, we have that $S(w)$ immediately tells us which bit needs to be corrected.

The algorithm:

1. Upon input w, compute $S(w) = w \cdot H^T$.
2. If $S(w) = 0$ return w.
3. Otherwise $S(w)$ equals the binary representation of j (for $1 \leq j \leq 2^r - 1$). Return $w - e_j$.

Complexity: $O(n \log n)$

3.4.3 Extended Codes

Observe that the Hamming bound is a little odd, since for every pair of values $d, d+1$ where d is odd, the bound does not decrease. This stems from the fact that for odd d,

$$\left\lfloor \frac{d-1}{2} \right\rfloor = \left\lfloor \frac{d}{2} \right\rfloor .$$

This behavior is not incidental (for binary codes) and a binary code with odd distance can always be extended so that the distance is increased by 1. This does not help with error correction, but does help with error detection.

Theorem 3.15 *Let d be odd. Then there exists a binary (n, M, d)-code if and only if there exists a binary $(n+1, M, d+1)$-code. Likewise there exists a binary linear $[n, k, d]$-code if and only if there exists a binary linear $[n+1, k, d+1]$-code.*

Proof: The idea behind the proof of this theorem is to add a parity-check bit to the code. Let C be a binary (n, M, d)-code with odd d. We construct a binary $(n+1, M, d+1)$-code C' as follows. For every $x = (x_1, \ldots, x_n)$ define

$$x' = (x_1, \ldots, x_n, \oplus_{i=1}^{n} x_i).$$

We now compute the distance of C'. For every $x, y \in C$ there are two cases:

1. $d(x, y) \geq d+1$: This implies that $d(x', y') \geq d+1$ since the first n bits are unmodified.

2. $d(x, y) = d$: In this case x and y differ in exactly d places. It suffices to show that the XOR of the bits in which x and y differ is different in x and y (this is because the XOR is clearly the same in all other bits and thus the overall XOR is different meaning that x' and y' also differ in the last coordinate). Let I be the set of indices where $x \neq y$ ($|I| = d$). Let $\alpha, \beta \in \mathbb{N}$ be such that in the I coordinates, x has α zeroes and β ones. Since x and y differ on all these coordinates this implies that y has α ones and β zeroes. Now, $\alpha + \beta = d$ and d is odd. Thus either α is odd and β even, or vice versa. We conclude that the XOR of the bits of x and y in the I coordinates is different.

The other direction of the theorem is left as an exercise. ■

We remark that the construction of an (n, M, d)-code from an $(n+1, M, d+1)$-code, where d is odd, is called puncturing. This is useful if we are given a code with an even distance and we are interested in correcting errors. In this case, we may as well reduce the code length by one, since our error-correction capability remains the same.

3.4.4 Golay Codes

The existence of Hamming codes raises the question as to whether there exist perfect codes with distance $d > 3$? Golay codes are exactly such codes and were used in the Voyager 1 & 2 spacecraft in the late 1970s in order to transmit back color pictures of Jupiter and Saturn with constrained bandwidth.

Definition 3.16 *Let G be a 12×24 matrix $G = (I_{12} \mid A)$ where A is the matrix shown below. Then a linear binary code with generator matrix G is called an* extended binary Golay code *and is denoted G_{24}.*

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

Note that the matrix G is a "legal" generator matrix because it contains I_{12} and so its rows are linearly independent.

Proposition 3.17 (properties of Golay codes):

1. *For the G_{24} code, we have that $n = 24$ and $k = 12$*
2. *The parity-check matrix for G_{24} is $H = (A \mid I_{12})$*
3. *G_{24} is self dual, meaning that $G_{24} = G_{24}^{\perp}$*
4. *The weight of any word in G_{24} is divisible by 4*
5. *There are no codewords in G_{24} of weight 4, and hence $d(G_{24}) = 8$*

Proof:

1. This is immediate from the dimensions of the generator matrix (which is always $n \times k$).
2. This follows from Theorem 2.17 and the fact that A is symmetrical and so $A = A^T$.
3. We can verify that for every two rows r_i and r_j for $i \neq j$ it holds that $r_i \cdot r_j = 0$. Thus $G_{24} \subseteq G_{24}^{\perp}$ (because every word in G_{24} is also in $G_{24}^{\perp}$). Since $\dim(G_{24}) = \dim(G_{24}^{\perp}) = 12$ we have that $G_{24} = G_{24}^{\perp}$.
4. Let $v \in G_{24}$. If v is one of the rows of G then it has either weight 8 or 12. If v is the sum of two rows of G, then from Lemma 2.7 we know that

   $$\mathsf{wt}(v) = \mathsf{wt}(r_i) + \mathsf{wt}(r_j) - 2\mathsf{wt}(r_i * r_j)$$

   where

   $$r_i * r_j = (r_i^1 r_j^1, \ldots, r_i^{24} r_j^{24}).$$

   Since G_{24} is self dual we have $r_i \cdot r_j = 0$, or in other words $\sum_{\ell=1}^{24} r_i^{\ell} r_j^{\ell} = 0 \mod 2$. This means that the sum of $r_i^{\ell} r_j^{l}$ is even, and so $2 \mid \mathsf{wt}(r_i * r_j)$ meaning that $4 \mid 2\mathsf{wt}(r_i * r_j)$. Now, 4 divides $\mathsf{wt}(r_i)$, $\mathsf{wt}(r_j)$ and $2\mathsf{wt}(r_i * r_j)$, and it therefore also divides $\mathsf{wt}(v)$. This can be extended to any linear combination and is left as an exercise. (Note that a linear combination in this case is just the sum of a subset of rows of the matrix.)
5. Since there are words in G with weight 8, we have that $d(G_{24}) \in \{4, 8\}$. Assume there exists a codeword $v \in G_{24}$ such that $\mathsf{wt}(v) = 4$. We write $v = (v_1, v_2)$ where each v_i is 12 bits long. We consider the following cases:

   (a) $\mathsf{wt}(v_1) = 0, \mathsf{wt}(v_2) = 4$: From G we can see that the only word with $v_i = 0$ is the word 0. This stems from the fact that $G = (I_{12} \mid A)$ and we cannot obtain 0 from summing rows of I_{12}.

(b) $\mathsf{wt}(v_1) = 1, \mathsf{wt}(v_2) = 3$: Since $\mathsf{wt}(v_1) = 1$ we have that v must be one of G's rows. However, every row of G has weight 8, 12 or 16.

(c) $\mathsf{wt}(v_1) = 2, \mathsf{wt}(v_2) = 2$: In this case, v is the sum of two rows of G. By looking at A, we can see that there are no two rows that differ in only two places (and so the sum of them is not of vector of weight 2).

(d) $\mathsf{wt}(v_1) = 3, \mathsf{wt}(v_2) = 1$: Since G_{24} is self dual then H is also a generator matrix for G. Since $H = (A \mid I_{12})$ this case is equivalent to (b).

(e) $\mathsf{wt}(v_1) = 4, \mathsf{wt}(v_2)$: Likewise this case is equivalent to (a).

We see that none of the above cases are possible and so there do not exist any codewords of weight 4. We conclude that $d(G_{24}) = 8$.

In summary, we have proven that G_{24} is a $[24, 12, 8]$-code. ■

As we have seen, G_{24} is called the *extended* Golay code. We now define the Golay code itself.

Definition 3.18 *Let $\hat{G}$ be the 12×23 matrix defined by $\hat{G} = (I_{12} \mid \hat{A})$ where $\hat{A}$ is obtained from A by deleting the last column of A. The binary linear code with generator $\hat{G}$ is called the* binary Golay code *and is denoted G_{23}.*

Theorem 3.19 *G_{23} is a binary linear $[23, 12, 7]$-code, and it is perfect.*

Proof: By erasing a single coordinate we can remove at most 1 from the distance, and so $d(G_{23}) \geq 7$. In addition, G has a word of weight 8 with 1 at the last coordinate and so G_{23} has a word of weight 7. We conclude that $d(G_{23}) = 7$. The fact that it is a $[23, 12]$-linear code follows from the dimensions of the matrix (note that no two rows become the same from the puncturing because G begins with I_{12}).

We now show that it is a perfect code. Now, it is perfect if $M = \frac{2^n}{V_2^n(3)}$. We first compute $V_2^n(3)$.

$$V_2^n(3) = \binom{n}{0} + \binom{n}{1} + \binom{n}{2} + \binom{n}{3} = \binom{23}{0} + \binom{23}{1} + \binom{23}{2} + \binom{23}{3} = 1 + 23 + 253 + 1771 = 2048 = 2^{11}.$$

Next, note that

$$M = 2^{12} = \frac{2^{23}}{2^{11}} = \frac{2^n}{V_2^n(3)}.$$

We therefore conclude that the code is perfect. ■

We remark that the extended Golay code is not perfect (and indeed cannot be because d is even!). There also exists a ternary Golay codes with parameters $[11, 6, 5]$, and Hamming codes over an alphabet of size q for every prime power q. All Hamming codes have distance 3.

3.4.5 Summary - Perfect Codes

We have seen that there exist perfect codes with distance 3 for values n of a specific type (every n that equals $2^r - 1$ for some integer r), and for specific lengths we have seen perfect codes with distance 7 (binary Golay code) and distance 5 (ternary Golay code). This begs the questions as to whether there exist perfect codes for all (odd) distances d. An amazing theorem, stated below, states that there are not! In fact, the only (interesting) perfect codes are the Hamming and Golay codes. Before stating the theorem, recall that the trivial perfect codes are $C = \mathbb{F}_q^n$, $C = \{0^n, 1^n\}$ and any C for which $|C| = 1$.

Theorem 3.20 *Let $q \geq 2$ be a prime power. Then every non-trivial perfect code over $\mathbb{F}_q$ has the same parameters of one of the Golay or Hamming codes.*

Observe that essentially the *only* perfect code that can correct 3 errors is the binary Golay code G_{23}.

3.5 The Singleton Bound and MDS Codes

Recall that for every n there exists the trivial $[n, n, 1]$-code. In addition, using code extension, we can construct a $[n+1, n, 2] = [n, n-1, 2]$-code. Can this be continued? In particular, is it possible to construct a $[n, n-d+1, d]$-code for every d? Furthermore, is it possible to do better than this?

We remark that in the case of *binary* codes we have already seen that this is impossible because $\log n$ bits must be added in order to achieve $d = 3$. However, this still leaves open the possibility that better can be achieved for larger alphabets.

Theorem 3.21 (Singleton bound): *For every natural number $q > 1$ and all natural numbers $n, d \in \mathbb{N}$ with $1 \leq d \leq n$ it holds that $A_q(n, d) \leq q^{n-d+1}$. In particular, if C is a linear $[n, k, d]$-code, then $k \leq n - d + 1$*

Proof: Let C be an optimal (n, M, d)-code and so $M = A_q(n, d)$. If we erase the last $d - 1$ coordinates from all words in C, we still remain with the same number of words (this holds because $d(C) = d$ and so the truncated words have at least one remaining different coordinate). Now, since we are left with $n - d + 1$ coordinates there are at most q^{n-d+1} different words, implying that $A_q(n, d) = M \leq q^{n-d+1}$. ∎

Definition 3.22 *A linear code with parameters $[n, k, d]$ such that $k = n - d + 1$ is called a* maximum distance separable (MDS) *code.*

The notions of MDS and perfect codes are incomparable. In particular, the Hamming code is not MDS and we shall see MDS codes that are not perfect. We remark that the Singleton bound is only of interest for large values of q. In particular, the Singleton bound tells us that $k \leq n - d + 1$ and thus for $d = 3$ it holds that $k \leq n - 2$. However, by the Hamming bound, we know that for $q = 2$ it really holds that $k \leq n - \log n$ and thus the bound given by Singleton is very weak. As we will see, for large values of q this is not the case, and Singleton even becomes tight.

Theorem 3.23 (properties of MDS codes): *Let C be a linear code over $\mathbb{F}_q$ with parameters $[n, k, d]$. Let G and H be generator and parity-check matrices for C. The following claims are equivalent:*

1. *C is an MDS code.*
2. *Every subset of $n - k$ columns in H is linearly independent.*
3. *Every subset of k columns in G is linearly independent.*
4. *$C^\perp$ is an MDS code.*

Proof: By Corollary 2.16 every subset of $n - k$ columns in H is linearly independent if and only if $d(C) \geq n - k + 1$. By the Singleton bound, it must be equality and so items (1) and (2) are equivalent. Likewise, (3) and (4) are equivalent for the same reason, since G is a parity check matrix for $C^\perp$.

We now show that (1) implies (4). Since H is a parity-check matrix of C, it is a generator of $C^\perp$ which is a $[n, n-k]$-code. We will show that $d(C^\perp) = k + 1$. If $d(C^\perp) \leq k$ the there exists a word $c \in C^\perp$ such that $\mathsf{wt}(c) \leq k$, and so at least $n - k$ of it's coordinates are 0. Assume without loss of generality that these are the last coordinates. Write $H = (A \mid H')$ where $A \in \mathbb{F}_q^{(n-k)\times k}$, $H' \in \mathbb{F}_q^{(n-k)\times(n-k)}$. Since all the columns of H' are linearly independent (by the proof of equivalence between (1) and (2), H' is invertible. This implies that its rows are also linearly independent (by the facts that a square matrix M is invertible if and only if M^T is invertible, and M is invertible if and only if its columns are linearly independent). Thus, the only way to obtain 0 from a linear combination of H''s rows is to take the "empty" linear combination. This implies that the codeword c with $\mathsf{wt}(c) \leq k$ must be the word $c = 0$. We conclude that all non-zero words have weight greater than k and so $d(C^\perp) = \mathsf{wt}(C^\perp) \geq k + 1$.

It remains to show that (4) implies (1). However, this follows immediately from the fact that $(C^\perp)^\perp = C$ and the fact that (1) implies (4). ∎

3.6 The Reed-Solomon Code

We will use the following useful fact:

Fact 3.24 *For every two different polynomials $p_1, p_2 \in F_q[x]$ of degree smaller than k, there are at most $k-1$ points $\alpha \in F_q$ for which $p_1(\alpha) = p_2(\alpha)$.*

This fact follows from the basic theorem of algebra, that every non trivial polynomial of degree $k-1$ has at most $k-1$ roots. Now, the polynomial $p = p_1 - p_2$ is not the zero polynomial (because $p_1 \neq p_2$) and thus it can have at most $k-1$ roots α. Note now that every root α of p is a point at which p_1 equals p_2 (i.e., $p(\alpha) = 0$ if and only if $p_1(\alpha) = p_2(\alpha)$).

The Reed-Solomon $RS_{q,n,k}$ code. Given n and $k \leq n$, take a prime power q such that $q \geq n$. We will work over the field $\mathbb{F}_q$. (We ignore how these fields are obtained and represented in this course and just comment that if q is prime then we just take $\mathbb{Z}_q$ and otherwise the field is constructed from an irreducible polynomial of the appropriate degree.) The construction is as follows:

1. Choose n different values $\alpha_1, \alpha_2, \ldots, \alpha_n \in \mathbb{F}_q$ (this is why we need $q \geq n$)

2. Let $m = m_0, \ldots, m_{k-1}$ be the original message (where for every i, $m_i \in \mathbb{F}_q$). Define a polynomial using the elements of m as the coefficients:

$$C_m(x) = \sum_{i=0}^{k-1} m_i x^i$$

3. Finally, encode m as follows:

$$RS(m) = \langle C_m(\alpha_1), \ldots, C_m(\alpha_n) \rangle$$

Theorem 3.25 *Let q, n and k be as above. Then, the Reed-Solomon code $RS_{q,n,k}$ is a linear code.*

Proof: We need to show that for every two codewords $\langle C_m(\alpha_1), \ldots, C_m(\alpha_n) \rangle$ and $\langle C_{m'}(\alpha_1), \ldots, C_{m'}(\alpha_n) \rangle$ and every scalar $\beta \in \mathbb{F}_q$ it holds that $\langle (C_m + C_{m'})(\alpha_1), \ldots, (C_m + C_{m'})(\alpha_n) \rangle$ and $\langle \beta C_m(\alpha_1), \ldots, \beta C_m(\alpha_n) \rangle$ are codewords. However this follows immediately from the fact that $C_m + C_{m'}$ and βC_m are polynomials of degree at most $k-1$. ∎

Theorem 3.26 *The Reed-Solomon code $RS_{q,n,k}$ is an $[n, k, n-k+1]$-code and so is MDS.*

Proof: The code length is n and its dimension is k (this holds because every message of length k defines a different polynomial and from Fact 3.24 each different polynomial in turn defines a different codeword). We show that $d(RS_{q,n,k}) = n-k+1$. Let $C(x)$ and $D(x)$ be two different polynomials of degree at most $k-1$. Then from Fact 3.24, there are at most $k-1$ values α for which $C(\alpha) = D(\alpha)$. This implies that for at least $n-k+1$ of the values $\alpha_1, \ldots, \alpha_n$ it holds that $C(\alpha_i) \neq D(\alpha_i)$. Thus, for every distinct m, m' we have that $d(RS(m), RS(m')) \geq n-k+1$ and so the code distance is at least $n-k+1$. Equality is obtained from the Singleton bound and so $d = n-k+1$. ∎

It is important to note that the general methods that we have seen for decoding linear codes are do not run in polynomial-time for the Reed-Solomon code, because d can be very large. Nevertheless, there are efficient decoding procedures and we will see some of them later in the course.

3.7 A Digression – Coding Theory and Communication Complexity

We will now take a short break from coding bounds and see an application of coding theory to theoretical computer science, specifically to the communication complexity. The problem of communication complexity is to study how many bits need to be transmitted in order to compute a function over distributed inputs. In particular, we consider two parties A and B with respective inputs x and y of length k who wish to compute some binary function f of their inputs. The communication complexity of f, denoted $CC(f)$ is the minimum number of bits that A and B need to transmit in order that they both output $f(x, y)$. Consider the equality function EQ defined by $EQ(x, y) = 1$ if and only if $x = y$. The following theorem can be proven:

Theorem 3.27 *$CC(EQ) = k + 1$.*

Observe that the trivial protocol is for A to send her entire k-bit input to B who then replies with the bit $EQ(x, y)$. This protocol requires the transmission of $k + 1$ bits and, as it turns out, is the best one can do.

In an attempt to improve on this, we can consider *probabilistic* protocols. In this case, the parties can toss coins (like any randomized algorithm), and we consider a protocol to be successful if the probability that the parties output an incorrect value is at most $1/3$. Note that the actual constant does not matter too much as the protocol can be repeated in order to lower the error. We denote by $PCC(f)$ the probabilistic communication complexity of computing f; this is the minimum number of bits needed to be transferred in order to compute f with probability of error at most $1/3$. We will prove the following theorem:

Theorem 3.28 *$PCC(EQ) = O(\log k)$.*

Proof: The motivation for the proof of this theorem is as follows. Consider a protocol whereby A sends B a random bit of her input. If $x \neq y$, then with some probability the bit that B receives will constitute a witness that $x \neq y$ and the parties can output 0. However, x and y may differ on only one bit and then the probability that the parties output 0 will only be $\frac{1}{k}$. This can be improved, but only mildly: sending ℓ bits yields a probability of $\frac{\ell}{k}$ that the difference will be detected and so an error will occur with probability $1 - \frac{\ell}{k}$. However, if the parties first apply an error correcting code C to their inputs, then $C(x)$ and $C(y)$ will differ on at least d coordinates, and so the probability that a random bit will be a witness that $x \neq y$ is $\frac{d}{n}$. We now formally demonstrate this.

Let $n = 3k$, $q \geq 3k$ (as close to $3k$ as possible) and $RS : \{0,1\}^k \to \{0,1\}n$ be a Reed-Solomon code with parameters $[3k, k, 2k + 1]$. The protocol works as follows:

1. Upon input $x \in \{0,1\}^k$, A computes $c = RS(x)$. A then chooses a random $i \in \{1, \ldots, 3k\}$ and sends (i, c_i) to B, where c_i is the ith bit of c.

2. B receives (i, c_i) and replies with 1 if the ith coordinate of $RS(y)$ equals c_i; otherwise it replies with 0.

3. A and B both output the bit sent in reply by Bob.

Now, if $x = y$ then A and B always output 1 as required. The interesting case is where $x \neq y$. In this case, the probability (over a random i) that the parties output 0 equals the probability that the ith coordinate of $RS(x)$ is different from the ith coordinate of $RS(y)$. Since $d(RS(x), RS(y)) \geq 2k + 1$ it follows that

$$\Pr_{i \leftarrow [3k]}[RS(x)_i \neq RS(y)_i] \geq \frac{2k+1}{3k} > \frac{2}{3},$$

implying that the probability of error is strictly less than $1/3$. In order to complete the proof, observe that A transmits $O(\log k)$ bits. Specifically, $\log 3k$ bits are used to send the index i and $\log q \approx \log 3k$ to send the ith coordinate of $RS(x)$. Thus, $PCC(EQ) = O(\log k)$. ∎

We conclude from the above that the deterministic communication complexity is exponentially greater than the probabilistic communication complexity. In order to demonstrate the usefulness of this, consider the

case that we wish to check if a 1 Gigabyte file that we worked on has been changed on a server. A 1 Gigabyte file is of size $8 \cdot 2^{30}$. Thus, $k = 8 \cdot 2^{30}$ and $\log 3k = \log 24 \cdot 2^{30} < 35$. We have that only about 70 bits need to be sent, with an error of 1/3 (recall that 2 values of size $\log 3k$ are transmitted). Such an error is typically unacceptable, but repeating this 20 times yields a tiny error of 3^{-20} at the cost of transmitting only about 1400 bits. Thus 175 bytes are communicated instead of a full Gigabyte! As you can see, this is a drastic saving.

3.8 The Gilbert-Varshamov Bound

We now present a lower bound for $B_q(n,d)$. Surprisingly, this bound will be useful for actually constructing codes later on.

Theorem 3.29 *let n, k and d be natural numbers such that $2 \leq d \leq n$ and $1 \leq k \leq n$. If $V_q^{n-1}(d-2) < q^{n-k}$ then there exists a linear code $[n,k]$ over $\mathbb{F}_q$ with distance at least d.*

Proof: We show that if $V_q^{n-1}(d-2) < q^{n-k}$ then there exists a parity-check matrix $H \in \mathbb{F}_q^{(n-k)\times n}$ for which every $d-1$ columns are linearly independent. We construct such an H as follows. Let c_j denote the j^{th} column of H. Then, choose any $c_1 \in \mathbb{F}_q^{n-k}$ and proceed to choose $c_2 \notin \mathsf{span}(c_1)$, and so on. In general, after having chosen $c_1, \ldots, c_j$, we choose c_{j+1} so that for every $I \subseteq \{1, \ldots, j\}$ such that $|I| \leq d-2$ it holds that $c_{j+1} \notin \mathsf{span}(\cup_{i \in I} c_i)$.

In order to ensure that the matrix that we obtain will be a legal parity-check matrix, we start with $c_1, \ldots, c_{n-k}$ that constitute the identity matrix I_{n-k}. We then proceed to choose $c_{n-k+1}, \ldots, c_n$ as described above.

We need to show that this procedure for choosing H can actually be achieved. That is, we need to show that there is always a vector that can be chosen. In order to see this, we count the number of vectors in the span of $d-2$ or less vectors from $c_1, \ldots, c_j$. Note that the number of vectors in the span of i vectors is $(q-1)^i$ (this excludes the case of multiplying a vector by 0 because we count this separately by looking at subsets of all sizes up to $d-2$). We therefore have that the number of vectors in the span of all the subsets of size up to $d-2$ is

$$\sum_{i=0}^{d-2} \binom{j}{i} (q-1)^i \leq \sum_{i=0}^{d-2} \binom{n-1}{i} (q-1)^i = V_q^{n-1}(d-2) < q^{n-k}$$

where the last inequality is by the assumption in the theorem. Since the number of vectors in the span is less than q^{n-k} we can always take a new column as required. This concludes the proof because all $d-1$ columns of H are linearly independent and so the distance of the code is at least d. ∎

Observe that the proof above is *constructive* and as such is very different from the previous lower bounds that we have seen. Unfortunately, however, the algorithm is exponential in d and as such is not efficient. There exists an alternative proof that relies on choosing a random generator matrix and proving that with good probability it has the required distance.

Corollary 3.30 *Let $q > 1$ be a prime power, and let $n, d \in \mathbb{N}$ such that $2 \leq d \leq n$. Then*

$$B_q(n,d) \geq \frac{q^{n-1}}{V_q^{n-1}(d-2)}$$

Proof: Define $k = n - \left\lceil \log_q(V_q^{n-1}(d-2)+1) \right\rceil$. It follows that

$$q^{n-k} = q^{\left\lceil \log_q(V_q^{n-1}(d-2)+1) \right\rceil} \geq V_q^{n-1}(d-2) + 1 > V_q^{n-1}(d-2).$$

Therefore, by the previous theorem there exists a linear $[n,k,d']$-code with $d' \geq d$. It therefore follows that $B_q(n,d) \geq q^k$. The bound is obtained by observing that:

$$q^k = q^{n - \left\lceil \log_q(V_q^{n-1}(d-2)+1) \right\rceil} \geq q^{n-1-\log_q(V_q^{n-1}(d-2)+1)} = \frac{q^{n-1}}{V_q^{n-1}(d-2)+1} \geq \frac{q^{n-1}}{V_q^{n-1}(d-2)}$$

∎

Self exercise: Show that the Hamming code fulfills the GV bound exactly.

3.9 The Plotkin Bound

We now present a bound that is much better than the Singleton and Hamming bounds. However, it is only relevant for limited parameters. This bound uses the Cauchy-Schwarz inequality. This inequality states that for all real $x_1, \ldots, x_m$ and $y_1, \ldots, y_m$ it holds that

$$\left(\sum_{i=1}^{m} x_i y_i\right)^2 \leq \left(\sum_{i=1}^{m} x_i^2\right)\left(\sum_{i=1}^{m} y_i^2\right)$$

Theorem 3.31 *Let $q > 1$ be a natural number, and assume that n and d fulfill the condition that $rn < d$ for $r = 1 - \frac{1}{q}$. Then*

$$A_q(n,d) \leq \left\lfloor \frac{d}{d - rn} \right\rfloor.$$

Proof: Let C be an (n, M, d)-code over an alphabet of size q. Define

$$T = \sum_{c \in C} \sum_{c' \in C} d(c, c').$$

Now, since $d(c, c') \geq d$ for every distinct $c, c' \in C$ we have that

$$T \geq M(M-1)d.$$

We now take a different approach and bound T from above. Let A be an $M \times n$ matrix whose rows are the M codewords in C. For $1 \leq i \leq n$ and a from the alphabet, we denote by $n_{i,a}$ the number of entries in the ith column of A that are equal to a. We have that for every $1 \leq i \leq n$, $\sum_a n_{i,a} = M$. Denoting $c = (c_1, \ldots, c_n)$ and $c' = (c'_1, \ldots, c'_n)$, we have:

$$\begin{aligned} T &= \sum_{i=1}^{n} \left(\sum_{c \in C} \sum_{c' \in C} d(c_i, c'_i) \right) \\ &= \sum_{i=1}^{n} \left(\sum_{a} n_{i,a}(M - n_{i,a}) \right) \\ &= \sum_{i=1}^{n} \sum_{a} n_{i,a} M - \sum_{i=1}^{n} \sum_{a} n_{i,a}^2 \\ &= nM^2 - \sum_{i=1}^{n} \sum_{a} n_{i,a}^2. \end{aligned}$$

We now use the Cauchy-Schwarz inequality with $m = q$, $x_1 = x_2 = \ldots = x_q = 1$ and $y_i = n_{i,a}$ for every i. It thus follows that

$$\sum_a n_{i,a}^2 = \frac{1}{q}\left(\sum_a 1^2\right)\left(\sum_a n_{i,a}^2\right) \geq \frac{1}{q}\left(\sum_a n_{i,a}\right)^2$$

and so we have

$$T = nM^2 - \sum_{i=1}^{n}\sum_a n_{i,a}^2 \leq nM^2 - \sum_{i=1}^{n} \frac{1}{q}\left(\sum_a n_{i,a}\right)^2 = nM^2 - \sum_{i=1}^{n} \frac{1}{q} M^2 = M^2 n\left(1 - \frac{1}{q}\right) = M^2 nr$$

We have therefore shown that

$$M^2 d - Md = M(M-1)d \leq T \leq M^2 nr$$

and so $M^2d - Md \leq M^2nr$. This implies that $Md - d \leq Mnr$ and so $M(d - nr) \leq d$. Since $rn < d$ we have that $d - nr > 0$ and so we conclude that

$$M \leq \frac{d}{d - nr}.$$

(The fact that $d - nr > 0$ means that we can divide both sides by this value without reversing the inequality.) ■

In order to compare this bound to the Singleton bound, consider the case of $q = 2$ and thus $r = 1/2$. Then, for $d > n/2$ we obtain that $A_q(n, d) \leq \left\lfloor \frac{d}{d-n/2} \right\rfloor$. Now, if $d = n/2 + 1$ then this bound gives us that $A_q(n, d) \leq d = \frac{n}{2} + 1$. In contrast, the Singleton bound just tells us that $k \leq n/2$ and so $A_2(n, d) \leq 2^{n/2}$. Thus, the Plotkin bound is exponentially better than Singleton.

We conclude with a version of the Plotkin bound for binary codes:

Theorem 3.32

1. *For even d*

$$A_2(n, d) \leq \begin{cases} 2 \cdot \lfloor \frac{d}{2d-n} \rfloor & n < 2d \\ 4d & n = 2d \end{cases}$$

2. *For odd d*

$$A_2(n, d) = \begin{cases} 2 \cdot \lfloor \frac{d+1}{2d+1-n} \rfloor & n < 2d + 1 \\ 4d + 4 & n = 2d + 1 \end{cases}$$

3.10 The Hadamard Code

Definition 3.33 *A square matrix H_n of dimension $n \times n$ is called a* Hadamard matrix *if all of its entries are in $\{-1, 1\}$ and it holds that $H_n \cdot H_n^T = nI_n$*

There are several ways that we know how to construct Hadamard matrices. We will see the construction by Sylvester that works for any n which is a power of 2. (We remark that there exist constructions for any $n = p^m + 1$ where $p > 2$ is prime.) The construction is as follows:

1. Define $H_1 = (1)$
2. For general n (that is a power of 2), define: $H_{2n} = \begin{pmatrix} H_n & H_n \\ H_n & -H_n \end{pmatrix}$

In order to see that the construction indeed yields a Hadamard matrix, observe that:

$$\begin{aligned} H_{2n}H_{2n}^T &= \begin{pmatrix} H_n & H_n \\ H_n & -H_n \end{pmatrix} \begin{pmatrix} H_n^T & H_n^T \\ H_n^T & -H_n^T \end{pmatrix} \\ &= \begin{pmatrix} H_nH_n^T + H_nH_n^T & H_nH_n^T - H_nH_n^T \\ H_nH_n^T - H_nH_n^T & H_nH_n^T + H_nH_n^T \end{pmatrix} \\ &= \begin{pmatrix} 2H_nH_n^T & 0 \\ 0 & 2H_nH_n^T \end{pmatrix} = \begin{pmatrix} 2\,\mathrm{nI}_n & 0 \\ 0 & 2\,\mathrm{nI}_n \end{pmatrix} = 2\,\mathrm{nI}_{2n} \end{aligned}$$

and thus if H_n is a Hadamard matrix then H_{2n} is also a Hadamard matrix. (Note that in this construction, H_n is a symmetrical matrix and thus $H_n = H_n^T$.)

Proposition 3.34 *Denote the rows of a Hadamard matrix H_n by $h_1, \ldots, h_n$. Then for every $i \neq j$ it holds that $d(h_i, h_j) = \frac{n}{2}$*

Proof: Denote the (i,j)th entry of the matrix H_n by $h_{i,j}$. Then the (i,j)th entry in the matrix $H_n H_n^T$ equals $\sum_{k=1}^{n} h_{i,k} \cdot h_{j,k}$ (because this entry is obtained by multiplying the ith row of H_n by the jth column of H_n). Since $H_n H_n^T = nI_n$, it follows that for $i \neq j$ it holds that $\sum_{k=1}^{n} h_{i,k} \cdot h_{j,k} = 0$. This in turn implies that exactly half of the values in $\{h_{i,k} \cdot h_{j,k}\}_{k=1}^{n}$ equal 1 and exactly half equal -1. Now, $h_{i,k} \cdot h_{j,k} = 1$ if and only if $h_{i,k} = h_{j,k}$, and $h_{i,k} \cdot h_{j,k} = -1$ if and only if $h_{i,k} \neq h_{j,k}$. Thus, $d(h_i, h_j) = \frac{n}{2}$ as required. ■

Definition 3.35 *Let H_n be a Hadamard matrix. A Hadamard code of length n, denoted Had_n, is the binary code derived from H_n by replacing all -1 values with 0 in H_n and then taking all the rows of H_n and their complements as codewords.*

We stress that H_n is *not* a generator matrix, but rather the codewords are the rows themselves.

Proposition 3.36 *Let H_n be a Hadamard matrix. Then, the Hadamard code Had_n derived from H_n is a binary $(n, 2n, \frac{n}{2})$-code.*

Proof: The length of each word is n. Furthermore, there are $2n$ codewords because we take all rows of H_n and their complements. All of these rows and complements define *distinct* codewords because if there exist an i and j such that $\overline{h_i} = h_j$ then $d(h_i, h_j) = n$, but we have already seen $d(h_i, h_j) = \frac{n}{2}$ for all $i \neq j$. Regarding the distance of the code, we have already seen that $d(h_i, h_j) = \frac{n}{2}$ for all $i \neq j$. This implies that $d(\overline{h}_i, \overline{h}_j) = \frac{n}{2}$ as well (because the length is n) and also that $d(h_i, \overline{h}_j) = \frac{n}{2}$ (because for every k, $h_{i,k} = h_{j,k}$ if and only if $h_{i,k} \neq \overline{h}_{j,k}$). Finally, for every i, $d(h_i, \overline{h}_i) = n$. Thus, $d(Had_n) = \frac{n}{2}$, as required. ■

Corollary 3.37 *The Hadamard code is an optimal code.*

Proof: We cannot use the original Plotkin bound here because $r = \frac{1}{2}$ and $d = \frac{n}{2}$ meaning that $rn = d$. However, the version of the Plotkin bound for binary codes states that for $n = 2d$ it holds that $A_2(n, d) \leq 4d = 2n$. This is exactly the size of the Hadamard code and so it is optimal. (If d is odd, then we have that $A_2(n, d) \leq 4d + 4$. In this case, the code is not optimal because we only have a code of size $4d$.) ■

We remark that the rate of this code is very poor. In particular $R(Had_n) = \frac{\log 2n}{n}$ and we send n bits in order to transmit only $\log 2n$ bits. Nevertheless, $\delta(C) \approx \frac{1}{2}$ which is very high.

In comparison, consider the repetition code with distance $d = \frac{n}{2}$. This is the code defined by $C = \{0^{n/2}, 1^{n/2}\}$ and it has rate $R(C) = \frac{\log 2}{n/2} = \frac{2}{n}$, which is considerably worse than the Hadamard code.

3.11 The Walsh-Hadamard Code

Definition 3.38 *The binary* Walsh-Hadamard code *of dimension k is defined by the following function C. For every $x \in \mathbb{F}_2^k$,*

$$C(x) = (\langle x, z_1 \rangle, \ldots, \langle x, z_{2^k-1} \rangle)$$

where $z_1, \ldots, z_{2^k-1}$ are all the nonzero words in $\mathbb{F}_2^k$, and $\langle x, z \rangle = \sum_{i=1}^{k} x_i \cdot z_i \mod 2$.

Equivalently, we can define the Walsh-Hadamard code by its generator matrix G of dimension $k \times 2^k - 1$ where the ith column of G equals z_i. Since a message x is encoded by computing xG, it follows that the codes are identical (because $xG = C(x)$).

Proposition 3.39 *Let $n = 2^k$. Then, the Walsh-Hadamard code C is a linear $[n-1, \log n, \frac{n}{2}]$-code (or equivalently, a linear $[n, \log(n+1), \frac{n+1}{2}]$-code). Furthermore, the code is optimal.*

Proof: The length and dimension of the code follow immediately from its definition (note that G defined above is a legal generator matrix because it contains I_n and so all its rows are linearly independent). In order to compute the code distance, we show that for every nonzero $x \in \mathbb{F}_2^k$, $\mathsf{wt}(C(x)) = d = 2^{k-1}$. Since x is nonzero, there exists an i such that $x_i = 1$. Now, for the computation, we arrange all the z vectors in

pairs that differ only in the ith coordinate. Now, in each pair, it holds that exactly one of the vectors z in the pair is such that $\langle x, z\rangle = 0$, and the other is such that $\langle x, z\rangle = 1$. (This is the case because they differ in only the ith bit and $x_i = 1$.) Since there are 2^{k-1} pairs, we have that there are 2^{k-1} values z such that $\langle x, z\rangle = 1$ and so $\mathsf{wt}(C(x)) \geq 2^{k-1}$. (We remark that in reality there are only $2^{k-1} - 1$ pairs because the pair of the vector e_i is the all zero vector. Nevertheless, this is fine because $\langle x, e_i\rangle = 1$ and so this this adds one to the weight.) We conclude that $d(C) = \mathsf{wt}(C) = 2^{k-1} = \frac{n}{2}$.

The general Plotkin bound can be used here because $d = \frac{n}{2}$ and so $rn = \frac{n-1}{2} < d$. The bound states that

$$A_2(n, d) \leq \left\lfloor \frac{d}{d - rn} \right\rfloor = \left\lfloor \frac{\frac{n}{2}}{\frac{n}{2} - \frac{n-1}{2}} \right\rfloor = \left\lfloor \frac{\frac{n}{2}}{\frac{1}{2}} \right\rfloor = n$$

Since $k = \log n$ we have that $2^k = n$, as required. ■

Hamming and Walsh-Hadamard. Observe that the generator matrix G above is exactly the parity check matrix of the Hamming code. Thus, the Hamming and Walsh-Hadamard codes are *dual codes* of each other.

Decoding the Walsh-Hadamard code. Let $n = 2^k$ (and so the code length is $n-1$). Since the distance of this code is $\frac{n}{2}$ we can correct up to $\frac{n}{4} - 1$ errors. Since the dimension is very small, it is possible to decode using the the following trivial procedure. Let w be the received word. Then, for all x, computing $d(C(x), w)$ and store the x with the smallest distance. This procedure takes time $2^k \cdot k2^k = k \cdot 2^{2k} \approx n^2 \log n$ (there are 2^k possible values of x and for each computing $C(x)$ takes time $k2^k$). This procedure thus takes time that is only slightly greater than *quadratic* in the size of the codeword.

If one is interested in correcting less than $\frac{n}{4}$ errors, then a much more efficient procedure exists. Specifically, assume that we are interested in correction $n(\frac{1}{4} - \epsilon)$ errors, for some constant ϵ. Then, it is possible to correct this amount of errors in time that is polynomial in $\frac{k}{\epsilon}$, which is significantly faster than the above trivial procedure (for large n). We sketch this idea here. Let $x \in \mathbb{F}_2^k$ be the original message, let $w = (w_1, \ldots, w_{2^k-1})$ be the word received and assume that there at most $n(\frac{1}{4} - \epsilon)$ errors. Then, for a random w_j it holds that

$$\Pr[w_j \neq \langle x, z_j\rangle] \leq \frac{n(\frac{1}{4} - \epsilon)}{n} = \frac{1}{4} - \epsilon.$$

Fix $i \in \{1, \ldots, k\}$ and let z_j and z_ℓ be such that $z_j = z_\ell \oplus e_i$ (recall that e_i is the vector of all zeroes except for a one in the ith coordinate). The main observation is that

$$\langle x, z_j\rangle \oplus \langle x, z_\ell\rangle = \sum_{t=1}^{k} x_t \cdot z_{j,t} + \sum_{t=1}^{k} x_t \cdot z_{\ell,t} = x_i \cdot z_{j,i} + x_i \cdot z_{\ell,i} = x_i$$

where the second last equality is due to the fact that for all $t \neq i$ it holds that $z_{j,t} = z_{\ell,t}$ and the last equality is because either $z_{j,i} = 1$ and $z_{\ell,i} = 0$ or $z_{j,i} = 0$ and $z_{\ell,i} = 1$. Thus, if $w_j = \langle x, z_j\rangle$ *and* $w_\ell = \langle x, z_\ell\rangle$ then $w_j + w_\ell = x_i$. We have:

$$\Pr[w_j \oplus w_\ell \neq x_i] \leq \Pr[w_j \neq \langle x, z_j\rangle \vee w_\ell \neq \langle x, z_\ell\rangle] \leq \Pr[w_j \neq \langle x, z_j\rangle] + \Pr[w_\ell \neq \langle x, z_\ell\rangle]$$

where the last inequality is by the union bound and holds even if the events are not independent. In our case, if we choose w_j at random, then the probability that w_j is incorrect is at most $\frac{1}{4} - \epsilon$ and likewise the probability that w_ℓ is incorrect is at most $\frac{1}{4} - \epsilon$. This is true when looking at each event in isolation. We stress that the events are not independent, but this is not needed for the union bound. Thus,

$$\Pr[w_j \oplus w_\ell \neq x_i] \leq \frac{1}{4} - \epsilon + \frac{1}{4} - \epsilon = \frac{1}{2} - 2\epsilon.$$

This implies that if we repeat this process for a number of times that is polynomial in $\frac{k}{\epsilon}$ then the majority will be correct with probability that is very close to 1 (the Chernoff bound is used to prove this). Thus, we

can find the ith bit of x in time poly(k/ϵ) and repeating for each i we can reconstruct the entire string x in time that is polynomial in $\frac{k}{\epsilon}$ (instead of time that is exponential as in the procedure above).

4 Asymptotic Bounds and Shannon's Theorem

Recall that the rate of a code is defined to be $R(C) = \frac{\log_q M}{n}$ and the relative distance is $\delta(C) = \frac{d-1}{n}$. In this section, we will study asymptotic bounds, specifically bounds on codes when $n \to \infty$.

Warm-Up: The Asymptotic Singleton Bound

Proposition 4.1 *Let $\delta = \lim_{n\to\infty} \delta(C)$ and let $R = \lim_{n\to\infty} R(C)$. Then, for every code C it holds that $\delta \leq 1 - R$.*

This follows directly from the Singleton bound that states that for every (n, M, d)-code, $d \leq n - \log_q M + 1$ and equivalently $d - 1 \leq n - \log_q M$. Thus,

$$\delta(C) = \frac{d-1}{n} \leq \frac{n}{n} - \frac{\log_q M}{n} = 1 - R(C) \to 1 - R.$$

Note that in this case, there is actually no difference between the regular and asymptotic Singleton bounds.

4.1 Background: Information/Entropy

An information source $\mathcal{S}$ is a pair (A, P) where $A = \{a_1, \ldots, a_q\}$ is an alphabet and $P = \{p_1, \ldots, p_q\}$ is a set of probabilities such that a_i is sent by $\mathcal{S}$ with probability p_i (and $\sum_{i=1}^{q} p_i = 1$). We assume that each transmission by $\mathcal{S}$ is independent. Now, we wish to measure the amount of "information" that is sent by a source $\mathcal{S}$. If $p_i = \frac{99}{100}$ then the source will almost certainly send the symbol a_i and so not much "information" is gained by actually seeing a_i (by just guessing a_i we are almost always correct anyway). In contrast, if $a_i = \frac{1}{100}$ and we see a_i then we have gained a lot of information! Indeed, if we hadn't seen this transmission, we would have guessed that a_i was sent with only very small probability. Thus, intuitively, the rarer the probability that a symbol is sent, the more information that is gained by seeing this symbol. Likewise, the more likely that a symbol is sent, the less information that is gained by actually seeing it. This leads us to the intuition that the information gained by seeing a symbol a_i that is sent with probability p_i should be very high when $p_i \approx 0$ and should decrease to 0 when p_i tends to 1.

Information function: Let $I(p)$ denote the "amount" of information learned by viewing a symbol that is sent with probability p. We have the following assumptions:

1. $I(p) \geq 0$
2. $I(p)$ is continuous
3. $I(p_i \cdot p_j) = I(p_i) + I(p_j)$: this assumption is due to the fact that each transmission is independent and so the information learned from a transmission of a_i followed by a transmission of a_j should be the sum of the individual information (note that such a transmission pair occurs with probability $p_i \cdot p_j$).

The following fundamental theorem states what the information function must look like:

Theorem 4.2 *A function $I(p)$ defined for all $0 < p \leq 1$ fulfills the three assumptions above if and only if there exists a constant c such that $I(p) = c \log \frac{1}{p}$.*

Proof: It is easy to see that any function $I(p) = c \log \frac{1}{p}$ fulfills the three assumptions. We now show the other direction. From the third requirement we have that:

$$I(p^2) = I(pp) = I(p) + I(p) = 2I(p)$$

and likewise for for every n we have that $I(p^n) = nI(p)$. Replace p with $p^{1/n}$ and we get that $I(p^{1/n}) = \frac{1}{n}I(p)$. In the same way, we obtain that for all natural numbers n, m it holds that $I(p^{n/m}) = \frac{n}{m}I(p)$ and so for every rational $q \in \mathbb{Q}$ we have that $I(p^q) = qI(p)$.

Now, for every real number r there exists a series q_n of rational numbers such that $\lim_{n\to\infty} q_n = r$. This implies that $\lim_{n\to\infty} I(q_n) = I(r)$ because I is continuous.[2] In addition, since the exponentiation function is continuous, we have that $\lim_{n\to\infty} p^{q_n} = p^r$. We therefore have that:

$$I(p^r) = I(\lim_{n\to\infty} p^{q_n}) = \lim_{n\to\infty} I(p^{q_n}) = \lim_{n\to\infty} q_n I(p) = I(p) \lim_{n\to\infty} q_n = rI(p)$$

Now, fix any value of q such that $0 < q < 1$. For every p such that $0 < p < 1$ we can write $p = q^{\log_q p}$. Note that $\log_q p = \frac{\log p}{\log q} = \frac{-\log p}{-\log q} = \frac{\log p^{-1}}{-\log q}$ and because $0 < q < 1$ we have that $-\log q$ is a *positive* constant. We therefore have:

$$I(p) = I(q^{\log_q p}) = I(q) \cdot \log_q p = I(q) \cdot \frac{\log p^{-1}}{-\log q} = \frac{I(q)}{-\log q} \cdot \log \frac{1}{p}.$$

Since q is a fixed value, $\frac{I(q)}{-\log q}$ is a positive constant, as required. The proof is finished by noting that by the continuity of I, it also holds that $I(0) = I(1) = 0$. ■

Definition 4.3 *The* information $I(p)$ *obtained from observing a symbol that is sampled with probability* $p > 0$ *is defined by*

$$I(p) = \log \frac{1}{p}$$

Definition 4.4 *Let* $\mathcal{S} = (A, P)$ *be a source with an alphabet* A *of size* q *and probabilities* $P = \{p_1, \ldots, p_q\}$ *such that the probability that* $\mathcal{S}$ *sends the* i*th symbol equals* p_i*. The average information obtained from a single sample from* $\mathcal{S}$*, denoted* $H(\mathcal{S})$ *and called the* entropy *of the source, is given by:*

$$H(\mathcal{S}) = \sum_{i=1}^{q} p_i \cdot I(p_i) = \sum_{i=1}^{q} p_i \log \frac{1}{p_i} = -\sum_{i=1}^{q} p_i \log p_i$$

where by definition we set $p_i \log p_i = 0$ *for* $p_i = 0$*.*

Notation: We use $H(p)$ to denote the entropy of a binary source where the probability of sampling 0 (or equivalently 1) is p.

Note 4.5 *For* $p_1 = p_2 = \frac{1}{2}$*,* $H(\mathcal{S}) = 1$ *and for* $p_1 = 0$ *or* $p_2 = 0$ *we have that* $H(\mathcal{S}) = 0$*.*

We now prove an important (technical) theorem that provides an asymptotic connection between the entropy function and the volume of a sphere.

Theorem 4.6 *Let* $0 \leq \lambda \leq \frac{1}{2}$*. Then*

$$V_2^n(\lambda n) = \sum_{i=0}^{\lambda n} \binom{n}{i} \leq 2^{nH(\lambda)} \tag{1}$$

$$\lim_{n\to\infty} n^{-1} \log V_2^n(\lambda n) = \lim_{n\to\infty} n^{-1} \log \left(\sum_{i=0}^{\lambda n} \binom{n}{i} \right) = H(\lambda) \tag{2}$$

[2] A real function f is continuous if for any sequence x_n such that $\lim_{n\to\infty} x_n = L$ it holds that $\lim_{n\to\infty} f(x_n) = f(L)$.

Before proving Theorem 4.6, recall **Stirling's approximation**. This approximation states that

$$n! \to \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

This implies that:

$$\begin{aligned}
\log n! &\to \log\left(\sqrt{2\pi n}\left(\frac{n}{e}\right)^n\right) \\
&= \log\sqrt{2\pi n} + n\log n - n\log e \\
&= \frac{1}{2}\log 2\pi + \frac{1}{2}\log n + n\log n - n\log e \\
&\to n\log n - n + O(\log n)
\end{aligned}$$

Furthermore recall that $H(\lambda) = -\lambda\log\lambda - (1-\lambda)\log(1-\lambda)$. We now prove the theorem.

Proof: We first prove Eq. (1). We have:

$$\begin{aligned}
1 = (\lambda + (1-\lambda))^n &= \sum_{i=0}^{n}\binom{n}{i}\lambda^i(1-\lambda)^{n-i} \\
&\geq \sum_{i=0}^{\lambda n}\binom{n}{i}\lambda^i(1-\lambda)^{n-i} \\
&= \sum_{i=0}^{\lambda n}\binom{n}{i}\lambda^i\frac{(1-\lambda)^n}{(1-\lambda)^i} \\
&= \sum_{i=0}^{\lambda n}\binom{n}{i}\left(\frac{\lambda}{1-\lambda}\right)^i(1-\lambda)^n \\
&\geq \sum_{i=0}^{\lambda n}\binom{n}{i}\left(\frac{\lambda}{1-\lambda}\right)^{\lambda n}(1-\lambda)^n \qquad (3)
\end{aligned}$$

where the last inequality holds because $\lambda \leq \frac{1}{2}$ and so $\frac{\lambda}{1-\lambda} \leq 1$ (implying that raising to a higher power can only reduce the size of the outcome). Now:

$$\begin{aligned}
2^{-nH(\lambda)} &= 2^{-n(-\lambda\log\lambda-(1-\lambda)\log(1-\lambda))} \\
&= 2^{\lambda n\log\lambda+(1-\lambda)n\log(1-\lambda)} \\
&= 2^{\lambda n\log\lambda}\cdot 2^{(1-\lambda)n\log(1-\lambda)} \\
&= \lambda^{\lambda n}(1-\lambda)^{(1-\lambda)n} \\
&= \lambda^{\lambda n}\frac{(1-\lambda)^n}{(1-\lambda)^{\lambda n}} \\
&= \left(\frac{\lambda}{1-\lambda}\right)^{\lambda n}(1-\lambda)^n
\end{aligned}$$

Combining this with Eq. (3) we have that:

$$1 \geq \sum_{i=0}^{\lambda n}\binom{n}{i}2^{-nH(\lambda)} \quad \text{and so} \quad \sum_{i=0}^{\lambda n}\binom{n}{i} \leq 2^{nH(\lambda)}.$$

This completes the proof of Eq. (1).

We now prove Eq. (2). Define $m = \lfloor \lambda n \rfloor$. This implies that either $m = \lambda n$ or $m = \lambda n - 1$; for simplicity, we will assume that $m = \lambda n$. Now,

$$\begin{aligned} n^{-1} \log \sum_{i=0}^{\lambda n} \binom{n}{i} &\geq n^{-1} \log \binom{n}{m} = n^{-1} \log \left(\frac{n!}{m!(n-m)!} \right) \\ &= n^{-1}(\log n! - \log m! - \log(n-m)!) \\ &\to n^{-1}(n \log n - n - m \log m + m - (n-m)\log(n-m) + n - m + O(\log n)) \\ &= \log n - \frac{m}{n} \log m - \frac{n-m}{n} \log(n-m) + o(1) \end{aligned}$$

where the second last step is by Stirling's theorem. Now, since $m = \lambda n$ we have that $\frac{m}{n} = \lambda$ and $\frac{n-m}{n} = 1-\lambda$. Thus:

$$\begin{aligned} \lim_{n\to\infty} n^{-1} \log \sum_{i=0}^{\lambda n} \binom{n}{i} &\geq \log n - \lambda \log \lambda n - (1-\lambda)\log((1-\lambda)n) + o(1) \\ &= \log n - \lambda \log \lambda - \lambda \log n - (1-\lambda)\log(1-\lambda) - (1-\lambda)\log n + o(1) \\ &= H(\lambda) + \log n - \lambda \log n - \log n + \lambda \log n + o(1) \\ &= H(\lambda) + o(1) \end{aligned}$$

Therefore,

$$\lim_{n\to\infty} n^{-1} \log \sum_{i=0}^{\lambda n} \binom{n}{i} \geq H(\lambda) + o(1)$$

On the other hand, from Eq. (1), we have that

$$\log \sum_{i=0}^{\lambda n} \binom{n}{i} \leq nH(\lambda)$$

and therefore

$$\lim_{n\to\infty} n^{-1} \log \sum_{i=0}^{\lambda n} \binom{n}{i} = H(\lambda)$$

as required. ■

4.2 Asymptotic Bounds on Codes

The Asymptotic Sphere-Packing Bound

From here on, we use the following notation. Let $\mathcal{C} = \{C_n\}$ be a family of codes, such that C_n is the concrete code of length n in the family. Then, $\delta = \delta(\mathcal{C}) = \lim_{n\to\infty} \delta(C_n)$ and $R = R(\mathcal{C}) = \lim_{n\to\infty} R(C_n)$. Some of the bounds below hold for every n and in this case we will write the bound for C_n, and some hold only for $n \to \infty$ in which case we will write the bound for $\mathcal{C}$.

Theorem 4.7 *For every binary code C with asymptotic relative distance $\delta \leq \frac{1}{2}$ and rate R it holds that*

$$R \leq 1 - H\left(\frac{\delta}{2}\right)$$

Proof: Recall that $R = \frac{\log_n M}{n}$ and so $n \cdot R = \log_2 M$. The sphere-packing bound states that

$$M \leq A_q(n,d) \leq \frac{q^n}{V_q^n\left(\lfloor \frac{d-1}{2} \rfloor\right)}$$

and so

$$\begin{aligned} Rn &= \log M \leq \log A_2(n,d) \\ &\leq \log\left(\frac{2^n}{V_2^n\left\lfloor\frac{d-1}{2}\right\rfloor}\right) \\ &= n - \log V_2^n\left(\left\lfloor\frac{d-1}{2}\right\rfloor\right) \\ &\to n - \log 2^{nH(\frac{\delta}{2})} \\ &= n - nH\left(\frac{\delta}{2}\right) \end{aligned}$$

Dividing by n, we obtain that

$$R \leq 1 - H\left(\frac{\delta}{2}\right)$$

■

The Asymptotic Gilbert-Varshamov Bound

Theorem 4.8 *Let n, k and d be such that $R \leq 1 - H(\delta)$ where $R = \frac{k}{n}$ and $\delta = \frac{d-1}{n} \leq \frac{1}{2}$. Then, there exists a binary linear code C_n with rate R and distance at least d.*

Proof: If $R \leq 1 - H(\delta)$ then $nR \leq n - nH(\delta)$ and $n - nR \geq nH(\delta)$. Since $nR = k$ we have that $n - k \geq nH(\delta)$. This implies that

$$2^{n-k} \geq 2^{nH(\delta)} \geq V_2^n(\delta n) = V_2^n(d-1) > V_2^n(d-2)$$

where the second inequality is by Theorem 4.6 (which holds as long as $\delta \leq 1/2$). Thus, by the Gilbert-Varshamov bound, there exists a binary linear code with distance at least d. ■

Corollary 4.9 *For every n, there exists a binary linear code C_n with asymptotic relative distance and rate that are* constant *and non-zero.*

Proof: Take any δ that is strictly between 0 and 0.5. For example, take $\delta = 1/4$. Then, $H(\delta) = -0.25\log 0.25 - 0.75\log 0.75 \approx 0.81$. This implies that there exists a code with relative distance 0.25 and rate 0.18. ■

Observe that as $\delta \to 1/2$, $H(\delta) \to 1$ and so $R \to 0$; this is as we would expect. Conversely, as $\delta \to 0$ we have that $R \to 1$. The above theorem tells us that we can choose anything we like in between these extremes.

Asymptotic behavior of the codes we have seen. All of the codes that we have seen so far have the property that either $R = 0$ or $\delta = 0$, or they are not binary. We will see this now:

- **Hamming:** This is a $[2^r - 1, 2^r - 1 - r, 3] = [n-1, n - \log n - 1, 3]$-code. Thus,

$$R = \frac{n - \log n - 1}{n-1} = 1 - \frac{\log n}{n-1} \to 1$$

and

$$\delta = \frac{2}{n-1} \to 0$$

- **Reed-Solomon:** This is a $[n, n-d+1, d]$ code. Thus we can set $d = \frac{n}{2}$ and obtain $R = \frac{n/2+1}{n} \to \frac{1}{2}$ and $\delta = \frac{1}{2}$. However, Reed-Solomon only achieves this for large alphabets; in particular where $q \geq n$.
- **Hadamard:** This is a $[n, \log(n+1), \frac{n+1}{2}]$-code. Thus, $R \to 0$ and $\delta \to \frac{1}{2}$.

The importance of Gilbert-Varshamov is that it tells us that it is actually possible to build a code that will *simultaneously* have a constant R and constant δ, for $n \to \infty$.

4.3 Shannon's Theorem

One of the most important bounds is given by Shannon's theorem. This theorem focuses on the probabilistic model of a binary symmetrical channel (BSC) and states that it is possible to correct errors, even if the probability of error per bit is close to 1/2.

Recall that in a BSC, the probability of error is some $p < \frac{1}{2}$ where

$$\Pr[0 \text{ received} \mid 1 \text{ was sent}] = \Pr[1 \text{ received} \mid 0 \text{ was sent}] = p$$

and

$$\Pr[0 \text{ received} \mid 0 \text{ was sent}] = \Pr[1 \text{ received} \mid 1 \text{ was sent}] = 1 - p.$$

Let z be a message. Then, we model the (noisy) transmission of a message z in such a channel by choosing an error string e of length $|z|$ such that every bit of e equals 1 with probability exactly p (and each bit is chosen independently). The output string after the transmission is the string $z + e \mod 2$. We denote the random process of choosing the error string according to this distribution by $e \leftarrow BSC_p$. In general we write $x \leftarrow D$ to say that x is chosen according to distribution D. We also denote by $x \leftarrow \{0,1\}^k$ the process of choosing x according to the *uniform distribution*.

Definition 4.10 *Let $E : \{0,1\}^k \to \{0,1\}^n$ be an encoding scheme and $D : \{0,1\}^n \to \{0,1\}^k$ a decoding scheme. We say that (E, D)* corrects BSC_p with error ϵ *if*

$$\Pr_{\substack{e \leftarrow BSC_p \\ x \leftarrow \{0,1\}^k}} [D(E(x) + e) \neq x] < \epsilon.$$

Theorem 4.11 (Shannon's Theorem): *Let $0 \leq p < \frac{1}{2}$ and let $\delta > 0, \epsilon > 0$ be any constants. Then there exist functions E and D as described above such that (E, D) corrects BSC_p with error ϵ and in addition $R = \frac{k}{n} = 1 - H(p) - \delta$ for all large enough n and k.*

Stated in words, Shannon's theorem says that we can fix errors over a BSC_p channel with rate as close to $1 - H(p)$ as we like (making δ small) and with a decoding error as small as desired (making ϵ small). The effect of ϵ and δ is on the size of n and k we need to take, as we will see in the proof.

The Chernoff bound. The proof of Shannon's theorem uses the Chernoff bound, a useful probabilistic bound that essentially states that independent random variables deviate from their expected value with exponentially small probability.

Lemma 4.12 (the Chernoff bound): *Let D be a distribution over $\{0,1\}$ and let $X_1, \ldots, X_N$ be independent random variables that are distributed according to D. If $\mu = \mathrm{Exp}_{x \leftarrow D}[x]$ then for every λ it holds that:*

$$\Pr_{X_1, \ldots, X_N \leftarrow D} \left[\left| \frac{\sum_{i=1}^N X_i}{N} - \mu \right| \geq \lambda \right] \leq e^{-\lambda^2 \cdot \frac{N}{2}}$$

We are now ready to prove Shannon's theorem.

Proof: We prove Shannon's theorem by showing that a random function E is a good enough code. Fix $p < \frac{1}{2}$, $\epsilon > 0$ and $\delta > 0$. Then, let n and k be such that $\frac{k}{n} = 1 - H(p) - \delta$ and let $E : \{0,1\}^k \to \{0,1\}^n$

be any function. For every $y \in \{0,1\}^n$ we define $D(y)$ to be the string $x \in \{0,1\}^k$ such that $d(E(x), y)$ is minimal over all $x' \in \{0,1\}^k$ (i.e., $E(x)$ is the nearest neighbor of y). We choose a small constant $\gamma > 0$ such that

$$H(p+\gamma) < H(p) + \delta.$$

We can always choose such a γ. Let δ' be such that $H(p+\gamma) = H(p) + \delta'$. Fix a string $x \in \{0,1\}^k$ and $E(x) \in \{0,1\}^n$ and view all the other strings $E(x')$ as being uniformly distributed. We say that a string $e \leftarrow BSC_p$ is bad if $\mathsf{wt}(e) > (p+\gamma)n$; equivalently e is bad if $\frac{\mathsf{wt}(e)}{n} > p + \gamma$.

We now use the Chernoff bound to calculate the probability that an error string e is bad. We can use Chernoff because each bit of e is independently chosen according to BSC_p and the weight of e is the sum of bits equalling 1. We have:

$$\Pr_{e \leftarrow BSC_p}[e \text{ is } \mathsf{bad}] \leq \Pr\left[\left|\frac{\mathsf{wt}(e)}{n} - p\right| \geq \gamma\right] \leq e^{-\gamma^2 \frac{n}{2}} = c^{-n}$$

for some constant $c > 1$ (note that $c = e^{\frac{\gamma^2}{2}}$ and $\gamma > 0$ implying that $c > 1$). We therefore have that e is bad with probability that decreases exponentially as n increases.

Let $y = E(x) + e$. We now bound the volume of the sphere of radius $(p+\gamma)n$ around y (this tells us how many codewords $E(x)$ there are within distance $(p+\gamma)n$ from y):

$$|S_2(y, (p+\gamma)n)| = V_2^n((p+\gamma)n) \leq 2^{nH(p+\gamma)} = 2^{nH(p)+n\delta'}$$

Let $x' \neq x$. Then,

$$\Pr_E[E(x') \in S_2(y, (p+\gamma)n)] = \frac{V_2^n((p+\gamma)n)}{2^n} \leq 2^{nH(p)+n\delta'-n}$$

(the probability above is taken over the random choice of values $E(x')$ for every $x' \neq x$). Since there are 2^k different values x', we can use the union bound to bound the probability that there *exists* an $x' \neq x$ in $S_2(y, (p+\gamma)n)$. (Recall that the union bound states that $\Pr[\bigcup X] \leq \sum \Pr[X]$.) We have:

$$\Pr_E[\exists x' \neq x \text{ s.t. } E(x') \in S_2(y, (p+\gamma)n)] \leq 2^k \cdot 2^{nH(p)+n\delta'-n} = 2^{nH(p)+n\delta'-n+k}$$

Now, $\frac{k}{n} = 1 - H(p) - \delta$ and so $k = n - nH(p) - n\delta$. Thus,

$$2^{nH(p)+n\delta'-n+k} = 2^{nH(p)+\delta' n - n + n - nH(p) - \delta n} = 2^{(\delta'-\delta)n}.$$

Since $\delta' < \delta$ there exists a constant $a > 1$ such that

$$\Pr_E[\exists x' \neq x \text{ s.t. } E(x') \in S_2(y, (p+\gamma)n)] < a^{-n}$$

(for example, take $a = 2^{\delta - \delta'}$).

We have now reached the key point in the proof. If an error in decoding occurs, then at least one of the following events must have occurred:

1. *There are too many errors:* Formally, e is bad, meaning that $\mathsf{wt}(e) > (p+\gamma)n$.
2. *There are not too many errors, but there are close codewords:* Formally, e is not bad, but there exists a string $x' \neq x$ such that $E(x') \in S_2(y, (p+\gamma)n)$.

It is clear that if neither of the above events occur, then $y = E(x)$ will be correctly decoded to x. We therefore have that for every fixed x,

$$\Pr_{\substack{E \\ e \leftarrow BSC_p}}[D(E(x)+e) \neq x] < a^{-n} + c^{-n}.$$

Since the above holds for every fixed x and $E(x)$ when $E(x')$ is chosen at random ($x' \neq x$), it also holds for a random x when $E(x)$ is random (for all x). Thus, we obtain that

$$\Pr_{E,e\leftarrow BSC_p,x\leftarrow\{0,1\}^k}[D(E(x)+e)\neq x] < a^{-n}+c^{-n}.$$

By the probabilistic method, this implies that there *exists* a function E such that

$$\Pr_{\substack{e\leftarrow BSC_p\\ x\leftarrow\{0,1\}^k}}[D(E(x)+e)\neq x] < a^{-n}+c^{-n}.$$

The proof is completed by observing for any $\epsilon > 0$ we have that for large enough n, $a^{-n}+c^{-n} < \epsilon$. (Since a and c are constants that do not depend on the value of k – they depend only on γ which is a function of δ and p – we can choose a large enough n so that $a^{-n}+c^{-n} < \epsilon$ and then fix $k = n - nH(p) - n\delta$ as required.) This then implies the existence of (E, D) as required. ■

Discussion: Compare Shannon's theorem to the Gilbert-Varshamov bound. In what way is Shannon stronger? Is it also weaker in some sense?

4.4 Channel Capacity (Shannon's Converse)

We saw that in a binary symmetrical channel with probability p it is possible to transmit at a rate that is as close to $R = 1 - H(p)$ as we desire. Shannon also showed that it is *not* possible to transmit at a higher rate. That is, if $R > 1 - H(p)$ then it is impossible to correct errors. Thus, $1 - H(p)$ is called the capacity of the channel. We now prove the converse.

Theorem 4.13 *Let $k, n \in \mathbb{N}$ and $0 \leq p < \frac{1}{2}$, such that $\frac{k}{n} = 1 - H(p) + \delta$ for some constant $\delta > 0$. Then for every pair of functions (E, D) it holds that*

$$\lim_{k,n\to\infty}\left(\Pr_{\substack{e\leftarrow BSC_p\\ x\leftarrow\{0,1\}^k}}[D(E(x)+e)=x]\right) = 0$$

Proof Sketch: We present a proof sketch of this theorem only. If $e \leftarrow BSC_p$ then this implies that $\mathrm{Exp}_e[\mathsf{wt}(e)] = pn$. Thus $\mathsf{wt}(e) \geq pn$ with good probability. Now, there are at least $\binom{n}{pn}$ strings z such that $\mathsf{wt}(z) \geq pn$ and the error string e equals one of them with good probability. In order for the decoding to work correctly, all of the errors need to be corrected. Thus, for every codeword x there must be at least $\binom{n}{pn}$ strings y such that $D(y) = x$ (because this is the number of different strings that can be obtained by adding errors to x). Counting now over all the 2^k different code words, we have that the number of strings in the range $\{0,1\}^n$ must be at least

$$2^k \cdot \binom{n}{pn} \approx 2^k \cdot 2^{nH(p)+o(1)}$$

(by Eq. (1) of Theorem 4.6). Noting now that $k = n - nH(p) + n\delta$, we obtain that

$$2^k \cdot \binom{n}{pn} \approx 2^{n-nH(p)+n\delta+nH(p)+o(1)} = 2^{n+n\delta+o(1)} > 2^n$$

but there are only 2^n possible strings in the range, and so this is not possible. ■

5 Constructing Codes from Other Codes

In this section, we will see a few rules for constructing codes from other codes. We will then present the Reed-Muller code which is obtained by applying such rules iteratively to an initial trivial code. We remark that an important method for constructing codes from other codes, called *code concatenation*, is not covered in this section but will be studied later. Before we begin, recall that we have already seen such a rule. Specifically, given any $[n, k, d]$-code where d is odd, we have seen how to obtain an $[n+1, k, d+1]$-code.

5.1 General Rules for Construction

Theorem 5.1 *Assume that there exists a linear $[n, k, d]$-code C over $\mathbb{F}_q$. Then:*

1. Extension: *There exists a linear $[n+r, k, d]$-code over $\mathbb{F}_q$ for every $r \geq 1$.*
2. Puncturing: *There exists a linear $[n-r, k, d-r]$-code over $\mathbb{F}_q$ for every $1 \leq r \leq d-1$.*
3. *There exists a linear $[n, k, d-r]$-code over $\mathbb{F}_q$ for every $1 \leq r \leq d-1$.*
4. Subcode: *There exists a linear $[n, k-r, d]$-code over $\mathbb{F}_q$ for every $1 \leq r \leq k-1$.*
5. *There exists a linear $[n-r, k-r, d]$-code over $\mathbb{F}_q$ for every $1 \leq r \leq k-1$.*

Proof:

1. Define
$$C' = \{(c_1, \ldots, c_n, 0^r) \mid (c_1, \ldots, c_n) \in C\}$$
It is clear that C' is a linear $[n+r, k, d]$-code.

2. Let $c \in C$ be a codeword with $\mathsf{wt}(c) = d$. Let I be the indices of r coordinates in which c has nonzero values. Then, erase the coordinates of I from all codewords (essentially, this is achieved by erasing columns in G). The resulting code is a linear $[n-r, k]$-code; the dimension is unchanged because $d > r$ and so the number of codewords remains unchanged. Furthermore $d(C') = d - r$ because there exists a codeword with weight $d - r$ (there cannot be a word with any less weight because this would imply that $d(C) < d$).

3. First apply (2) and then (1).

4. Let $\{c_1, \ldots, c_k\}$ be a basis of C. Let C' be the code with basis $\{c_1, \ldots, c_{k-r}\}$. It is clear that C' is a linear $[n, k-r]$-code. In addition, $d(C') \geq d(C)$ since $C' \subseteq C$. To guarantee that $d(C') = d$ exactly, we apply item (3) of the theorem.

5. If $k = n$ then $d = 1$ and so we take the code $\mathbb{F}_q^{n-r}$. If $k < n$ then we show the existence of a linear $[n-r, k-r, d]$-code for $k \geq r+1$. Denote the parity check matrix of C by $H = (I_{n-k} \mid X)$ and erase the last r columns of H. The result is a matrix H_1 of dimensions $(n-k) \times (n-r)$ with linearly independent rows (this holds because X has k columns and $k > r$; thus we did not erase any part of I_{n-k} from H). In addition, every $d-1$ columns in H_1 are linearly independent as in H and so d is not reduced. Thus, we have obtained a linear $[n-r, k-r, d']$-code where $d' \geq d$. Item (4) can be used to reduce d' to d if necessary.

■

We remark that items (4) and (5) are the more interesting cases in the above theorem. We now present the *direct sum* construction.

Theorem 5.2 (direct sum): *Let C_i be a linear $[n_i, k_i, d_i]$-code over $\mathbb{F}_q$ for $i \in \{1, 2\}$. The direct sum of C_1 and C_2, defined by*

$$C_1 \oplus C_2 = \{(c_1, c_2) \mid c_1 \in C_1, c_2 \in C_2\}$$

is a linear $[n_1 + n_2, k_1 + k_2, \min(d_1, d_2)]$-code.

Proof: It is easy to see that $C_1 \oplus C_2$ is a linear $[n_1 + n_2, k_1 + k_2]$-code (the length is immediate and the dimensions is due to the fact that there are $|C_1| \cdot |C_2|$ distinct codewords). Concerning distance, notice that if $d_1 \leq d_2$ then the word with smallest weight is $(c, 0)$ where $c \in C_1$ where $\mathsf{wt}(c) = d_1$. (More formally, if $(c_1, c_2) \neq 0$ then one of them is not zero and so $\mathsf{wt}(c_1, c_2) \geq \min(d_1, d_2)$.) ∎

Home exercise: write the generator matrix for $C_1 \oplus C_2$.

The disadvantage of the direct sum construction is that the distance is not increased at all. In the next construction, this is improved:

Theorem 5.3 *Let C_i be a linear $[n, k_i, d_i]$-code over $\mathbb{F}_q$ for $i = 1, 2$. Then the code C defined by*

$$C = \{(u, u + v) \mid u \in C_1, v \in C_2\}$$

is a linear $[2n, k_1 + k_2, \min\{2d_1, d_2\}]$-code.

Proof: It is easy to see that C is a linear code with length $2n$. In addition, the mapping $f(C_1, C_2) \to C$ defined by $f(c_1, c_2) = (c_1, c_1 + c_2)$ is a bijection (i.e., it is 1–1 and onto). Therefore $\dim(C) = \dim(f(C_1, C_2)) = k_1 + k_2$ (because there are $|C_1| \cdot |C_2|$ words in C). It remains to show that $d(C) = \min\{2d_1, d_2\}$.

Let $c = (c_1, c_1 + c_2) \in C$ be a nonzero codeword. Since c is nonzero, either $c_1 \neq 0$ or $c_2 \neq 0$ or both. There are two cases:

1. *Case 1 – $c_2 = 0$:* In this case, $c_1 \neq 0$ and so

$$\mathsf{wt}(c_1, c_1 + c_2) = \mathsf{wt}(c_1, c_1) = 2\mathsf{wt}(c_1) \geq 2d_1 \geq \min\{2d_1, d_2\}$$

2. *Case 2 – $c_2 \neq 0$:* In this case,

$$\mathsf{wt}(c_1, c_1 + c_2) = \mathsf{wt}(c_1) + \mathsf{wt}(c_1 + c_2) \geq \mathsf{wt}(c_1) + (\mathsf{wt}(c_2) - \mathsf{wt}(c_1)) = \mathsf{wt}(c_2) \geq d_2 \geq \min\{2d_1, d_2\}$$

where the first inequality is by Lemma 2.9. Thus, we have shown that $d(C) \geq \min\{2d_1, d_2\}$. In order to obtain equality, let $c_1 \in C_1$ and $c_2 \in C_2$ be such that $\mathsf{wt}(c_1) = d_1$ and $\mathsf{wt}(c_2) = d_2$. Next, observe that the words (c_1, c_1) and $(0, c_2)$ are in C and thus for $d(C) = \min\{2d_1, d_2\}$.

∎

Corollary 5.4 *Let $\overline{x}$ denote the bitwise complement of a vector x, and let A be a binary linear $[n, k, d]$-code. Then the code*

$$C = \Big\{(c, c), (c, \overline{c}) \mid c \in A\Big\}$$

is a binary linear $[2n, k + 1, \min\{2d, n\}]$-code.

Proof: Take $C_1 = A$ and $C_2 = \{0^n, 1^n\}$. C_1 is an $[n, k, d]$-code and C_2 is an $[n, 1, n]$-code. Furthermore C is the code obtained by C_1 and C_2 via the construction in Theorem 5.3 (observe that $c + 0^n = c$ and $c + 1^n = \overline{c}$). Thus, C is a linear $[2n, k + 1, \min\{2d, n\}]$-code. ∎

We remark that Corollary 5.4 provides us a way to *double* all parameters (the code length, its size, and its length).

5.2 The Reed-Muller Code

The Reed-Muller code is obtained by applying the construction of Theorem 5.3 to a trivial initial code.

Definition 5.5 Reed-Muller codes of the first degree, *denoted $R(1,m)$, are binary linear codes defined for $m \geq 1$ recursively as follows:*

1. *R(1,1)=*$\mathbb{F}_2^2 = \{00, 01, 10, 11\}$
2. *For every $m \geq 1$,* $R(1, m+1) = \Big\{(u,u), (u, \overline{u}) \mid u \in R(1,m)\Big\}$

Proposition 5.6 *For every $m \geq 1$, the code $R(1,m)$ is a linear $[2^m, m+1, 2^{m-1}]$-code in which every codeword except 0^n and 1^n (where $n = 2^m$) has weight 2^{m-1}.*[3]

Proof: It is clear that $R(1,1)$ is a linear $[2,2,1]$-code. We will prove the general case by induction. Assume that $R(1, m-1)$ is a linear $[2^{m-1}, m, 2^{m-2}]$-code. By Corollary 5.4, we have that $R(1,m)$ is a linear $[2 \cdot 2^{m-1}, m+1, 2 \cdot 2^{m-2}] = [2^m, m+1, 2^{m-1}]$-code.

We now show that every codeword except $0^{2^m}, 1^{2^m}$ has weight 2^{m-1}; the proof is by induction. For $R(1,1)$ the only words apart from $00, 11$ are $01, 10$ which are both of weight $1 = 2^{1-1}$, as required. Assume that this holds for $R(1,m)$; we prove it for $R(1, m+1)$. Every word in $R(1, m+1)$ is either of the form (u,u) or $(u, \overline{u})$ for $u \in R(1,m)$. Let $c \in R(1, m+1)$ be a codeword. There are two cases:

1. *Case 1 – c is of the form $c = (u,u)$:* $u \neq 0^{2^{m-1}}$ and $u \neq 1^{2^{m-1}}$ since otherwise $c \in \{0^{2^m}, 1^{2^m}\}$. Thus by the inductive assumption $\mathsf{wt}(u) = 2^{m-2}$, implying that $\mathsf{wt}((u,u)) = 2 \cdot 2^{m-2} = 2^{m-1}$, as required.
2. *Case 1 – c is of the form $c = (u, \overline{u})$:*
   (a) If $u \notin \{0^{2^{m-1}}, 1^{2^{m-1}}\}$ then as before $\mathsf{wt}(u) = \mathsf{wt}(\overline{u}) = 2^{m-2}$ ($\mathsf{wt}(u) = \mathsf{wt}(\overline{u})$ because exactly half of the coordinates equal 1 by the assumption). Hence $\mathsf{wt}((u, u+\overline{1})) = 2 \cdot 2^{m-2} = 2^{m-1}$, as required.
   (b) If $u = 0^{2^{m-1}}$ then $c = (0^{2^{m-1}}, 1^{2^{m-1}})$ and so $\mathsf{wt}(c) = 2^{m-1}$.
   (c) If $u = 1^{2^{m-1}}$ then $\mathsf{wt}(c) = \mathsf{wt}(1^{2^{m-1}}, 0^{2^{m-1}}) = 2^{m-1}$.

■

Proposition 5.7 *The matrix $G = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$ is a generator matrix for $R(1,1)$. If G_m is a generator matrix for $R(1,m)$ then $G = \begin{pmatrix} G_m & G_m \\ 0 \ldots 0 & 1 \ldots 1 \end{pmatrix}$ is a generator matrix for $R(1, m+1)$.*

Proof: The case of $R(1,1)$ is immediate. Regarding $R(1, m+1)$, observe that every codeword is either a linear combination of all the rows except for the last or a linear combination of all the rows including the last. In the former case, a codeword is of the form (u,u) for $u \in R(1,m)$, and in the latter case a codeword is of the form $(u, \overline{u})$ for $u \in R(1,m)$. This is therefore identical to the recursive definition given above. ■

Reed-Muller and Hamming codes. We will now prove an interesting claim stating that the dual code of $R(1,m)$ is equivalent to the extended Hamming code. The proof of this is really an exercise, but we present it in class in any case.

Definition 5.8 *The* extended Hamming code *is obtained by preceding each codeword with a parity bit, and is a linear $[2^r, 2^r - 1 - r, 4]$-code. We denote this code by $H_e(r)$.*

The parity-check matrix of $H_e(r)$ is constructed by taking the parity-check matrix of $Ham(r,2)$ and adding a column of all zeroes and *then* adding a row of all ones.

Proposition 5.9 *For every $r \geq 1$, the extended Hamming code $H_e(r)$ is the dual code of $R(1,r)$.*

[3] For $n = 2^m$ we have that $R(1,m)$ is a linear $[n, \log n + 1, \frac{n}{2}]$ code.

Proof Sketch: It suffices to show that the generator matrix of $R(1, r)$ is a parity check matrix of $H_e(r)$. This can be seen by the recursive definition of the generator matrix of $R(1, r)$ given above. Specifically, the parity check matrix of $H_e(1)$ is exactly the generator matrix of $R(1, r)$. Furthermore, the first row of G contains all ones and the first column of G (excluding the first coordinate) contains all zeroes. In addition, the submatrix obtained by removing the first row and column of G contains all the vectors of length $2^r - 1$ except for the all-zero vector. This is due to the fact that we start with $G = (0\ 1)$ and then add 0 and 1 to the end of all columns iteratively time. Thus, the columns contain all strings of length r and so the result is exactly the parity check matrix of $H_e(r)$. ■

6 Generalized Reed-Solomon Codes (GRS)

6.1 Definition

In this section we will define a family of codes, called *generalized Reed-Solomon codes* and we will show how to efficiently decode them.

Definition 6.1 *Let q be a prime power and let $\alpha_1, \ldots, \alpha_n$ be distinct non-zero values in $\mathbb{F}_q$. In addition, let $v_1, \ldots, v_n$ be non-zero values in $\mathbb{F}_q$ (not necessarily distinct). A* Generalized Reed Solomon code (GRS) *is a linear $[n,k]$-code over $\mathbb{F}_q$, denoted C_{GRS} defined by the following parity check matrix:*

$$H_{\text{GRS}} = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ \alpha_1 & \alpha_2 & \cdots & \alpha_n \\ \alpha_1^2 & \alpha_2^2 & \cdots & \alpha_n^2 \\ \vdots & \vdots & \cdots & \vdots \\ \alpha_1^{n-k-1} & \alpha_2^{n-k-1} & \cdots & \alpha_n^{n-k-1} \end{pmatrix} \begin{pmatrix} v_1 & & 0 \\ & \ddots & \\ 0 & & v_n \end{pmatrix} = \begin{pmatrix} v_1 & \cdots & v_n \\ v_1\alpha_1 & \cdots & v_n\alpha_n \\ \vdots & & \vdots \\ v_1\alpha_1^{n-k-1} & \cdots & v_n\alpha_n^{n-k-1} \end{pmatrix}$$

$\alpha_1, \ldots, \alpha_n$ are called the code locators *and $v_1, \ldots, v_n$ are called the* column multipliers.

Proposition 6.2 *H_{GRS} is a legal parity check matrix. In addition every code GRS is MDS, i.e., $d = n-k+1$.*

Proof: We first prove the distance. It suffices to show that the matrix with the α values is such that every subset of $n-k$ columns are linearly independent. (This is because multiplying the ith column with a value $v_i \neq 0$ does not change the distance of the code.) Now, every submatrix of $n-k$ columns is a square Vandermonde matrix of the form

$$B = \begin{pmatrix} 1 & 1 & 1 \\ \beta_1 & \beta_2 & \beta_{n-k} \\ \cdots & \cdots & \cdots \\ \beta_1^{n-k-1} & \beta_2^{n-k-1} & \beta_{n-k}^{n-k-1} \end{pmatrix}$$

for distinct values $\beta_1, \ldots, \beta_{n-k} \in \{\alpha_1, \ldots, \alpha_n\} \subseteq \mathbb{F}_q$. Note that this is true for *every* submatrix of $n-k$ columns.

From linear algebra, we know that the determinant of a square Vandermonde matrix is given by the following formula:

$$\det(B) = \prod_{1 \leq i < j \leq n-k} (\beta_j - \beta_i)$$

Since $\beta_i \neq \beta_j$ for all $i \neq j$ we have that determinant is non-zero and so B is invertible. Thus, its rows and columns are linearly independent.

We conclude that every $n-k$ columns in H_{GRS} are linearly independent and hence $d(C_{\text{GRS}}) \geq n-k+1$; equality holds from the Singleton bound. The above also proves that the rows of H_{GRS} are linearly independent and thus H_{GRS} is a legal parity check matrix, as required. ■

There are a number of types of GRS codes:

1. *Primitive GRS code:* This is the case that $n = q-1$ and so $\{\alpha_1, \ldots, \alpha_n\} = \mathbb{F}_q \setminus \{0\}$
2. *Normalized GRS code:* This is the case that $v_1 = v_2 = \cdots = v_n = 1$
3. *Narrow-sense GRS code:* This is the case that $\alpha_j = v_j$ for every j
4. *Singly extended GRS code:* This is the case when one of the code locators equals 0 (note that this itself is *not* a GRS code).

Claim 6.3 *The dual code of a GRS code with parameters $[n,k]$ where $k < n$ is a GRS code with parameters $[n, n-k]$. In addition, the two codes can be defined with the same code locators.*

Proof: Let C_{GRS} be a GRS $[n,k]$-code over $\mathbb{F}_q$ with and let H_{GRS} be the parity check matrix as defined above. Consider the following matrix G_{GRS}:

$$G_{\text{GRS}} = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ \alpha_1 & \alpha_2 & \cdots & \alpha_n \\ \vdots & \vdots & & \vdots \\ \alpha_1^{k-1} & \alpha_2^{k-1} & \cdots & \alpha_n^{k-1} \end{pmatrix} \begin{pmatrix} v_1' & & 0 \\ & \ddots & \\ 0 & & v_n' \end{pmatrix}$$

where for every i, $v_i' \in \mathbb{F}_q$. We will prove that there is a choice of the v_i' values such that $G_{\text{GRS}} \cdot H_{\text{GRS}}^T = 0$. This will imply that $C_{\text{GRS}}^{\perp}$ is a GRS code with the same code locators (because G_{GRS} generates C_{GRS} and thus by definition it is the parity check matrix of $C_{\text{GRS}}^{\perp}$; furthermore G_{GRS} is a parity-check matrix of the GRS form and it has code locators $\alpha_1, \ldots, \alpha_n$).

For every $i = 0, \ldots, k-1$ and $\ell = 0, \ldots, n-k-1$ we need to show that the product of the ith row in G_{GRS} with the ℓth column in H_{GRS}^T equals 0 (note that the ℓth column of H_{GRS}^T is the ℓth row of H_{GRS}). Thus, we need to show that

$$\sum_{j=1}^{n} \left(v_j' {\alpha_j}^i\right) \cdot \left(v_j {\alpha_j}^\ell\right) = \sum_{j=1}^{n} v_j' v_j \alpha_j^{\ell+i} = 0$$

Since $0 \leq i + \ell \leq n-2$ this is equivalent to showing that for every $0 \leq r \leq n-2$:

$$\sum_{j=1}^{n} v_j' v_j {\alpha_j}^r = 0$$

In matrix form, we have that the following needs to hold:

$$(v_1' \ldots v_n') \begin{pmatrix} v_1 & v_1\alpha_1 & \ldots & v_1\alpha_1^{n-2} \\ \vdots & \vdots & & \vdots \\ v_n & v_n\alpha_n & \ldots & v_n\alpha_n^{n-2} \end{pmatrix} = (0 \ \cdots \ 0)$$

However, note that the matrix in the middle is exactly the transpose of a parity check matrix H for a GRS code with $k = 1$ (this is because $H^T \in \mathbb{F}_q^{n \times (n-1)}$ and so $H \in \mathbb{F}_q^{(n-1 \times n}$ implying that $k = 1$). From the theorem we have already proven, this is a $[n, 1, n]$-code. Now, for every vector $v' = (v_1', \ldots, v_n')$ that is a codeword we have that $v' \cdot H^T = 0$. Since there exist nonzero codewords in this code, it follows that there exist values $v_1', \ldots, v_n'$ that are all nonzero (this holds because $d = n$ in this code) and such that $v' \cdot H^T = 0$. We conclude that for these v_i' values, the matrix G_{GRS} that we defined is indeed a parity check matrix for a GRS code that is the dual code of the code C_{GRS} that we started with. ■

6.2 Polynomial representation of GRS

Let C_{GRS} be a GRS code with parameters $[n, k, d]$. We saw that the generator matrix of C_{GRS} is

$$G_{\text{GRS}} = \begin{pmatrix} 1 & \ldots & 1 \\ \alpha_1 & & \alpha_n \\ \vdots & & \vdots \\ \alpha_1^{k-1} & \ldots & \alpha_n^{k-1} \end{pmatrix} \begin{pmatrix} v_1' & & 0 \\ & \ddots & \\ 0 & & v_n' \end{pmatrix}$$

for some column multipliers $v_1', \ldots, v_n'$. As usual, we encode a word $m \in \mathbb{F}_q^k$ by computing $C(m) = m \cdot G_{\text{GRS}}$. Denote $m = m_0, \ldots, m_{k-1}$. Then the ith coordinate in the vector $m \cdot G_{\text{GRS}}$ is

$$v_i' \cdot \sum_{j=0}^{k-1} m_j \cdot \alpha_i^j = v_i' \cdot (m_0 + m_1\alpha_i + m_2\alpha_i^2 + \ldots + m_{k-1}\alpha_i^{k-1})$$

If we define a polynomial $m(x) = m_0 + m_1 x + m_2 x^2 + \ldots + m_{k-1} x^{k-1}$ it follows that the ith element in the vector $m \cdot G_{\text{GRS}}$ is exactly $v'_i \cdot m(\alpha_i)$ and so

$$m \cdot G_{\text{GRS}} = (v'_1 m(\alpha_1), \ldots, v'_n m(\alpha_n)).$$

This yields an alternative way of proving that $d(C_{\text{GRS}}) = n - k + 1$; namely, the way that we proved the distance of the classical Reed-Solomon code.

Classical Reed-Solomon: The basic/classical Reed-Solomon code is simply a GRS code for which all the column multipliers equal 1. (We called this a normalized GRS code above.) We remark that although this is the way we defined Reed-Solomon, most define it by taking a *primitive element* α and setting $\alpha_i = \alpha^i$ for all i.

6.3 Decoding GRS Codes

Let C_{GRS} be a GRS code with code locators $\alpha_1, \ldots, \alpha_n$ and column multipliers $v_1, \ldots v_n$ (and so all these values are nonzero). We will describe an algorithm that corrects up to $\lfloor \frac{d-1}{2} \rfloor$ errors. Denote by $e = (e_1, \ldots, e_n)$ vector of errors and by J the set of indices of the errors. Stated differently, $J = \{i \mid e_i \neq 0\}$. We work under the assumption that $|J| \leq \lfloor \frac{d-1}{2} \rfloor$.

6.3.1 Step 1: Computing the Syndrome

Let $y \in \mathbb{F}_q^n$ be the received word. The syndrome of y is given by $S_H(y) = y \cdot H^T$. We denote $S_H(y) = (S_0, \ldots, S_{d-2})$. Then, for every ℓ $(0 \leq \ell \leq d-2)$ we have

$$S_\ell = \sum_{j=1}^{n} y_j v_j \alpha_j^\ell$$

(because the ℓth column of H^T is $(v_1 \alpha_1^\ell, \ldots, v_n \alpha_n^\ell)$).

Definition 6.4 *The syndrome polynomial, denoted $S(x)$, is defined by*

$$S(x) = \sum_{\ell=0}^{d-2} S_\ell \cdot x^\ell$$

Observe that the polynomial is defined for a given y; it is not a general polynomial for the code. As we saw in syndrome decoding $S_H(y) = S_H(e)$ (this holds for all linear codes), and so

$$S_\ell = \sum_{j=1}^{n} e_j v_j \alpha_j^\ell = \sum_{j \in J} e_j v_j \alpha_j^\ell$$

This holds because $e_j = 0$ if $j \notin J$ (by definition, if J is empty then the sum equals 0). From here we have:

$$S(x) = \sum_{\ell=0}^{d-2} \left(\sum_{j \in J} e_j v_j \alpha_j^\ell \right) \cdot x^\ell = \left(\sum_{j \in J} e_j v_j \right) \sum_{\ell=0}^{d-2} (\alpha_j \cdot x)^\ell$$

Consider the polynomial ring $\mathbb{F}_q[x]/x^{d-1}$ (the polynomial ring of the remainders after dividing by x^{d-1})). The elements of this ring are just all polynomials of degree smaller than $d-1$ over $\mathbb{F}_q$. For example, take $d = 4$ and $p(x) = x^5 + 4x^4 + 2x^3 + x^2 + 5x + 7$. We have that $p(x) = q(x) \cdot x^3 + r(x)$ for $q(x) = x^2 + 4x + 2$ and $r(x) = x^2 + 5x + 7$. Thus, all of the higher-degree elements just disappear.

Now, the elements in the ring that are not divisible by $p(x) = x$ form a multiplicative group. This is because they are relatively prime to x^{d-1} (i.e., their greatest common divisor is 1), and so they all have an inverse.

The polynomial $p(x) = 1 - \alpha_j x$ is not divisible by x and its inverse is:

$$\sum_{j=0}^{d-2} (\alpha_j x)^\ell$$

This is due to the fact that

$$(1 - \alpha_j x) \sum_{\ell=0}^{d-2} (\alpha_j x)^\ell = \sum_{\ell=0}^{d-2} (\alpha_j x)^\ell - \sum_{\ell=0}^{d-2} (\alpha_j x)^{\ell+1}.$$

This is a telescopic sum and so it is equal to $1 - (\alpha_j x)^{d-1}$. However, $1 - (\alpha_j x)^{d-1} \equiv 1 \mod x^{d-1}$ and so $\sum_{j=0}^{d-2} (\alpha_j x)^\ell$ is indeed the inverse of $1 - \alpha_j x$.

We conclude this portion by combining the above into an alternative definition of the syndrome polynomial. Namely, we conclude that:

$$S(x) \equiv \sum_{j \in J} \frac{e_j v_j}{1 - \alpha_j x} \mod x^{d-1}$$

Observe that we have succeeded in *reducing* the degree of $S(x)$ from $d-2$ to $|J| \leq \frac{d-1}{2}$. This is important because we are interested in constructing a system of linear equations and we need the number of variables (which include the e_j values) to be less than or equal to the number of equations.

6.3.2 Step 2 – The Key Equation of GRS Decoding

We define the error-locator polynomial Λ (Lambda) by

$$\Lambda(x) = \prod_{j \in J} (1 - \alpha_j x)$$

and the error evaluator polynomial Γ (Gamma) by

$$\Gamma(x) = \sum_{j \in J} e_j v_j \prod_{m \in J \setminus \{j\}} (1 - \alpha_m x)$$

We remark that by definition, the product of an empty set equals 1. It holds that for every k, $\Lambda(\alpha_k^{-1}) = 0$ *if and only if* $k \in J$ (recall that all α_j values are nonzero). Thus, the roots of the polynomial Λ are the values $\{\alpha_k^{-1} \mid k \in J\}$. For this reason, Λ is called the error-locator polynomial (its roots indicate *where* the errors are). In addition, for every $k \in J$

$$\begin{aligned} \Gamma(\alpha_k^{-1}) &= \sum_{j \in J} e_j v_j \prod_{m \in J \setminus \{j\}} (1 - \alpha_m \alpha_k^{-1}) \\ &= e_k v_k \prod_{m \in J \setminus \{k\}} (1 - \alpha_m \alpha_k^{-1}) \\ &\neq 0 \end{aligned}$$

where the second equality holds because for every $j \neq k$ the term $(1 - \alpha_k \alpha_k^{-1}) = 0$ appears in the product, and the inequality is due to the fact that all α_i are different.

We conclude that the roots of Λ are exactly $\{\alpha_k^{-1}\}_{k \in J}$ and these are all *not* roots of Γ. Since Λ and Γ have no common roots, there is no polynomial that divides both. Thus,

$$\gcd(\Lambda(x), \Gamma(x)) = 1 \tag{4}$$

(The fact that there are no common roots suffices for concluding that the greatest common divisor is 1 because Λ has degree $|J|$ and $|J|$ roots. Thus, there certainly cannot be any irreducible polynomials of degree greater than 1 that divides Λ.) In addition, $\deg(\Lambda) = |J|$ and $\deg(\Gamma) < |J|$ and thus

$$\deg(\Gamma) < \deg(\Lambda) \leq \frac{d-1}{2} \tag{5}$$

where the second inequality is due to the assumption that there are at most $\frac{d-1}{2}$ errors.

Observe that:

$$\frac{\prod_{m \in J \setminus \{j\}}(1 - \alpha_m x)}{\prod_{j \in J}(1 - \alpha_j x)} = \frac{1}{1 - \alpha_j x}$$

and thus

$$\begin{aligned}
\Gamma(x) &= \sum_{j \in J} e_j v_j \prod_{m \in J \setminus \{j\}} (1 - \alpha_m x) \\
&= \Lambda(x) \cdot \sum_{j \in J} e_j v_j \frac{\prod_{m \in J \setminus \{j\}}(1 - \alpha_m x)}{\Lambda(x)} \\
&= \Lambda(x) \cdot \sum_{j \in J} e_j v_j \frac{\prod_{m \in J \setminus \{j\}}(1 - \alpha_m x)}{\prod_{j \in J}(1 - \alpha_j x)} \\
&= \Lambda(x) \cdot \sum_{j \in J} e_j v_j \frac{1}{1 - \alpha_j x}
\end{aligned}$$

Since

$$S(x) \equiv \sum_{j \in J} \frac{e_j v_j}{1 - \alpha_j x} \mod x^{d-1}$$

we conclude that

$$\Lambda(x) S(x) \equiv \Gamma(x) \mod x^{d-1} \tag{6}$$

The above three equations, Equations (4), (5) and 6), are called the key equations for GRS decoding.

The next step in GRS decoding is to solve the equations in order to find Λ and Γ. Given Λ we can find for which values α_i it holds that $\Lambda(\alpha_i^{-1}) = 0$ and thus we can find J, the set of error locations. Then, given these values, it is possible to solve the linear system of equations given by

$$S_\ell = \sum_{j \in J} e_j v_j \alpha_j^\ell$$

(Given the set J we obtain $\frac{d-1}{2}$ equations and $\frac{d-1}{2}$ variables. This is because the values S_ℓ are given, as are v_j and α_j. Thus, only e_j is unknown.)

6.3.3 Step III - Solving the Key Equations

We will present an algorithm for solving the system of equations presented above. This algorithm is not the most efficient possible. Nevertheless, our aim is more to just show that decoding can be carried out in polynomial time.

Let $\tau = \left\lfloor \frac{d-1}{2} \right\rfloor$. Then we can write

$$\Lambda(x) = \sum_{m=0}^{\tau} \Lambda_m x^m \qquad \text{and} \qquad \Gamma(x) = \sum_{m=0}^{\tau-1} \Gamma_m x^m$$

We denote the variables by $\{\gamma_m\}_{m=0}^{\tau-1}$ and $\{\lambda_m\}_{m=0}^{\tau}$. Then the values $\{\Gamma_m\}_{m=0}^{\tau-1}$ and $\{\Lambda_m\}_{m=0}^{\tau}$ (that are the coefficients of the polynomials Γ and Λ) are a solution to the following system of equations:

$$\begin{pmatrix} S_0 & 0 & 0 & \cdots & 0 & 0 \\ S_1 & S_0 & 0 & \cdots & 0 & 0 \\ S_2 & S_1 & S_0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ S_{\tau-1} & S_{\tau-2} & S_{\tau-3} & \cdots & S_0 & 0 \\ \\ S_\tau & S_{\tau-1} & S_{\tau-2} & \cdots & S_1 & S_0 \\ S_{\tau+1} & S_\tau & S_{\tau-1} & \cdots & S_2 & S_1 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ S_{d-2} & S_{d-3} & S_{d-4} & \cdots & S_{d-\tau-1} & S_{d-\tau-2} \end{pmatrix} \cdot \begin{pmatrix} \lambda_0 \\ \lambda_1 \\ \vdots \\ \lambda_\tau \end{pmatrix} = \begin{pmatrix} \gamma_0 \\ \gamma_1 \\ \vdots \\ \gamma_{\tau-1} \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

In order to see that this is the case, recall that $\Lambda(x)S(x) = \Gamma(x) \mod x^{d-1}$, and so

$$\sum_{k=0}^{\tau-1} \Gamma_k x^k = \sum_{m=0}^{\tau} \Lambda_m x^m \sum_{\ell=0}^{d-2} S_\ell x^\ell = \sum_{m=0}^{\tau} \sum_{\ell=0}^{d-2} \Lambda_m S_\ell x^{m+\ell} \bmod x^{d-1}$$

We now look at each coefficient of the polynomial equivalence separately, for $k = 0$ to $\tau - 1$:

1. $k = 0$: In this case, $\ell = m = 0$ and so $\Gamma_0 = S_0 \cdot \Lambda_0 \mod x^{d-1}$
2. $k = 1$: In this case, we have $\ell = 1, m = 0$ and $\ell = 0, m = 1$. Therefore, $\Gamma_1 = S_0\Lambda_1 + S_1\Lambda_0 \mod x^{d-1}$
3. $k = 2$: In this case, $\Gamma_2 = S_0\Lambda_2 + S_1\Lambda_1 + S_2\Lambda_0 \mod x^{d-1}$
4. We continue in the same way. Note that for $k \geq \tau$, we always obtain 0 because the degree of Γ is $\tau - 1$.

Observe that these equations are exactly what appear in the equations in the system presented above. Thus, $\{\Lambda_m\}$ and $\{\Gamma_m\}$ constitute a solution.

Now, this system of equations has $2\tau - 1 = d - 2$ variables and $d - 1$ equations. We also know that it has a solution (because we have seen one). However, there may be many solutions (because the equations are not linearly independent). Now, the $d-1-\tau$ last equations (and there are at least τ of these) involve the variables $\lambda_0, \ldots, \lambda_\tau$ only. Thus, it is possible to solve these and every possible solution defines a unique solution for $\gamma_0, \ldots, \gamma_{\tau-1}$.[4] (Once we are given a solution for $\lambda_0, \ldots, \lambda_\tau$ we remain with τ variables $\gamma_0, \ldots, \gamma_{\tau-1}$ and τ equations being the first τ which are already diagonalized. Since the number of variables equals the number of equations, the solution is unique.)

In addition to the above, every solution to the system of equations fulfills that

$$\lambda(x)S(x) = \gamma(x) \mod x^{d-1}$$

(The mod does not have any effect because λ and γ zero all coefficients of degree greater than τ and therefore equality is the same as modular equivalence.)

It remains to show every solution is beneficial to us (since we want the original Λ and Γ it is not enough for us to find any solution).

Claim 6.5 *Let $\lambda(x)$ and $\gamma(x)$ be polynomials over $\mathbb{F}_q$ such that $\deg(\gamma) < \deg(\lambda) \leq \frac{d-1}{2}$. Then*

1. *$\lambda(x)$ and $\gamma(x)$ fulfill $\lambda(x)S(x) = \gamma(x) \mod x^{d-1}$ if and only if there exists a polynomial $c(x) \in \mathbb{F}_q[x]$ such that $\gamma(x) = c(x)\Gamma(x)$ and $\lambda(x) = c(x)\Lambda(x)$.*

[4] We know that it is possible to solve for the variables $\lambda_0, \ldots, \lambda_\tau$ because we already know of one solution.

2. *The solution $\lambda(x)$ and $\gamma(x)$ that fulfills the degree requirements and the equation $\lambda(x)S(x) = \gamma(x) \bmod x^{d-1}$ with $\lambda(x) \neq 0$ of smallest possible degree (over all solutions) is unique up to scaling by a nonzero constant $c \in \mathbb{F}_q$ and is given by $\lambda(x) = c \cdot \Lambda(x)$, $\gamma(x) = c \cdot \Gamma(x)$.*[5]

3. *The solution from item (2) is the only solution for which* $\gcd(\lambda(x), \gamma(x)) = 1$.[6]

Proof: Assume that $\lambda(x)$ and $\gamma(x)$ fulfill $\lambda(x)S(x) = \gamma(x) \mod x^{d-1}$. By the definition of $\Lambda(x)$ we have that $\Lambda(0) = 1$. Therefore, $\Lambda(x)$ has no common roots with the polynomial x^{d-1}, implying that $\Lambda(x)$ has an inverse in the ring $\mathbb{F}_q[x]/x^{d-1}$ (recall that an element has an inverse if and only if it is relatively prime to the modulus). Thus,

$$S(x) = \Gamma(x) \left(\Lambda(x)\right)^{-1} \mod x^{d-1}$$

Since $\lambda(x)$ and $\gamma(x)$ fulfill $\lambda(x)S(x) = \gamma(x) \mod x^{d-1}$ it follows that

$$\lambda(x)\Gamma(x)(\Lambda(x))^{-1} = \gamma(x) \mod x^{d-1}$$

and so

$$\lambda(x)\Gamma(x) = \gamma(x)\Lambda(x) \mod x^{d-1}. \tag{7}$$

Since $\deg(\gamma) < \deg(\lambda) \leq \frac{d-1}{2}$, all of the polynomials in Eq. (7) have degree less than $d-1$ and so the modular reduction has no effect. We therefore have:

$$\lambda(x)\Gamma(x) = \gamma(x)\Lambda(x) \tag{8}$$

Now, since $\gcd(\Lambda(x), \Gamma(x)) = 1$ it follows that $\Lambda(x)$ divides $\lambda(x)$. Stated differently, there exists a polynomial $c(x) \in \mathbb{F}_q[x]$ such that

$$\lambda(x) = c(x)\Lambda(x).$$

By Eq. (8) this in turn implies that

$$\gamma(x) = c(x)\Gamma(x)$$

concluding this direction of the proof of item (1). For the other direction, if there exists such a $c(x)$ then

$$\lambda(x)S(x) = c(x)\Lambda(x)S(x) = c(x)\Gamma(x) \mod x^{d-1}$$

where the second equality is by Eq. (6). But

$$c(x)\Gamma(x) = \gamma(x) \mod x^{d-1}$$

and so

$$\lambda(x)S(x) = \gamma(x) \mod x^{d-1}$$

as required.

Item (2) is derived by dividing $\lambda(x)$ and $\gamma(x)$ by the polynomial $c(x)$ of the greatest degree guaranteed to exist by item (1); multiplication by a constant makes no difference to the degree of the polynomial. Item (3) follows immediately from (1). ∎

[5] The degree of λ determines the number of errors and we are therefore interested in the solution with the minimum degree.

[6] This tells us how to find the solution with the smallest degree for $\lambda(x)$; take any solution and divide $\lambda(x)$ by its greatest common divisor with $\gamma(x)$. Note: the greatest common divisor of two polynomials is defined to the monic polynomial of the highest degree that divides both polynomials.

6.3.4 The Peterson-Gorenstein-Zierler GRS Decoding Algorithm

Input: a word $(y_1, \ldots, y_n) \in \mathbb{F}_q^n$

Output: an error vector $(e_1, \ldots, e_n) \in \mathbb{F}_q^n$

The algorithm:

1. Compute the syndrome $S_H(y)$ and obtain $S_0, \ldots, S_{d-2}$
2. Solve the system of equations defined by the $d-1-\tau$ equations of the overall system in order to obtain a polynomial $\lambda(x)$ with coefficients $\lambda_0, \ldots, \lambda_\tau$
3. Use $\lambda(x)$ and the system of equations to compute the polynomial $\gamma(x)$ and define

$$\Lambda(x) = \frac{\lambda(x)}{\gcd(\lambda(x), \gamma(x))}$$

4. Given $\Lambda(x)$ find the roots that define the set J (this is achieved by just computing $\Lambda(\alpha_1^{-1}), \ldots, \Lambda(\alpha_n^{-1})$).
5. Finally, solve the system of equations defined by

$$S_\ell = \sum_{j \in J} e_j v_j \alpha_j^\ell$$

for $\ell = 0, \ldots, d-2$, where the variables are $\{e_j\}_{j \in J}$. (Note that there are $d-1$ equations and $\frac{d-1}{2}$ variables.)

Complexity: Solving a system of d equations takes $O(d^3)$ time. There are much faster algorithms (e.g., $O(d^2)$ and $O(d \cdot |J|)$ but we will not see them in this course).

A remark about intuition. Although the math above is not overly intuitive, the algorithm itself is quite intuitive. Specifically, the system of equations is used to find the set J; once we know this set, we can solve the system of equations defined by the syndrome (without J this set of equations has n variables and only $d-1$ equations and so does not help). Thus, all of the work with Λ and Γ is dedicated to finding the set J. This is confusing because it appears that only Λ is needed for this. However, as we have seen, we need Γ in order to find the "correct" polynomial Λ. That is, solving the system of equations for λ alone does not suffice because there are many solutions $\lambda(x) = c(x) \cdot \Lambda(x)$. We therefore need also to find $\gamma(x)$ because this then gives us $\Lambda(x) = \frac{\lambda(x)}{\gcd(\lambda(x), \gamma(x))}$. Observe also that we may not actually find the polynomial $\Lambda(x)$ but rather $c \cdot \Lambda(x)$ for some scalar $c \in \mathbb{F}_q$. However, multiplying a polynomial by a scalar does not modify its roots, and we are only interested in the roots of Λ. This therefore suffices.

7 Asymptotically Good Codes

7.1 Background

The question that we address in this section is whether it is possible to (efficiently) construct a family of *binary* codes $\mathcal{C} = \{C_n\}_{n=1}^{\infty}$ with the following features:

1. There exists a constant $\epsilon > 0$ such that $R(\mathcal{C}) = \lim_{n\to\infty} R(C_n) = \epsilon$
2. There exists a constant $\delta > 0$ such that $\delta(\mathcal{C}) = \lim_{n\to\infty} \delta(C_n) = \delta$

That is, both the asymptotic rate and relative distance are simultaneously nonzero. More formally:

Definition 7.1 *A family of codes $\mathcal{C}$ is called* asymptotically good *if $R(\mathcal{C}) > 0$ and $\delta(\mathcal{C}) > 0$.*

The Gilbert-Varshamov bound tells us that asymptotically good codes exists (because there exists a code for every R and δ that have $R \leq 1 - H(\delta)$). However, we are interested in the efficient construction of such a code. Formally, a family of linear codes $\mathcal{C} = \{C_n\}$ is efficiently constructible if there exists a polynomial-time machine M such that for every n, $M(1^n)$ outputs a generator and parity-check matrix for C_n.[7] We remark that for many years it was assumed that it is *not* possible to efficiently construct such a code.

Hamming: The Hamming code has parameters $[2^r - 1, 2^r - 1 - r, 3]$ and so $R = \frac{k}{n} = 1 - \frac{r}{2^r-1} \to 1$ and $\delta = \frac{2}{n} \to 0$.

Reed-Muller: The Reed-Muller code has parameters $(n, 2n, \frac{n}{2})$ and so $R = \frac{\log 2n}{n} \to 0$ and $\delta = \frac{\frac{n}{2}-1}{n} \to \frac{1}{2}$.

GRS: GRS codes have parameters $[n, \frac{n}{2}, \frac{n}{2}+1]$ and so $R = \frac{1}{2}$ and $\delta = \frac{1}{2}$. However, GRS is not a binary code and has a very large alphabet. If we try to make it into a binary code directly (by encoding each letter of the alphabet using $\log q \geq \log n$ bits) then we obtain a code with parameters $[n \log q, \frac{n}{2} \log q, \frac{n}{2} + 1]$ (the distance remains the same as before). Thus, $R = \frac{1}{2}$ and $\delta = \frac{1}{\log n} \to 0$.

The main result of this section is a proof of the following theorem:

Theorem 7.2 *There exists an efficiently-constructible family of binary linear codes $\mathcal{C}$ that is asymptotically good, and has a polynomial-time decoding procedure.*

7.2 Concatenation of Codes

The first step in proving Theorem 7.2 is to show how it is possible to construct a code with a small alphabet from a code with a large alphabet (in a way that is better than the naive method mentioned above).

Theorem 7.3 *Let q be a prime power, let A be a linear $[N, K, D]$-code over $\mathbb{F}_{q^k}$ and let B a linear $[n, k, d]$-code over $\mathbb{F}_q$. Then there exists a linear code C over $\mathbb{F}_q$ with parameters $[nN, kK, d']$ such that $d' \geq dD$. A is called the* external code*, and B is called the* internal code*.*

Proof: We define a 1–1 and onto linear transformation $\phi : \mathbb{F}_{q^k} \to B$ as $\phi(u) = uG$ where G is the generator matrix of B. (Notice that we can look at $\mathbb{F}_{q^k}$ as $\mathbb{F}_q^k$ which is necessary for computing uG where $u \in \mathbb{F}_{q^k}$ rather than in $\mathbb{F}_q^k$.) It is clear that this transformation is indeed linear, and a bijection.

Next, we extend the transformation to $\phi^* : \mathbb{F}_{q^k}^N \to \mathbb{F}_q^{nN}$ by defining

$$\phi^*(u_1, \ldots, u_N) = (\phi(u_1), \ldots, \phi(u_N))$$

where each $u_i \in \mathbb{F}_{q^k}$. Note that ϕ^* is 1–1 (but not onto unless $k = n$). Finally, we define the code C to be $\phi^*(A)$. That is

$$C = \{\phi^*(v)\}_{v \in A}$$

[7] M receives n in unary form so that it can run in time that is polynomial in n.

We will now show that C is a linear $[nN, kK, d']$-code with $d' \geq dD$.

C is a subspace of $\mathbb{F}_q^{nN}$ because ϕ^* is a linear transformation from $\mathbb{F}_{q^k}^N$ to $\mathbb{F}_q^{nN}$. Therefore, C is a linear code with length nN. In addition

$$\dim(C) = \log_q |C| = \log_q |A|$$

where the second equality is due to the fact ϕ^* is one to one. Now, since A is a linear $[N, K, D]$-code over $\mathbb{F}_{q^k}$, it follows that $|A| = q^{kK}$ and thus $\dim(C) = \log_q q^{kK} = kK$. (If A is a vector space of dimension K over $\mathbb{F}_{q^k}$ then one can define an equivalent vector space of dimension kK over $\mathbb{F}_q$.)

It remains to prove that the distance of the code is $d' \geq dD$. Let $c \in C$ be a nonzero codeword and let $v = (v_1, \ldots, v_N) \in A$ be such that $c = \phi^*(v) = (\phi(v_1, \ldots, \phi(v_n))$. Then, since $v \in A$ we have that $wt(v) \geq D$. Furthermore, for every $v_i \neq 0$ it holds that $\phi(v_i) \neq 0$ (this is because ϕ is 1–1 and so its kernel contains 0 only). Now, for every i, $\phi(v_i) \in B$ and thus if $\phi(v_i) \neq 0$ it holds that $wt(\phi(v_i)) = d$ (because $d(B) = d$). We conclude that there are at least D coordinates of weight d in c and so $wt(c) \geq dD$. This holds for all nonzero codewords of C and so $d(C) \geq dD$, as required. ∎

Code concatenation (algorithm):

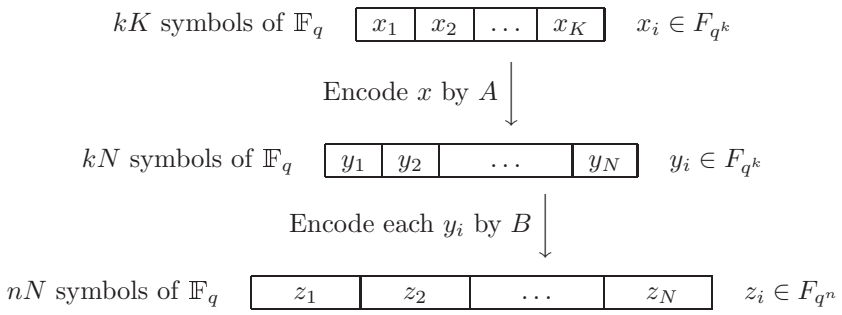
Input: a word $x \in F_q^{kK}$

Output: a codeword $c \in C$ in the concatenated code

Construction:

1. Denote $x = (x_1, \ldots, x_K)$ where each $x_i \in \mathbb{F}_{q^k}$.
2. Compute $A(x) = y = (y_1, \ldots, y_N)$ where each $y_i \in \mathbb{F}_{q^k}$ (this is valid because A is a code of length N and dimension K over alphabet $\mathbb{F}_{q^k}$).
3. For every $i \in \{1, \ldots, N\}$ compute $B(y_i)$, and output $C(x) = (B(y_1), \ldots, B(y_N))$ (this is valid because B is a code of dimension k over alphabet $\mathbb{F}_q$).

A visual representation:



A naive decoding procedure. Given a word $z \in F_q^{nN}$ carry out the following:

1. Denote $z = z_1, \ldots, z_N$ where for every i, $z_i \in \mathbb{F}_{q^n}$.
2. Correct the errors for every $z_i \in B$ and obtain $y_1, \ldots, y_N$ (decoding works as long as the number of errors in z_i is at most $\frac{d-1}{2}$)
3. Correct the errors for $y = (y_1, \ldots, y_N)$ according to A, and output the result x (decoding works as long as at most $\frac{D-1}{2}$ of the y_i symbols are incorrect).

When does this procedure succeed? As we have described, if there are more than $\frac{D-1}{2}$ symbols y_i that are incorrect then we may fail. This can occur if there are more than $\frac{D-1}{2}$ words z_i which have more than $\frac{d-1}{2}$ errors each. Thus, we can correct up to $\frac{D-1}{2} \cdot \frac{d-1}{2} < \frac{dD}{4}$ errors. This is unsatisfactory because in principle we should be able to fix up to $\frac{dD-1}{2}$ errors. Fortunately, this can be achieved and we will see how later on.

Example: Concatenating GRS with Reed-Muller: The idea is to concatenate two codes with good distances. We therefore take a GRS and Reed-Muller code. We take the external code to be a GRS code with parameters $[n, \frac{n}{2}, \frac{n}{2}+1]$ over $\mathbb{F}_q$ with $q = n = 2^{m+1}$ for some m. (Such a code exists because q is a prime power.) We then concatenate this code with the Reed-Muller code with parameters $[2^m, m+1, 2^{m-1}]$; equivalently, its parameters are $[\frac{n}{2}, \log n, \frac{n}{4}]$. Concatenation here is legal because the symbols of the GRS alphabet are vectors that can be encoded using Reed-Muller. By Theorem 7.3, we have that the concatenated code has parameters $\left[\frac{n^2}{2}, \frac{n}{2}\log n, \frac{n^2}{8}\right]$. Thus, $\delta = \frac{1}{4}$ and $R(C) = \frac{\log n}{n} \to 0$. This is therefore *not* an asymptotically good code.

7.3 The Forney Code

The problem with concatenating GRS with Reed-Muller is that the internal code (Reed-Muller) is *not* asymptotically good. Thus, the result of the concatenation is also not asymptotically good. We would therefore like to concatenate a GRS code with an asymptotically good internal binary code. The problem is that we don't know of such a code that is efficiently constructible, and we are therefore back to square one.

The observation by Forney is that the internal code does not actually need to be efficiently constructible. This is due to the fact that its codewords are symbols of the external code and so the internal code's length is logarithmic in the length of the entire code. This means that if the internal code can be constructed in time that is exponential in the length of the code, then this will still be polynomial in the length of the concatenated code.

As we have seen, the Gilbert-Varshamov bound states that for every $\delta \leq 1/2$ and R such that $R \leq 1-H(\delta)$ there exists a code with rate R and relative distance δ (for $n \to \infty$). Let $\delta > 0$ be any constant. We then define the following code:

External code: *A GRS code with parameters* $[n, \frac{n}{2}, \frac{n}{2}+1]$ *over* $\mathbb{F}_n$ *where* n *is a power of 2 (and so* $\log n$ *is a whole number)*

Internal code: *A Gilbert-Varshamov code with parameters* $\left[\frac{\log n}{1-H(\delta)}, \log n, \frac{\delta \log n}{1-H(\delta)}\right]$, *where* $\delta < \frac{1}{2}$

Before proceeding, note that by Theorem 4.8 such a internal code exists because $R = 1 - H(\delta)$ and the relative distance is δ. Furthermore, recall that a Gilbert-Varshamov code is constructed by iteratively finding columns that are linearly independent of all subsets of $d-2$ columns. The number of subsets is greatest when $\delta = \frac{1}{2}$, in which case there can be up to $\binom{\gamma \log n}{(\gamma \log n)/2}$ different subsets (actually less), where $\gamma = \frac{1}{1-H(\delta)} > 1$. Since $\sum_{i=1}^{m} \binom{m}{i} = 2^m$ we have that $\binom{\gamma \log n}{(\gamma \log n)/2} < n^\gamma$ which is polynomial because γ is a constant that depends on δ. Now, in order to find a new column, we can hold a list of all n possible vectors of length n and then for each subset of $d-1$ columns we remove the vectors that are in the span. This is carried out by just computing all linear combinations. Noting that there are 2^{d-1} such combinations (just take every possible subset and add them together), we have that this too is upper bounded by n^γ. Thus, the overall amount of time to find a new column is polynomial in n, as required.

Theorem 7.3 states that the concatenation of the above two codes yields a code with parameters:

$$\left[\frac{n \log n}{1-H(\delta)}, \frac{n \log n}{2}, \frac{\delta n \log n}{2(1-H(\delta))}\right]$$

Thus

$$R(C) = \frac{1-H(\delta)}{2} \qquad \text{and} \qquad \delta(C) = \frac{\delta}{2}$$

which are both constant! We have therefore demonstrated the first part of Theorem 7.2 (specifically, the existence of an efficiently-constructible asymptotically good code).[8] It thus remains to prove that the Forney code can be efficiently decoded. We do this in the next section.

[8] Note that the concatenated code is not optimal in that we lost a factor of 2 with respect to the Gilbert-Varshamov bound. Nevertheless, this is a relatively small price to pay given that we have obtained an efficiently-constructible code with a constant rate and relative distance.

7.4 Efficient Decoding of the Forney Code

We have already seen that a naive decoding procedure can only correct up to $\frac{dD}{4}$ errors, which is unsatisfactory. In this section, we will show how to improve this and *efficiently* correct up to $\frac{dD-1}{2}$ errors. This will therefore complete the proof of Theorem 7.2 (specifically, the part relating to the polynomial-time decoding procedure). Before proceeding, we prove the following useful lemma:

Lemma 7.4 *Let C be a GRS code with parameters $[N, N-D+1, D]$ over $\mathbb{F}_q$. Let $r \in (\mathbb{F}_q \cup \{?\})^N$ be a vector that is obtained from $c \in C$ by inserting up to e errors and S erasures (where the erasures are marked by "?"). Then there exists a polynomial-time procedure to find c from r as long as $2e + S < D$.*

Proof: Given the received word $r = (r_1, \ldots, r_N)$, we erase from the code all the coordinates i for which $r_i =?$. This is carried out by erasing the corresponding columns from the generator matrix G_{GRS}. Observe that the code that we obtain by erasing $S < D$ columns from G_{GRS} is a GRS code with parameters $[N - S, N - D + 1, D - S]$ (see code puncturing in Theorem 5.1 for the parameters; the fact that the code remains GRS is from Claim 6.3). We know how to efficiently decode this code and correct up to $e = \frac{D-S-1}{2}$ errors. If it indeed holds that $2e + S < D$ then $e < \frac{D-S}{2}$ and so we can correct all of the errors in the punctured code.

Using this procedure we can decode the punctured code. This does not yield the codeword for the original GRS code. However, we can use it to find the vector x such that $x \cdot G_{\text{GRS}}$ is the original GRS codeword, and this suffices. ∎

The idea behind the decoding procedure is as follows. Given a word $z_1, \ldots, z_N$ we will decode every z_i to y_i by exhaustive search (recall that this is carried out on the internal code of logarithmic length). If z_i is of distance $\frac{d}{2}$ or more from every codeword, then we decode z_i to an erasure "?". Otherwise, we decode it to its correct codeword. Unfortunately, this doesn't solve all of our problems because if enough errors were incurred by z_i then it may be within $\frac{d}{2}$ distance from an *incorrect* codeword y_i. We will therefore want to sometimes insert erasures even when we are closer than $\frac{d}{2}$. We will do this probabilistically at first.

7.4.1 A Probabilistic Decoding Algorithm

We denote $A(x) = y = y_1, \ldots, y_n$ and $C(x) = B(y_1), \ldots, B(y_n)$, where A is the external GRS code (of length n and distance D) and B is the internal Gilbert-Varshamov code (of dimension $\log n$ and distance d). The algorithm is as follows:

- **Input:** a vector $z = z_1, \ldots, z_n$ where each $z_i \in \mathbb{F}_{2^{\log n}}$
- **The algorithm:**
  1. Choose τ randomly between 0 and 1 (this involves choosing a uniformly distributed real number which is problematic; nevertheless, as we will see later something simpler suffices).
  2. For every $i = 1 \ldots n$ decode z_i to y_i by finding the closest codeword $B(y_i) \in B$ such that $d(z_i, B(y_i))$ is minimized.
  3. For every i, set $y_i = ?$ if $\frac{2e_i}{d} \geq \tau$, where $e_i = d(z_i, B(y_i))$.
  4. Run the GRS decoding algorithm given in Lemma 7.4 above on input $(y_1, \ldots, y_n)$ to find x.
  5. Output x.

We remark that the above probabilistic algorithm can be more simply rewritten so that no τ is chosen, but rather for every i we set $y_i = ?$ with probability $\max\{1, \frac{2e_i}{d}\}$. We also note that if $e_i \geq \frac{d}{2}$ then we always set $y_i =?$; this makes sense because in such a case y_i is far from all codewords. (Observe also that if $e_i = \frac{d}{4}$ then we set $y_i = ?$ with probability $\frac{2e_i}{d} = \frac{1}{2}$.)

We will now prove the correctness of the algorithm. Before doing so, we define two random variables:

- E: The number of words z_i that are decoded incorrectly (i.e., the number of words z_i that are decoded to $B(u_i)$ where $u_i \neq y_i$ and y_i is the correct word)
- S: The number of words z_i that are decoded to "?"

Lemma 7.5 *In the probabilistic decoding algorithm above, it holds that* $\mathsf{Exp}[2E + S] < D$

Proof: We use the following notation:

- z_i: the values received
- u_i: the word received after decoding z_i by Gilbert-Varshamov in the algorithm
- y_i: the "correct" word for the i'th coordinate in the GRS code (after GV decoding)
- e_i: the distance between z_i and the nearest word $d(z_i, B(u_i))$
- e'_i: the distance between z_i and the correct word; i.e., $e'_i = d(z_i, B(y_i))$

We assume that $\sum_{i=1}^{n} e'_i < \frac{Dd}{2}$ (otherwise there are too many errors and the algorithm does not need to work.

Define random variables E_i and S_i as follows:

$$S_i = \begin{cases} 1 & \text{the algorithm sets } u_i = ? \\ 0 & \text{otherwise} \end{cases}$$

$$E_i = \begin{cases} 1 & \text{the algorithm sets } u_i \notin \{y_i, ?\} \\ 0 & \text{the algorithm sets } u_i \in \{y_i, ?\} \end{cases}$$

Thus, $S = \sum_{i=1}^{n} S_i$ and $E = \sum_{i=1}^{n} E_i$. By the definition of the random variable S_i and the algorithm, $\mathsf{Exp}[S_i] \leq \frac{2e_i}{d}$ because $\mathsf{Exp}[S_i] = \Pr[S_i = 1] = \Pr[y_i = ?]$ and

$$\Pr[y_i = ?] = \Pr\left[\frac{2e_i}{d} \geq \tau\right] = \Pr\left[\tau \leq \frac{2e_i}{d}\right] = \frac{2e_i}{d}.$$

(It holds that $\mathsf{Exp}[S_i] < \frac{2e_i}{d}$ when $\frac{2e_i}{d} > 1$; otherwise we have equality.) We will now show that for every i, $\mathsf{Exp}[E_i] \leq \frac{e'_i - e_i}{d}$. There are two possible outcomes for the decoding:

1. *Case 1 – $u_i = y_i$:* In this case $\Pr[E_i = 1] = 0$ and so $\mathsf{Exp}[E_i] = 0$. Since $e'_i \geq e_i$ always (the number of real errors is always at least the distance between the received word and the decoded word) we have that $\frac{e'_i - e_i}{d} \geq \mathsf{Exp}[E_i]$ as required.
2. *Case 2 – $u_i \neq y_i$:* $e'_i = d(z_i, B(y_i)) \geq d(B(y_i), B(u_i)) - d(B(u_i), z_i) \geq d - e_i$, where the last inequality holds because $B(u_i), B(y_i) \in B$ and so $d(B(u_i), B(y_i)) \geq d$. Now,

$$\Pr[E_i = 1] = \Pr[u_i \neq ?] = 1 - \frac{2e_i}{d} = \frac{d - 2e_i}{d} = \frac{(d - e_i) - e_i}{d} \leq \frac{e'_i - e_i}{d}$$

   where the first equality is because we already have that $u_i \neq y_i$ and so $E_i = 1$ if and only if $u_i \neq ?$. (Note that in case $\frac{2e_i}{d} \geq 1$ we have that $\Pr[E_i = 1] = 0$ and so the inequality still holds.)

We conclude that in both cases, $\mathsf{Exp}[E_i] \leq \frac{e'_i - e_i}{d}$. (Formally, the above analysis should be written by conditioning E_i on the above two cases.) Furthermore, we have already shown that $\mathsf{Exp}[S_i] \leq \frac{2e_i}{d}$. We are now ready to compute the expected value of $2E + S$ (using the linearity of expectations). For every i, we have:

$$\mathsf{Exp}[2E_i + S_i] = 2\mathsf{Exp}[E_i] + \mathsf{Exp}[S_i] \leq \frac{2(e'_i - e_i)}{d} + \frac{2e_i}{d} = \frac{2e'_i}{d}$$

and so

$$\mathsf{Exp}[2E+S] = \mathsf{Exp}\left[\sum_{i=1}^{n}(2E_i+S_i)\right] = \sum_{i=1}^{n}\mathsf{Exp}[2E_i+S_i] \leq \sum_{i=1}^{n}\frac{2e_i}{d} = \frac{2}{d}\sum_{i=1}^{n}e'_i < \frac{2}{d}\cdot\frac{Dd}{2} = D$$

completing the proof of the lemma. ∎

Intuitively, this suffices for bounding the error of the algorithm, because by Lemma 7.4, the phase of GRS decoding works as long as $2e+S<D$ and we have just seen that the *expected* value of $2e+S$ from the first phase of GV decoding fulfills this requirement. Nevertheless, instead of formally proving this, we show how the algorithm can be made deterministic (and thus errorless).

7.4.2 A Deterministic Decoding Algorithn

Lemma 7.6 *There exists a threshold τ for which $\Pr[2E+S<D]=1$. Furthermore, the threshold τ is one of the values in the set $\{0,1\}\cup\left\{\frac{2e_i}{d}\right\}_{i=1}^{n}$.*

Proof: We have already proven that $\mathsf{Exp}[2E+S]<D$. Thus, by the probabilistic method there exists a value τ for which $\Pr[2E+S<D]=1$. In order to see this, notice that the probability in $\mathsf{Exp}[2E+S]$ is over the choice of τ. Thus, once τ is set, the question of whether $2E+S$ is fully determined. This implies that for some values of τ it must hold that $2E+S<D$ (otherwise if for all τ it holds that $2E+S\geq D$ then it must be that $\mathsf{Exp}[2E+S]\geq D$). In addition, the question of whether or not $y_i=?$ is determined by the threshold τ. Since the algorithm compares the values $\frac{2e_1}{d},\frac{2e_2}{d},\ldots,\frac{2e_n}{d}$ to the threshold, the only values that can affect the outcome are these, and also $0,1$ (0 when τ is smaller than all the $\frac{2e_i}{d}$ values and 1 when τ is larger than them all). ∎

The deterministic algorithm: Run Forney's probabilistic decoding algorithm with all $n+2$ possible thresholds. (Note that we know the values $e_1,\ldots,e_n$ because these are the distances of the decoded values u_i from z_i.) Stop and output any codeword that is found that is within distance less than $\frac{Dd}{2}$ from the received word.

We have therefore proven the following theorem:

Theorem 7.7 *There exists a deterministic polynomial-time algorithm that corrects $\left\lfloor\frac{Dd-1}{2}\right\rfloor$ errors in the Forney code (GRS concatenated with GV).*

8 Local Decodability

In some cases, especially when a codeword is very long, we are interested in decoding only part of the word. In such a case, we would like a decoding algorithm which is much quicker than decoding the entire word. A code that has this property (that it is possible to decode the ith symbol in time that is much quicker than the entire word) is called *locally decodable*.

We consider algorithms that have random access to the word (and so can read specific symbols of the word without linearly traversing it). We formalize this by giving the algorithm an oracle which contains the received string. Specifically, when y is the oracle, then upon query i to the oracle, the algorithm receives back y_i, the ith letter of y.

Definition 8.1 *A code C with parameters $[n,k]$ over $\mathbb{F}_q$ is called* ℓ-locally decodable with error δ *if there exists a probabilistic algorithm A that upon input j and oracle $y \in \mathbb{F}_q^n$ asks at most ℓ queries to y, runs in time* $\mathrm{poly}(\ell, \log n)$ *and has the property that*

$$\Pr[A^y(j) \neq c_j] < \frac{1}{2}$$

assuming there exists a codeword $c \in C$ such that $d(y,c) < \delta n$.

Remark 8.2 *Note that if for every $c \in C$, $d(y,c) \geq \delta n$ then the algorithm is allowed to return any value. However, if the algorithm can detect such a case and outputs a special fail symbol when this happens, then the code is called* locally testable.

Proposition 8.3 *Let C be an $[n,k]$ code over $\mathbb{F}_q$. If $C^\perp$ has distance $d \geq \ell + 1$ then C is not $\ell - 1$-locally decodable for any $\delta > 0$.*

Proof: By the assumption $d(C^\perp) \geq \ell + 1$. By Theorem 2.15 this implies that every subset of ℓ columns in the parity-check matrix of $C^\perp$ are linearly independent. This in turn implies that every subset of ℓ columns of the generator of C are linearly independent.

Denote the columns of G (the generator of C) by $G_1, \ldots, G_n$, and denote $c = c_1, \ldots, c_n$. Then there exists a vector $x \in \mathbb{F}_q^k$ such that for every $j \in \{1, \ldots, n\}$ it holds that $c_j = x \cdot G_j$. Now, since every subset of ℓ columns are linearly independent, it holds that c_j is independent of $\{c_i\}_{i \in I}$ for every subset $I \subseteq \{1, \ldots, n\} \setminus \{j\}$ of size at most $\ell - 1$. This implies that seeing any subset of $\ell - 1$ bits provides no information on c_j, and thus viewing $\ell - 1$ bits does not suffice for decoding.[9] ∎

Exercise: Show that there exists a code of length n that is 1-locally decodable for $\delta < 1/2$.

8.1 Locally decoding Hadamard codes

Recall that the values $z_1, \ldots, z_{2^k-1}$ denote all the nonzero strings in $\mathbb{F}_{2^k}$. Furthermore, the Walsh-Hadamard code, defined by:

$$C(x) = (\langle x, z_1 \rangle, \ldots, \langle x, z_{2^k-1} \rangle)$$

is a linear code with parameters $[2^k - 1, k, 2^{k-1}] = [n - 1, \log n, \frac{n}{2}]$. We prove the following:

Theorem 8.4 *The Walsh-Hadamard code is 2-locally decodable with $\delta \leq \frac{1}{4}$.*

[9] If you are not convinced by the "linear independence" argument, then we claim this more slowly. Specifically, we claim that for every subset $\{c_i\}_{i \in I}$ ($|I| < \ell$) and every $c_j \in \mathbb{F}_q^n$ (for $j \notin I$) there exists a vector $x \in \mathbb{F}_q^k$ such that for every $i \in I$ it holds that $x \cdot G_i = c_i$ and furthermore $x \cdot G_j = c_j$. This follows from the fact that we can write a system of linear equations with k variables $x_1, \ldots, x_k$ and ℓ equations. Now, since each column of G is of length k it must be that $\ell \leq k$ (it is not possible to have more than k vectors of length k that are linearly independent). Thus, we have k variables and at most k linearly independent equations, and so we have a solution. Since this holds for every $c_j \in \mathbb{F}_q^n$ it follows that the values $\{c_i\}_{i \in I}$ do not give us any information about c_j.

Proof: We describe an algorithm A which receives an oracle $y \in \mathbb{F}_2^n$ and an index j. A chooses a random $r \in \{1, \ldots, 2^k - 1\}$ and reads the values y_r and y_m where m is the index for which $z_m = z_r + z_j \mod 2$. Finally, A outputs $c_j = y_m - y_r \mod 2$.

First we show that if y_r and y_m are both correct (without errors), then the output is correct. This follows simply because

$$\begin{aligned} y_m - y_r &= \langle x, z_m \rangle - \langle x, z_r \rangle \\ &= \sum_{i=1}^{k} x_i \cdot z_m^i - \sum_{i=1}^{k} x_i \cdot z_r^i \\ &= \sum_{i=1}^{k} x_i \cdot (z_r^i + z_j^i) - \sum_{i=1}^{k} x_i \cdot z_r^i \\ &= \sum_{i=1}^{k} x_i \cdot z_j^j = \langle y, z_j \rangle = c_j \end{aligned}$$

Now, assume that there are errors. Since r is chosen randomly, so is m (although they are not independent). Therefore

$$\Pr[y_r \neq \langle x, z_r \rangle] < \delta$$

because $d(y, c) < \delta n$ and so a random bit is incorrect with probability less than δ. Likewise,

$$\Pr[y_m \neq \langle x, z_m \rangle] < \delta.$$

Therefore,

$$\Pr[y_m \neq \langle x, z_m \rangle \vee y_r \neq \langle x, z_r \rangle] < 2\delta = \frac{1}{2}$$

as required. ■

Hadamard and Hamming. Recall that the dual code of Walsh-Hadamard is the Hamming code which has distance 3. Thus, by Proposition 8.3 the Walsh-Hadamard code is not locally-decodable for $\ell = 1$. This shows also that Proposition 8.3 is tight because there does exist a code C that is ℓ-locally decodable and whose dual code has distance $d = \ell + 1$.

9 List Decoding

OMITTED THIS YEAR.

10 Hard Problems in Coding Theory

10.1 The Nearest Codeword Problem and NP-Completeness

In this section, we study the nearest codeword problem and show that it is NP-complete. We then proceed to show that it is even hard to approximate. Intuitively, the nearest codeword problem is the natural task of finding the nearest codeword to some given word, in an arbitrary linear code. Formally:

Definition 10.1 *Let C be a linear $[n,k]$-code with generator matrix $G \in \mathbb{F}_q^{k \times n}$. The* nearest codeword problem (NCP) *is defined as follows: given G and a vector $y \in \mathbb{F}_q^n$, find $x \in \mathbb{F}_q^k$ such that $d(xG, y)$ is minimal over all $x \in \mathbb{F}_q^k$. The* decisional NCP *problem is defined by the set*

$$\{(G, y, \ell) \mid \exists x \in \mathbb{F}_q^k \text{ s.t. } d(xG, y) \leq \ell\}$$

where G is a $k \times n$ generator matrix and $y \in \mathbb{F}_q^n$.

We now prove that NCP is NP-complete; it suffices to consider the binary case that $q = 2$ only. This is proven by an elegant reduction to MAX-CUT. Recall that a cut in a graph (V, E) is defined by a subset of vertices $S \subseteq V$; the cut itself is the set of edges going between S and $V \setminus S$. The *size* of the cut therefore equals

$$|\{(u, v) \in E \mid u \in S, v \in V \setminus S\}|.$$

The MAX-CUT problem is therefore to find a subset of vertices $S \subseteq V$ for which the size of the cut defined by S is the largest. The decision version of MAX-CUT is simply the set of all triples (V, E, t) for which there exists a cut of size at least t in the graph defined by the vertices V and edges E.

Theorem 10.2 *The nearest codeword problem is NP-complete.*

Proof: It is clear that $NCP \in \mathcal{NP}$; just guess a codeword x and then verify that $d(xG, y) \leq \ell$. We now show that NCP is NP-hard by reducing it to MAX-CUT. The reduction is defined as follows:

1. Let (V, E, t) be the input to MAX-CUT. Define the incidence matrix of the graph (V, E) with $|V| = k$ and $|E| = n$ to be the matrix $M \in \mathbb{F}_2^{k \times n}$ where $M_{i,j} = 1$ if and only if one of the endpoints of e_j is the vertex v_i.

2. Define the generator matrix $G = M$, set $y = 1^n$ and set $\ell = n - t$.

3. Output (G, y, t).

Before proceeding we remark that it is important that G be a valid generator matrix, meaning that its rows are linearly independent. It is easy to see that G above is actually *never* a valid generator matrix. We will deal with this below.

Claim 10.3 *$(V, E, t) \in$ MAX-CUT if and only if $(G, y, \ell) \in NCP$.*

Proof: Assume that $(V, E, t) \in$ MAX-CUT and let S be a subset of vertices defining a cut of size at least t in (V, E). Define a word $x \in \{0,1\}^k$ such that $x_i = 1$ if and only if $v_i \in S$. Observe that for every $1 \leq j \leq n$, e_j is in the cut defined by S if and only if $(xG)_j = 1$. In order to see this, note first that every column of G has at most two 1s (if there are self edges then there may be a column with only one 1). Furthermore, the jth column corresponds to the edge e_j and thus if $e_j = (v_a, v_b)$ then the ath and bth bits of the column equal 1 and all other bits equal 0. Now, if $e_j = (v_a, v_b)$ is in the cut then $v_a \in S$ or $v_b \in S$ but not both. Thus, either $x_a = 0$ and $x_b = 1$, or $x_a = 1$ and $x_b = 0$. In both cases, when multiplying x by the jth column, we have $0 \cdot 1 + 1 \cdot 1 = 1$ and thus $(xG)_j = 1$. For the other direction, if $(xG)_j = 1$ then it cannot be that $x_a = x_b = 0$ or $x_a = x_b = 1$ and thus either $v_a \in S$ or $v_b \in S$, implying that e_j is in the cut.

Now, since $(V, E, t) \in$ MAX-CUT we have that the above implies that $wt(xG) \geq t$ and so $d(xG, y) \leq n - t = \ell$ (recall that $y = 1^n$). Thus, $(G, y, \ell) \in NCP$, as required. Furthermore, if $(G, y, \ell) \in NCP$ then by the same reasoning (the "other direction" above) there exists a cut of size t in (V, E), as required. ■

In order to complete the proof, we need to show that the above also holds for G with rows that are linearly independent. However, this can be easily achieved as follows. Within the reduction, instead of taking the matrix G which is defined by the incidence matrix M, take G' which is a subset of linearly independent rows of M (and find G' via Gaussian elimination). This defines a code of length n and dimension $k' < k$. However, the important point to notice is that if there exists a word $x \in \mathbb{F}_2^k$ such that $d(xG, y) \leq \ell$ then there exists a word $x' \in \mathbb{F}_2^{k'}$ such that $d(x'G', y) \leq \ell$. This holds because $d(xG, y) \leq \ell$ if and only if $wt(xG) \geq n - \ell$. Since G and G' span the same vector space, it follows that the existence of a word y of weight at least ℓ in the span of the rows of G implies the existence of a word of the same weight in the span of G'. Thus, we obtain that $(V, E, t) \in$ MAX-CUT if and only if $(G', y, \ell) \in NCP$, where $y = 1^n$. This completes the proof of NP-completeness. ■

10.2 Hardness of Approximation

Given that NCP is NP-hard, the next natural question is to ask whether or not there exists an approximation algorithm. In this section, we show that the nearest codeword problem is even *hard to approximate*. From now on, we will not consider the decision version of the problem. Rather, we will consider the *minimization problem* which is: given a generator matrix G and a word $y \in \mathbb{F}_q^n$, find $x \in \mathbb{F}_q^k$ such that $d(xG, y)$ is minimal over all $x \in \mathbb{F}_q^k$. We provide the following informal definition of an approximation algorithm:

1. **α-approximation for a maximization problem:** Let $\alpha < 1$. A polynomial-time algorithm A is an α-approximation if for every input x with optimal solution of value C^*, algorithm A returns a solution of value C where $\alpha \cdot C^* \leq C \leq C^*$.

2. **β-approximation for a minimization problem:** Let $\beta > 1$. A polynomial-time algorithm A is an β-approximation if for every input x with optimal solution of value C^*, algorithm A returns a solution of value C where $C^* \leq C \leq \beta \cdot C^*$.

Observe that the aim of a maximization problem is to find the solution with the highest value. Thus, an α-approximation is guaranteed to find one with value *at least* α times the optimal (since $\alpha < 1$ this is less than optimal, as expected). Likewise, when considering minimization problems, the output of the algorithm is guaranteed to be no more than β times the optimal, where $\beta > 1$.

Approximating MAX-CUT. We begin with some known results regarding MAX-CUT.

Proposition 10.4 *For every graph (V, E) there exists a cut of size $\frac{|E|}{2}$.*

Proof: Let $|V| = k$ and $|E| = n$. Choose a random set S of vertices by placing every vertex in the set S with probability exactly $1/2$. For every $1 \leq i \leq n$ define the random variable X_i such that $X_i = 1$ if and only if e_i is inside the cut defined by S. Now, for every i we have that $\Pr[X_i = 1] = \frac{1}{2}$; this holds because there are 4 possibilities, 2 of which yield $X_i = 1$ (namely, either both endpoints of the edge are in S, both are not, or exactly one of the endpoints is). Thus, for every i, $E[X_i] = \frac{1}{2}$. By the linearity of expectations, we have:

$$E[\text{size of cut}] = E\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} E[X_i] = \frac{n}{2} = \frac{|E|}{2}$$

Since the expected size of the cut of a random subset is $|E|/2$, by the probabilistic method we have that there exists a cut of at least this size. ■

A very important theorem, proven using PCP, states that MAX-CUT is NP-hard to approximate for some $\alpha < 1$.

Theorem 10.5 *If $\mathcal{P} \neq \mathcal{NP}$ then MAX-CUT is hard to α-approximate for $\alpha = \frac{16}{17} + \epsilon$ and any $\epsilon > 0$.*

We use the above fact to prove hardness of approximation for the nearest codeword problem as well.

Approximating the nearest codeword problem. As we have seen, there is a close connection between NCP and MAX-CUT. We will use this to derive hardness of approximation for NCP. We stress that a regular NP-reduction between problems does not necessarily imply that hardness of approximation follows as well. However, in this case, as we will see, it does follow.

Theorem 10.6 *If $\mathcal{P} \neq \mathcal{NP}$ then there exists a constant $\beta > 1$ for which the NCP problem is hard to β-approximate.*

Proof: In the proof of Theorem 10.2 we saw that if there exists a polynomial-time algorithm that finds a word within distance $\ell = n - t$ from the word $y = 1^n$ then there exists a polynomial-time algorithm that finds a cut of at least size $t = n - \ell$ in a graph.

Assume now that there exists a β-approximation algorithm A for NCP. Let (V, E) be the input graph and denote by t^* the size of the maximum cut in the input graph. Letting G be the incidence matrix in the reduction of Theorem 10.2 we have that there exists a word $x^* \in \mathbb{F}_2^k$ such that $d(x^*G, 1^n) \leq n - t^* = \ell^*$.

By the assumption that A is a β-approximation algorithm for NCP, we have that $A(G, 1^n)$ returns a word x such that $d(xG, 1^n) = \ell$ and

$$\ell^* \leq \ell \leq \beta \cdot \ell^*.$$

Equivalently, A returns x such that $d(xG, 1^n) = \ell = n - t$ and

$$n - t^* \leq n - t \leq \beta \cdot (n - t^*).$$

This implies that

$$\beta \cdot (t^* - n) \leq t - n \leq t^* - n$$

and so

$$\beta t^* - \beta n + n \leq t \leq t^*.$$

Now,

$$\beta t^* - \beta n + n = \beta t^* - n(\beta - 1) \geq \beta t^* - 2t^*(\beta - 1)$$

where the inequality is due to the fact that there is always a cut of size $\frac{n}{2}$ and so $t^* \geq \frac{n}{2}$, or equivalently, $n \leq 2t^*$.

We have

$$\beta t^* - \beta n + n \geq \beta t^* - 2t^*(\beta - 1) = \beta t^* - 2t^*\beta + 2t^* = 2t^* - \beta t^* = t^*(2 - \beta)$$

and so

$$t^*(2 - \beta) \leq t \leq t^*.$$

This implies that the reduction that we have seen gives us an algorithm that finds a cut of size t such that $t^*(2 - \beta) \leq t \leq t^*$. Setting $\alpha = 2 - \beta$ we have that if there exists a β-approximation algorithm for the NCP problem then there exists an α-approximation for MAX-CUT. Since no such approximation exists for $\alpha = \frac{16}{17} + \epsilon$ for any $\epsilon > 0$, we have that no $\frac{18}{17} - \epsilon$-approximation algorithm exists for the NCP problem, for any $\epsilon > 0$. ■

The above provides an interesting hardness of approximation result. However, we can actually do much better. Specifically, we can show that there does not exist a β-approximation algorithm for the nearest codeword problem *for any constant* $\beta > 1$.

Theorem 10.7 *If $\mathcal{P} \neq \mathcal{NP}$ then there does not exists a β-approximation algorithm for the nearest codeword problem, for any constant $\beta > 1$.*

Proof: We have actually shown that the nearest codeword problem is NP-hard to β-approximate even in the specific case that we wish to find the closest codeword to 1^n (otherwise known as the heaviest codeword). We will therefore focus on this special case only here (for sake of simplicity we still call the problem NCP although it is more accurate to call it the *heaviest codeword problem*). In any case, we stress that if there exists a β-approximation to NCP then there exists a β-approximation to the heaviest codeword problem. Thus, it suffices to show hardness of approximation for the latter problem.

We will show that if there exists a β-approximation algorithm for NCP then there exists a $\sqrt{\beta}$-approximation algorithm for NCP. This suffices to prove the theorem as follows. From Theorem 10.6 we know that there exists a constant $\beta > 1$ for which NCP is hard to approximate; indeed we know that this holds for *any* $\beta < \frac{18}{17}$. Now, if there exists a $\gamma \geq \frac{18}{17}$ such that there exists a γ-approximation algorithm for NCP, then by repeatedly applying the above k times, we obtain that there exists a $\gamma^{2^{-k}}$-approximation for NCP. By taking k to be a constant such that $\gamma^{2^{-k}} < \frac{18}{17}$ we obtain a contradiction.

Let G be the generator matrix for a binary code C of length n. Define a code $C^{(2)}$ such that every word in $C^{(2)}$ is represented by an n-by-n matrix where every column of the matrix is a code word of C or its complement. In order to determine whether a column is a codeword or its complement, we add a column headers to the matrix which are actually a codeword of C; if the header of a column equals 0 then the column is a codeword, and if the header equals 1 then the column is the complement of a codeword. (Overall, a single word in $C^{(2)}$ is comprised of n codewords of C. However, note that the codewords of C composing the word in $C^{(2)}$ need not be distinct.)

For example, for $C = \{000, 011, 110, 101\}$, the following is a codeword in $C^{(2)}$:

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

where the column header is the codeword 110.

Claim 10.8 *If C is a linear code of length n, then $C^{(2)}$ is a linear code of length n^2.*

Proof: We show that the sum of any two codewords (matrices) is a code word. This is due to the fact that if the column header bits of a given column are both 0 or both 1 then we added two original codewords in C or two complements; in both cases the resulting header is 0 and the column is an original codeword. Likewise, if the column headers are different then we add a codeword and its complement and the result is a codeword complement, which is consistent with the header. (Observe that when we add two matrices we also "virtually add" the column headers as well.) This shows that closure holds; all other properties of vector spaces are easy to show. ■

Let $G^{(2)}$ be a generator matrix of $C^{(2)}$. Note that we can find this matrix efficiently by constructing $k \cdot n$ words where each word is comprised of a single row of G in one of the columns, and the rest of the columns all being zero. In addition, we add k words where each word is made up of all zero and all one columns, where the column is set to 0 or 1 based on a column header which is a row of G. It is easy to see that these words span $C^{(2)}$. (It appears that the words may also be linearly independent. However, this is not important because we can diagonalize efficiently. In any case, it makes no difference to the proof.)

Claim 10.9 *There exists a word of weight $n - \ell$ in C if and only if there exists a word of weight $n^2 - \ell^2$ in $C^{(2)}$.*

Proof: We first show that if there exists a word c of weight $n - \ell$ in C then there exists a word c^2 of weight $n^2 - \ell^2$ in $C^{(2)}$. We generate this word c^2 as follows:

1. Take the column header of the matrix c^2 to be c.
2. for every column i for which $c_i = 1$ set the column to be all ones

3. for every column i for which $c_i = 0$ set the column to be c

There are $n - \ell$ columns with n ones and ℓ columns with weight $n - \ell$. Thus, the total weight of the word c^2 is $(n - \ell) \cdot n + (n - \ell) \cdot \ell = n^2 - n\ell + \ell n - \ell^2 = n^2 - \ell^2$, as desired.

For the other direction, the word c^2 constructed from c is the heaviest possible codeword (where c is the heaviest word of C). Thus, if the heaviest word of C is of weight $w < n - \ell$ then there does not exist a codeword of weight $n^2 - \ell^2$ in $C^{(2)}$. ∎

We are now ready to prove that any algorithm that constitutes a β-approximation of NCP can be converted into an algorithm that achieves a $\sqrt{\beta}$-approximation. Before doing so, we assume without loss of generality, than any algorithm for NCP returns a codeword that is of the form c^2 above. This is without loss of generality because we can take the heaviest codeword in the columns of c^2 and in the label and can reconstruct c^2 from it. (Note that although the label in the column is implicit, it can easily be checked by multiplying the column and its complement by H and seeing which results in 0. This holds as long as the code is such that no word and its complement are in the code. Otherwise, the label is not defined by the column. However, if this happens then the code contains the all-one word, in which case this is the heaviest word and we are done.) The resulting word is at least as heavy as the one output by the algorithm.

Let A be a β-approximation algorithm. We construct A' as follows:

1. Given a generator matrix G of a linear code C, construct the generator matrix $G^{(2)}$ of $C^{(2)}$, as above.
2. Run A on $G^{(2)}$ and receive back a word c^2 of distance at most $\beta\ell^{*2}$ from 1^{n^2}, where the heaviest codeword of $C^{(2)}$ is of weight $n^2 - \ell^{*2}$.
3. Derive the word c used to construct c^2 (recall that we assume that A returns a word of the form above), and output c.

It remains to calculate the distance of c from 1^n. Let t denote the number of zeroes in c^2 (we have that $t \leq \beta\ell^{*2}$). For every zero in the column label of c^2, there are t zeroes in that column. This implies that there are $\sqrt{t}$ zeroes in the column label; equivalently that there are $\sqrt{t}$ zeroes in the codeword $c \in C$. Since $t \leq \beta\ell^{*2}$ we have that c has at most $\sqrt{\beta}\ell^*$ zeroes and so $d(c, 1^n) \leq \sqrt{\beta}\ell^*$, where ℓ^* is the closest possible. We conclude that A' outputs a word that is at most a factor of $\sqrt{\beta}$ away from the optimal. In other words, A' is a $\sqrt{\beta}$-approximation algorithm. This concludes the proof. ∎

We remark that the size of the instance that A' needs to deal with is n^2. Thus, if we repeatedly apply this k times, we have that the algorithm needs to work in time n^{2k}. If k is not constant, the resulting algorithm will not be polynomial time. Thus, the above only works to show hardness of approximation for any *constant* β.

11 CIRC: Error Correction for the CD-ROM

CIRC stands for *cross interleaved Reed-Solomon code*; it was developed in 1979 by Phillips and Sony (the CD-ROM standard includes the error correcting code). The code provides a very high level of error correction and is the reason that dirty and scratched CDs typically keep working. In this section we will present the code (on a relatively high level) and will informally discuss how it works. We stress that the aim here is *not* to provide a rigorous analysis of CIRC, but to demonstrate that the knowledge obtained in this course is enough to understand real codes that are used.

Background. Every disk contains a spiral track of length approximately 5 km. The track contains pits and lands (for representing bits). The width of the track is 0.6μm and the depth of a pit is 0.12μm. The laser is shone onto the track and is reflected with a different strength if it is focused on a land or a pit, enabling the bit to be read. Errors are very common on CD-ROMs because of dirt, scratches, fingerprints and more; these can turn pits into lands (e.g., dirt) and lands into pits (e.g., scratches). The errors are usually *burst errors*, meaning that they come in a continuous region.

CIRC Encoding

Every audio sample is 16 bits long; in order to achieve stereo, two samples are taken for each time point. There are 44,100 samples per second and every sample pair is 4 bytes long. Therefore, every second of audio is made up of 176,400 bytes. Each sample in a pair is a vector in $\mathbb{F}_2^{16}$; each such vector is divided into two, and we view each half as an element of $\mathbb{F}_{2^8}$ (note that $\mathbb{F}_{2^8}$ is not the same as $\mathbb{F}_2^8$). Two Reed-Solomon codes, denoted C_1 and C_2 are used in an interleaved fashion (this results in distributing large burst errors into small local errors).

Step 1: encoding with C_1

We define a frame of 6 samples for a total of 24 bytes. A frame is a series $L_1R_1 \cdots L_6R_6$ were every L_i and R_i are two bytes long. Before the actual coding the following permutation (interleaving) is applied:

1. Two different frames are interleaved, where the interleaved frames are not adjacent but have one other frame in between them. Denoting the frames by $L_1R_1 \cdots L_6R_6$ and $\tilde{L}_1\tilde{R}_1 \cdots \tilde{L}_6\tilde{R}_6$, we take the even samples from one of the frames with the odd samples from the other frame.

2. We then interleave the samples so that the result is:

$$L_1L_3L_5R_1R_3R_5\tilde{L}_2\tilde{L}_4\tilde{L}_6\tilde{R}_2\tilde{R}_4\tilde{R}_6$$
$$\tilde{L}_1\tilde{L}_3\tilde{L}_5\tilde{R}_1\tilde{R}_3\tilde{R}_5L_2L_4L_6R_2R_4R_6$$

   (This permutation places the even samples as far away as possible from the odd samples in the same frame.)

Next, the Reed-Solomon code $[28, 24, 5]$ over $\mathbb{F}_{256}$ is applied to the permuted frames; denote this code by C_1. Note that every frame is made up of 24 bytes as required by this code (recall also that each byte is viewed as an element of $\mathbb{F}_{2^8} = \mathbb{F}_{256}$). The encoding results in 4 bytes of redundancy; with standard encoding we obtain the original word with separate redundance. We denote the redundancy by P_1, P_2 (each being of length 2 bytes). Next, we once again reorder the frame so that we obtain:

$$L_1L_3L_5R_1R_3R_5P_1P_2\tilde{L}_2\tilde{L}_4\tilde{L}_6\tilde{R}_2\tilde{R}_4\tilde{R}_6$$

once again moving the odd and even frames as far apart as possible from each other.

We now take the above codeword and once again interleave. Specifically, every frame is a codeword of length 28 bytes in C_1. Denote the codewords obtained by applying C_1 to the input by $c_1, c_2 \ldots$ and denote

the jth byte of the ith word by $c_{i,j}$. Then, we take 28 codewords $c_1, \ldots, c_{28}$ and arrange them as follows:

$$\begin{array}{cccccccccc} c_{1,1} & c_{2,1} & c_{3,1} & c_{4,1} & c_{5,1} & c_{6,1} & c_{7,1} & c_{8,1} & c_{9,1} & c_{10,1} & \cdots \\ 0 & 0 & 0 & 0 & c_{1,2} & c_{2,2} & c_{3,2} & c_{4,2} & c_{5,2} & c_{6,2} & \cdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & c_{1,3} & c_{2,3} & \cdots \end{array}$$

We continue this for 28 rows so that all 28 bytes of every codeword are included. Note that the first row contains the first byte of each codeword, the second row contains the second byte of each codeword and so on. Since there are 28 bytes in each codeword, there are 28 rows. We pad the ends of the rows with zeroes as needed (in the last row, no padding is needed at the end). Observe that the matrix has 28 rows and $27 \cdot 4 + 28 = 136$ columns.

Step 2: encoding with C_2

Every column of the matrix is a word in $\mathbb{F}_{256}^{28}$. We therefore encode each *column* using the Reed-Solomon $[32, 28, 5]$ code over $\mathbb{F}_{256}$; every codeword here is 32 bytes long. Once again, the result is interleaved the result so that the even bytes are written together with the odd bytes of the consecutive word. The words are then written in one long stream, divided into 32 byte segments (after each segment an additional byte of general data is written, so that we actually have 33 bytes in each segment). This completes the encoding process.

CIRC Decoding

Step 1: decoding by C_2

The 33th byte containing general data is remove and the segments are rearranged so that each codeword is stored separately. Then, decoding is carried out according to C_2 but only *one error* is corrected (note that C_2 has distance 5 and so 2 errors can be corrected if desired). The reason for working in this way is so that the decoding provides the following:

1. Error correction of 1 error
2. Error detection of 2 or 3 errors

(Since $d(C_2) = 5$, a word of distance 3 from some codeword is of distance at least 2 from every other codeword.)

What is the probability that C_2 does not detect 4 errors when it is used to correct a single error? This event occurs if 4 errors in a codeword result in a word that is within radius 1 of some other codeword. Assuming that all vectors are equally probable, we have that the probability that this occurs equals the number of words in spheres of radius 1 around codewords divided by the overall number of words. There are 256^{28} codewords, each sphere of radius 1 contains $1 + 32(256 - 1) = 8161$ words, and there are 256^{32} vectors overall. Thus, the probability that the above occurs is

$$\frac{256^{28} \cdot 8161}{256^{32}} = \frac{8161}{256^4} \approx 1.9 \times 10^{-6}$$

In contrast, if we were to correct 2 errors using C_2, then the probability that we would fail due to 3 errors being introduced into a codeword (failure occurring because we mistakenly correct the word to an incorrect codeword of distance 2 from the received word) is

$$\frac{256^{28} \cdot \left(1 + 32 \cdot 255 + \binom{32}{2} \cdot 255^2\right)}{256^{32}} \approx 7.5 \times 10^{-3}$$

Since we still have another stage of decoding (by C_1), it is better at this stage to correct less errors, and detect more errors. (The implicit assumption in the above analysis is that a *serious error* results in a random-looking vector.)

Step 2: decoding by C_1

If the output of the frame from the previous step is a codeword (this happens if there were no errors or 1 error that was corrected), then we take the result of the frame as is for this stage. In contrast, if there were 2 or 3 errors, then the *entire word* of length 28 bytes is transformed into a special *erasure symbol* $\perp$ (like what we saw in Forney's decoding algorithm). The words of length 28 bytes are columns in the matrix that was constructed above in the encoding phase, and so we now reconstruct the matrix based on the words from the previous stage (all words are either codewords or erasures). We stress that not all codewords in C_2 yield valid codewords in C_1 and so error correction (and not just erasure correction) is still needed. So, at this stage the 28 byte frames for C_1 are taken and C_1 decoding is used as follows:

1. If there are $d - 1 = 4$ erasures, then these are corrected
2. If there are up to 2 erasures, then 2 erasures and 1 error is corrected (see Lemma 7.4 that states that this is possible if $2e + S < D$)

Since there is a shift of 4 bytes when building the matrix, and we can correct up to 4 erasures, we have that we can correct a very long burst error.

Step 3 of the decoding

If there remain any errors after the previous stage, then *error concealment* is used. Since consecutive frames are separated from each other, there is a good chance that the neighboring frames of one that could not be corrected are OK. In this case, linear interpolation is used to approximate the frame that could not be corrected. This usually works well. Other heuristics (that are specific to music) are used when not.