Chapter 1

6.897 Algorithmic Introduction to Coding Theory	September 5, 2001
Lecture 1	
Lecturer: Madhu Sudan	*Scribe: Ryan O'Donnell*

1.1 Error correcting codes and this class

This course teaches the mathematics underlying objects called error correcting codes (ECCs). We won't define what they are today - for reasons to be clarified later. For now it will suffice to know that they are combinatorial objects that possess certain extremal properties which are very useful for the communication and storage of information.

Modulo the definition of ECCs, we can still discuss the big outline of this course. The contents of this course can be divided into four roughly equal parts. Our first part will be explain some of the basic constructions of error-correcting codes. Next we will show "negative results" showing limitations, or bounds, on the performance of codes. The two parts above are essentially combinatorial in nature, with few computational themes. We will get into the computational aspects in the third and fourth parts of this course. The third part of the course is where we focus on algorithmic tasks associated with ECCs and some solutions to these tasks. Finally, in the fourth part we will discuss some consequences one can derive in computational complexity theory as a result of the existence of ECCs and from the algorithmic capabilities surrounding them.

1.1.1 Standard references

Since the subject is quite old, there are plenty of texts. Yet we won't follow any one of them. Here are comments on some of them.

1. The text by MacWilliams and Sloane [74].

(a) This is possibly the most referenced text in CS?

(b) Unfortunately, it is getting outdated pretty fast.

(c) Has detailed coverage of (too) many families of codes.

(d) Coverage of algorithms is not so good.

2. The text by van Lint [117].

    (a) The book is much more concise and handy than MacWilliams and Sloane. Easier to get to some of the results here, if they are available.

    (b) Still, the emphasis is more combinatorial than algorithmic.

3. A text by Blahut [19].

    (a) Book was targetted at engineers rather than mathematics, or so the author claims.

    (b) Yet the coverage is excellent, especially for insight, motivations, definitions, even algorithms

    (c) The main drawback is that it is out of print and not easily available (not in MIT library, e.g.).

4. The Handbook of Coding Theory [88].

    (a) Offers extensive coverage of many recent themes.

    (b) Sometimes excessive. (E.g. 130 pages of table of best known codes.)

    (c) But contains many interesting chapters. E.g., chapter on algebraic-geometry codes, and the chapter on deep-space applications.

1.2 Information Theory

Even though this course is on coding theory, we start with a brief coverage of information theory. Part of the reason is historic. Coding theory was initiated by two seminal papers:

1. In 1948, Shannon wrote a detailed treatise on the mathematics behind communication [100].

2. In 1950, Hamming, motivated by the task of correcting small number of errors on magnetic storage media, wrote the first paper introducing error-correcting codes [48].

In this lecture we will discuss the main results from the Shannon paper, which founded the theory of information, while also co-founding (with Hamming) the theory of error-correcting codes. The theory of information provides the motivation for many questions in coding theory and so we will study this first.

Shannon considered the problem of two parties, say Alice and Bob, who wish to communicate digitally. In particular, Alice wishes to send a message to Bob over a certain digital channel. Shannon considered both "noiseless" and "noisy" channels. We start by considering the "noiseless" case.

1.2.1 Noiseless coding

In this case, the channels perfectly transmits the bits Alice wants to send to Bob. However, we might imagine that the channel is slow, so our goal is to compress the information we want to send down to as few bits as possible. Then we send these bits across the channel for Bob to receive and decompress. We start with an example which shows how to effect such a compression scheme.

1.2.2 An example

Consider transmitting the contents of a piece of paper, which contains some handwritten material on it, by fax. If the paper has just a small amount of black text on a white background, when it is digitized it might consist of 99% 0's and 1% 1's. Let's say for the sake of argument that we want to transmit a message in which each bit is independently 0 with probability .99 and 1 with probability .01.

Consider the following encoding scheme: We split the message up into blocks of 10 bits. If the block is 0000000000, we send the bit 0. If the block is anything else, say x, we send the string $1x$. Now the expected length of the encoding of a block is:

$$1 \cdot \Pr[\text{block is } 0000000000] + 11 \cdot \Pr[\text{block is not } 0000000000] \quad = \quad 11 - 10q$$

where $q := \Pr[\text{block is } 0000000000] = .99^{10} \geq .9$. Hence the expected length of the encoding of one 10 bit block is at most 2 bits. Thus the original message has been compressed to within 20% of its length! (Food for thought: Do we really need the probabilistic model to acheive this compression?)

Can we do better? This is exactly the question Shannon raised and answered with his theorem on noiseless coding.

Entropy

To analyze how much a distribution could be compressed, Shannon introduced some mathematical definitions associated with information. The source of information is modeled as a probability distribution, and a message is a random variable drawn from this distribution. The goal of compression is to encode the support of the probability space (using, say binary strings) so as to minimize the expected length of the message. To any source (or more correctly, to any distribution), Shannon assigned a non-negative real number, that he termed "entropy", that measures the information content of the distribution. He then showed that this quantity (to within an additive term of *one bit*) measures the compressibility of a bit.

The simplest possible distribution is a coin flip, in which heads has probability p and tails has probability $1-p$. The entropy assigned to this is:

$$H(p) \stackrel{\text{def}}{=} p \log_2 \frac{1}{p} + (1-p) \log_2 \frac{1}{1-p}.$$

We can generalize this notion to arbitrary distributions.

Definition 1.1 *Let U be any finite set, and let D be a probability distribution on U, i.e., $D : U \to [0,1]$ with $\sum_{x \in U} D(x) = 1$. Let X be a random variable distributed according to D. Then the entropy* *of D is*[1]*:*

$$H(D) \stackrel{\text{def}}{=} \sum_{x \in U} D(x) \log_2 \frac{1}{D(x)}.$$

The noiseless coding theorem of Shannon essentially says the following:

Theorem 1.2 *For every finite set U and for every distribution $D : U \to [0,1]$, There exists an encoding function $\mathsf{Enc} : U \to \{0,1\}^*$ and a decoding function $\mathsf{Dec} : \{0,1\}^* \to U$ such that for every $x \in U$, $\mathsf{Dec}(\mathsf{Enc}(x)) = x$, and*

$$\mathrm{Exp}_{x \leftarrow D}[|\mathsf{Enc}(x)|] \in [H(D), H(D)+1],$$

where $|s|$ denotes the length of a string s. Conversely, no error-free encoding can do better.

Shannon's actual theorem is often stated differently, but we will state the theorem in the above since it is clearer. We don't prove the theorem here, but the main idea for the existence result is the following: (1) Round all probabilities down so that they become powers of 2 (i.e., construct D' such that for every x, $D'(x) = 2^{-i}$ for some integer i, and $D(x)/2 < D'(x) \leq D(x)$. (2) Show that there exists an encoding Enc that encodes an element x, whose new probability $D'(x)$ equals 2^{-i}, with i bits (so that no strings have the same encoding). (3) Conclude that this encoding has an expected length in the desired range! The converse is harder to prove - we won't get into it.

An elegant algebraical identity

The entropy function exhibits some very nice mathematical properties. For example if a distribution D decomposes into two independent distributions D_1 and D_2 (i.e., $U = U_1 \times U_2$, $D_1 : U_1 \to [0,1]$ and $D_2 : U_2 \to [0,1]$ and $D(x,y) = D_1(x)D_2(y)$), then $H(D) = H(D_1) + H(D_2)$. This fact can be use to prove an interesting algebraic identity, which is otherwise quite tricky to prove.

From the above property and the entropy of a bit, we find that the entropy in n independent p-biased coin flips is $nH(p)$. On the other hand, this should be equal to the entropy of the distribution D given by $D(x) := p^{\#1's \text{ in } x}(1-p)^{\#0's \text{ in } x}$. Using the second formula, we calculate this as:

$$\sum_{t=0}^{n} \binom{n}{t} D_t \log_2 \frac{1}{D_t}$$

where $D_t := p^t(1-p)^{n-t}$. Thus we get

$$\sum_{t=0}^{n} \binom{n}{t} D_t \log_2 \frac{1}{D_t} = nH(p).$$

Entropy and its variants play an important role in combinatorics and probability. Some very useful notions to keep track of in this area are those of mutual information, conditional entropy, and relative entropy!

To turn back to our example, what is the optimal compression factor for the case of our message to be faxed, whose bits were 1 with probability 1%? The answer is $H(.01) \approx 8\%$ — message can be compressed to within $H(.01)$ of their original length, if we know ones occur with probability about 1%.

[1]In lecture, we used a random variable X drawn according to D and defined this to be the entropy of X, rather than D. In these notes we will switch to the more appropriate notation, which defines it to be a function of the distribution and not the variable.

1.2.3 Noisy channels

For the purposes of this course, the more interesting notion of a channel is the class of "noisy" channels considered by Shannon — i.e., channels that flip some of the bits sent across them. The problem here is for Alice to encode the message she wishes to send in such a way that even if the channel corrupts some of the bits in the encoding, Bob will be able to decode the result into Alice's original message, with high probability. Shannon considered a large class of probabilistic models for noisy channels, and for each proved the surprising theorem that you could always overcome a constant error rate by sending an encoded message that was longer by a constant factor.

The general model Shannon gave for channels is as follows. There is an input alphabet Σ and an output alphabet Γ (both usually finite). Then we have a bipartite graph with Σ on the left and Γ on the right; each edge (σ, γ) is labeled with a probability of the channel converting a σ to a γ. (The channel operates independently on each character.) Of course, for each σ on the left, the sum of the labels on the edges touching it must be 1.

We will illustrate his results for such channels only in cases where $\Sigma \subseteq \Gamma$. (In such cases, it is clear what an error is.)

Two commonly considered channels:

1. Binary Symmetric Channel — the channel considered in the theorem. $\Sigma = \Gamma = \{0, 1\}$, $(0, 0)$ and $(1, 1)$ get probability $1 - p$, and $(0, 1)$ and $(1, 0)$ get probability p.
2. Binary Erasure Channel — in this channel, bits don't get flipped — rather they get erased. Specfically, $\Sigma = \{0, 1\}$, $\Gamma = \{0, 1, ?\}$, $(0, 0)$ and $(1, 1)$ get probability $1 - p$, and $(0, ?)$ and $(1, ?)$ get probability p.

Noisy coding theorem

The noisy coding theorem of Shannon is a powerful and general one. When specialized to the case of a binary symmetric channel with error probability p, we get the following result.

Theorem 1.3 *For every $p < 1/2$, there exists a constant $c < \infty$ and a pair of functions $E : \{0, 1\}^k \to \{0, 1\}^{ck}$, and $D : \{0, 1\}^{ck} \to \{0, 1\}^k$ with the following property: If we pick a message uniformly at random from $\{0, 1\}^k$, encode with E and then send the result across the noisy channel, and decode the result, then we recover the original message with probability* $1 - o(1)$.[2]

Theorem 1.3 applies — with different constants — to the binary erasure channel as well.

Hamming notations

We need just a few definitions before proceeding with the proof of the theorem.

Definition 1.4 *If x and y are in Σ^n, then the* Hamming distance between them is $\Delta(x, y) := \#$ of coordinates on which x and y differ.

[2]The $o(1)$ term depends only on k.

Definition 1.5 *The* Hamming ball of radius r centered at y is $B(y,r) := \{x \in \Sigma^n : \Delta(x,y) \le r\}$.

Definition 1.6 $\mathrm{Vol}(r,n)$ *denotes the volume of (any) radius-r ball in $\{0,1\}^n$; i.e., $|B(y,r)|$. Exactly, this quantity is $\sum_{i=0}^{r} \binom{n}{i}$. If we fix some $p > 0$ and let $n \to \infty$ then we get* $\mathrm{Vol}(pn,n) = 2^{(H(p)+o(1))n}$.

Proof of Theorem 1.3

Proof The proof is highly non-constructive; it uses the probabilistic method.

Let $n > k$ be decided upon later. Pick the encoding function $E : \{0,1\}^k \to \{0,1\}^n$ at random, i.e., for every $m \in \{0,1\}^k$, $E(m)$ is chosen uniformly at random from $\{0,1\}^n$, independently of all other choices.

The decoding function $D : \{0,1\}^n \to \{0,1\}^k$ works as follows. Given a string $y \in \{0,1\}^n$, we find (non-constructively) the $m \in \{0,1\}^k$ such that $\Delta(y, E(m))$ is minimized. This m is the value of $D(y)$.

We now prove that D and E have the required property, with high probability, over the random choice of E.

Fix $m \in \{0,1\}^k$ as also $E(m)$. (The rest of the proof will use the fact that $E(m')$ for $m' \neq m$ is still random.) the message being sent. Denote by y the corrupted version of $E(m)$ that is received by Bob. Let η denote the error vector $y - E(m)$. Note that the η is a random variable with each of its coordinates being 1 w.p. p and 0 w.p. $1-p$, independent of other coordinates.

Fix $\epsilon > 0$. Let $r = (p+\epsilon)n$. In order for $D(y) \neq m$ at least one of the following two events must occur:

1. $y \notin B(E(m), r)$ (i.e., too many errors occurred in transmission.)
2. There exists some $m' \neq m$ such that $E(m') \in B(y.r)$. (The errors take the received word too close to the encoding of some other message.)

In particular, if neither event occurs, then m is the unique message such that $E(m)$ is within a distance of r from y and so $D(y) = m$.

We show that for an appropriate choice of n, the events above happen with low probability. For the first event to happen, it must be that η has more than $p + \epsilon$ fraction of 1s. We can apply Chernoff bounds (see appendix at the end of this lecture) to see that:

$$\Pr_{\eta}[y \notin B(E(m), r))] \le 2^{-(\epsilon^2/2)n}.$$

For any $\epsilon > 0$, we can pick n to be large enough that the above quantity is as small as we want.

We now move onto the second event. First fix y and an $m' \neq m$ and consider the event that $E(m') \notin B(y,r)$. The probability of this event, taken over the random variable $E(m')$, is exactly $\mathrm{Vol}(B(y,r))/2^n$. Using the approximation $\mathrm{Vol}(B(y,pn)) \approx 2^{H(p)n}$ and ignoring ϵ (which is arbitrarily small), we find that for every $m' \neq m$.

$$\Pr_{E}[E(m') \in B(y,r)] \approx 2^{H(p)n-n}.$$

Using the union bound, we get that

$$\Pr_E[\exists m' \neq m \text{ s.t. } E(m') \in B(y,r)] \approx 2^{k-n+H(p)n}.$$

Thus we find that if $c > \frac{1}{1-H(p)}$, then $k/n < 1 - H(p)$ and thus the quantity above is less than one. This gives the choice of c that we need.

Our proof is not yet complete! Why? Well, we have argued that a fixed m is likely to be decoded correctly, but what about other messages? To complete the argument, we will actually need to lose a little bit.

Let $\delta = 2^{-(\epsilon^2/2)n} + 2^{k-n+H(p)n}$, denote the total probability of error in either of the two steps above. We have shown that for any fixed m, for a random choice of E and the associated D, the expected decoding error probability is at most δ. Thus we can now conclude that this expectation continues to hold if we make m a random variable chosen uniformly from $\{0,1\}^k$. We get

$$\mathrm{Exp}_{m,E,\eta}[D(E(m)+\eta) \neq m] \leq \delta.$$

In particular this implies that there exists an E such that for the corresponding D, we have

$$\mathrm{Exp}_{m,\eta}[D(E(m)+\eta) \neq m] \leq \delta.$$

This concludes the proof of Shannon's theorem. ∎

Capacity of the channel

Shannon's theorem above shows that if we are willing to slow down the effective transmission rate of the channel to $1/c < 1 - H(p)$, then we can acheive error-free communication with vanishingly small probability. Furthermore this quantity $1 - H(p)$ is only a function of the "noisy channel" and not of the number of bits we wish to transmit over it. I.e., the effective rate of transmission can be an absolute constant independent of n. Shannon called this quantity (the limiting rate of k/n, as $n \to \infty$) the *capacity* of the noisy channel.

For the Binary Symmetric Channel, how large can its capacity be? The proof above shows that the capacity, denoted $C_{\mathrm{BSC}}(p)$, is at least $1 - H(p)$. It is natural to ask if this quantity is an artifact of the proof, or is it the correct capacity for the channel. Shannon proved that the latter was the case. We will prove this in the next lecture, but first let us see why this is the case intuitively.

Suppose that we are actually is a setup there is a noiseless channel between Alice's location and Bob's, but this channel has been "hijacked" by Eve and Fred. Say Alice and Eve are in the same physical location while Fred and Bob are at the other. To send a message over, Alice must hand it over to Eve who then sends it through the channel (after some potential corruption). Similarly at the receiving end, Fred receives the message and hands it over to Bob. Suppose Eve and Fred want to use this channel to exchange some messages of their own (at Alice & Bob's expense). They do so by informing Alice and Bob that the channel is noisy with bits being flipped with probability p. They advise Alice and Bob to use some encoding/decoding schemes. Alice and Bob agree on an encoding scheme E and a decoding scheme D and in their naivette share these functions with Eve and Fred as well.

In truth it may be that Eve wishes to use the "noise" to send some messages of her own to Fred. Say she has a message η which is a plain paper image, where each pixel of the page is 1 independently with probability p. The way she sends η to Fred (at Alice's expense) is that when Alice gives her a encoded message $E(m)$ to transmit, Eve sends over $E(m) + \eta$. Fred receives $y = E(m) + \eta$ (the channel does not introduce any noise) at the receiving end and passes it on, untampered, to Bob, but also retains a copy. As far as Alice and Bob are concerned, nothing mailicious is occuring - with high probability $D(y) = m$ and so they are exchanging messages at capacity of the "noisy channel". But note the situation w.r.t. Eve and Fred. Fred also knows E and D and can compute $\eta = y - E(D(y)) = E(m) + \eta - E(m)$, with high probability. So Eve and Fred are also exchanging messages among themselves (with some small probability of exchanging incorrect messages). But now if we consider Alice & Eve together at one end of the noiseless channel, and Bob & Fred together at the other end, the parties are exchanging bits at a rate of at least $C(p) + H(p)$ across the noiseless channel (assuming we believe the tightness of the noiseless coding theorem). Since we are normalizing so that the rate the noiseless channel is 1, we get $C(p) + H(p) < 1$! This is the link between the noisy case and the noiseless case of the Shannon theorems.

Appendix

Notation used in the lecture. We mention some notation that we used earlier or may use in later lectures. $\mathbb{Z}$ denotes the set of integers, $\mathbb{Z}^{\geq 0}$ denotes the set of non-negative integers, and $\mathbb{Z}^+$ the set of positive integers. $\mathbb{R}$ denote the set of reals, and $\mathcal{Q}$ the rationals. For real numbers a and b, the notation $[a, b]$ stands for the closed interval from a to b, i.e., the set $\{x \in \mathbb{R} | a \leq x \leq b\}$, while (a, b) is the open interval between a and b. For an integer k, we will use $[k]$ to denote the set $\{1, \ldots, k\}$. If D is a distribution on the universe U, then $X \leftarrow D$ denotes a random variable X drawn from U according to the distribution D. For an event $\mathcal{E} \subseteq U$, the quantity $\Pr_{X \leftarrow D}[X \in \mathcal{E}]$ denotes the probability of the event $\mathcal{E}$ when X is chosen according to D. For a real-valued function $f : U \to \mathbb{R}$, the quantity $\mathrm{Exp}_{X \leftarrow D}[f(X)]$ denotes the expected value of $f(X)$ when X is chosen according to D. When the distribution D is clear from context, we may abbreviate these quantities to $\Pr[\mathcal{E}]$ and $\mathrm{Exp}_X[f(X)]$. Similarly $\mathrm{Var}_{X \leftarrow D}[f(X)]$ denotes the variance of $f(X)$ (i.e., $\mathrm{Var}(f(X)) = \mathrm{Exp}[(f(X))^2] - \mathrm{Exp}[f(X)]^2$).

Some basic probability facts Here is a quick recap of basic facts on probability and expectations.

Probability One of the most used facts on probability is the union bound: $\Pr[\mathcal{E}_1 \cup \mathcal{E}_2] \leq \Pr[\mathcal{E}_1] + \Pr[\mathcal{E}_2]$. Note that the bound makes no "independence" assumptions.

Expectations Analogous to the above we have: $\mathrm{Exp}[X_1 + X_2] = \mathrm{Exp}[X_1] + \mathrm{Exp}[X_2]$. Note that this is an equality! If random variables are independent, then we get a product relationship $\mathrm{Exp}[X_1 X_2] = \mathrm{Exp}[X_1]\mathrm{Exp}[X_2]$.

Converting probabilities to expectations: Since expectations are more amenable to algebraic manipulations, it is often useful to convert statements on probability to statements of events. The standard way to do this is to use "indicator variables". For an event $\mathcal{E}$, let $I_{\mathcal{E}}$ be the 0/1-valued variable given by $I_{\mathcal{E}}(X) = 1$ if $X \in \mathcal{E}$ and $I_{\mathcal{E}}(X) = 0$ otherwise. Then we have $\mathrm{Exp}_X[I_{\mathcal{E}}(X)] = \Pr[\mathcal{E}]$.

Converting expectations to probabilities: Since probabilities are the quantities that have more intuitive meaning, these are the more standard targets of our investigation. Since expectations figure in proofs, we would like to find ways to convert statements on expectations back into probability statements. This conversion is not standard. Several "tail inequalities" are used to acheive this conversion, e.g., Markov's, Chebychev's, and Chernoff's: We state them below:

Markov's inequality For a non-negative random variable X and positive real α, then $\Pr[X \geq \alpha] \leq \frac{E[X]}{\alpha}$.

Chebychev's inequality In its general form, this inequality is just an application of Markov's inequality to the random variable Y^2, for arbitrary Y. In the general form, it is quite hard to see its strength, so we give a special form.

Let $Y = \sum_{i=1}^{n} Y_i$, where the Y_i's identically distributed random variables that are pairwise (but not fully) independent. Let $\text{Exp}[Y_i] = \mu$ and $\text{Var}[Y_i] = \sigma^2$. Then for $\lambda > 0$, we have $\Pr[Y \geq (\mu + \lambda)n] \leq \frac{\sigma^2}{n\lambda^2}$.

Chernoff bounds If $Y_1, \ldots, Y_n$ are completely independent it is possible to get stronger bounds on the probability that their sum deviates much from their expectation. We consider the special case of variables taking values in the interval $[0, 1]$

Let $Y_1, \ldots, Y_n$ be independent and identically distributed random variables taking values in the interval $[0, 1]$ with mean μ. Let $Y = \sum_i Y_i$. Then $\Pr[Y \geq (\mu + \lambda)n] \leq e^{-(\lambda^2/2)n}$, where e is the base of the natural logarithm.

For further elaboration on probabilities and expectations one may consult the text on Randomized Algorithms [82].

Chapter 2

6.897 Algorithmic Introduction to Coding Theory	September 10, 2001
Lecture 2	
Lecturer: Madhu Sudan	*Scribe: Omprakash Gnawali*

Today we will cover the following topics:

- Converse to Shannon's coding theorem.
- Some remarks on Shannon's coding theorem.
- Error correcting codes.
- Linear codes.

2.1 Converse to Shannon's coding theorem

Recall that the Binary Symmetric Channel (BSC) with parameter p is the channel that transmits bits, flipping each transmitted bit with probability p independent of all other events. Lets start by recalling Shannon's coding theorem informally.

> Over the BSC with parameter p, it is possible to transmit information at any rate less than $1 - H(p)$.

(To give a sense of how to make the above formal, here is the formal version in all its quantified glory: "For every $p < \frac{1}{2}$ and $\epsilon, \delta > 0$, there exists an $n_0 < \infty$ such that for every $n \geq n_0$ and $k \leq (1 - H(p) - \epsilon)n$, there exist functions $E : \{0,1\}^k \to \{0,1\}^n$ and $D : \{0,1\}^n \to \{0,1\}^k$ such that

$$\Pr_{\eta \leftarrow D_{p,n}, m \leftarrow U_k} [D(E(m) + \eta) = m] \geq 1 - \delta,$$

where U_k is the uniform distribution on $\{0,1\}^k$ and $D_{p,n}$ is the distribution on n bits chosen independently with each bit being 1 with probability p.")

We will now proof a converse to this theorem.

Theorem 2.1 *For every $p < \frac{1}{2}$ and $\epsilon, \delta > 0$, there exists an $n_0 < \infty$ such that for every $n \geq n_0$, $k \geq (1 - H(p) + \epsilon)n$, and functions $E : \{0,1\}^k \to \{0,1\}^n$ and $D : \{0,1\}^n \to \{0,1\}^k$, it is the case that*

$$\Pr_{\eta \leftarrow D_{p,n}, m \leftarrow U_k} [D(E(m) + \eta) \neq m] \geq 1 - \delta,$$

where U_k is the uniform distribution on $\{0,1\}^k$ and $D_{p,n}$ is the distribution on n bits chosen independently with each bit being 1 with probability p.

Ignoring all quantifiers above, the essense is that if we are trying to send information at a rate of $1 - H(p) + \epsilon$, then the decoding is erroneous with probability almost 1, *no matter* which encoding and decoding function we use.

Proof The hard part of this proof is deciding how to deal with the encoding and decoding functions which are completely arbitrary! Turns out, we will ignore the encoding function entirely, and ignore the decoding function almost entirely! The main focus is on the error, and the fact that the error distributes the transmitted word over a large space of possibilities (and so any decoding function should be helpless). Specifically, we note the following:

- The number of errors is unlikely to be too small or too large: Formally, for every m and E,

  $$\Pr_{\eta} [E(m) + \eta \in B(E(m), (p - \epsilon)n)] = \Pr_{\eta} [\eta \in B(\mathbf{0}, (p - \epsilon)n)] \leq e^{-\epsilon^2 n}. \tag{2.1}$$

  The equality above is obtained by simply translating the center of the ball from $E(m)$ to the origin $\mathbf{0}$, the string consisting of all 0's. The inequality above is a straightforward application of Chernoff bounds - however, this time we are using the fact that it proves that a random variable is not likely to take on a value much less than its expectation. Similarly, we get that

  $$\Pr_{\eta} [E(m) + \eta \notin B(E(m), (p + \epsilon)n)] = \Pr_{\eta} [\eta \in B(\mathbf{0}, (p - \epsilon)n)] \leq e^{-\epsilon^2 n}.$$

- Given that the error is large (but not too large), no single point in the space has a high probability of being the received word. Specifically, for every m, E and $y \in B(E(m), (p + \epsilon)n) - B(E(m), (p - \epsilon)n)$,

  $$\Pr_{\eta} [E(m) + \eta = y] \leq \frac{n}{2^{H(p-\epsilon)n}}. \tag{2.2}$$

  To see why the above is true, let $R = \Delta(y, E(m))$ be the Hamming distance between y and $E(m)$. Note we have $R \in [(p - \epsilon)n, (p + \epsilon)n]$. Let N_R be the number of binary vectors with R ones and $n - R$ zeroes. Since all error patterns η with the same number of errors are equally likely, we note we the probability of having any fixed vector containing exactly R ones as the error vector is at most $\frac{1}{N_R}$. Thus to upper bound the probability of the event in question it suffices to lower bound N_R for $R \in [(p - \epsilon)n, (p + \epsilon)n]$. Using the fact that $N_R = \mathrm{Vol}(R, n) - \mathrm{Vol}(R - 1, n)$, and the fact that N_R is increasing for R in the range $[0, \frac{n}{2}]$, we get than $N_R \geq \frac{\mathrm{Vol}(R,n)}{n} \geq \frac{\mathrm{Vol}((p-\epsilon)n,n)}{n}$. Now using the fact that $\mathrm{Vol}((p - \epsilon)n, n) \approx 2^{-H(p-\epsilon)n}$, we get that the probability $\eta = y - E(m)$ is at most $\frac{n}{2^{H(p-\epsilon)n}}$.

To continue the proof, we finally look at the decoding function (though even this look will be very superficial). Let $K = 2^k$ and let $\{m_1, \ldots, m_K\}$ denote the K possible messages. Let $S_i = \{y | D(y) = m_i\}$ be the set of received words that are decoded to the ith message. The only property we use about the decoding is that $\sum_{i=1}^{K} |S_i| = 2^n$, i.e., the decoding is a *function*! (On the other hand, there is little else to use!) We now use the observations in the previous paragraph to prove that the decoding function is not very likely to succeed.

Let ρ be the probability of decoding successfully. In order for the decoding to succeed, we must pick some message m_i to encode and transmit, and the error vector η must be such that the received vector $y = E(m_i) + \eta$ must lie in S_i. This gives:

$$\rho = \sum_{i=1}^{K} \sum_{y \in S_i} \Pr_{m,\eta}[m = m_i \text{ and } \eta = y - E(m_i)] = \Pr_m[m = m_i] \Pr_\eta[\eta = y - E(m_i)],$$

where the second equality follows from the fact that the events considered are independent. Fixed m_i and let us bound the inner summation above. The probability that m equals m_i is exactly $1/K = 2^{-k}$. The event that $\eta = y - E(m_i)$ is independent of m and so we can estimate this quantity separately. Fix m_i. Let $U = B(E(m_i), (p-\epsilon)n)$ and $V = \{0,1\}^n - B(E(m_i), (p+\epsilon)n)$ (i.e., U is the points too close to $E(m)$ and V the points too far from $E(m)$. Then

$$\begin{aligned}
&\sum_{y \in S_i} \Pr_\eta[\eta = y - E(m_i)] \\
\leq\ & \sum_{y \in U} \Pr_\eta[\eta = y - E(m_i)] + \sum_{y \in V} \Pr_\eta[\eta = y - E(m_i)] + \sum_{y \in S_i - U - V} \Pr_\eta[\eta = y - E(m_i)] \\
\leq\ & 2e^{-\epsilon^2 n} + |S_i| \frac{n}{2^{H(p-\epsilon)n}},
\end{aligned}$$

where the second inequality above follows from Equations (2.1) and (2.2). Combining the above, we have:

$$\begin{aligned}
\rho &\leq \sum_{i=1}^{K} 2^{-k} \left(2e^{-\epsilon^2 n} + \frac{n|S_i|}{2^{H(p-\epsilon)n}} \right) \\
&= 2e^{-\epsilon^2 n} + \frac{n 2^{-k}}{2^{H(p-\epsilon)n}} \left(\sum_{i=1}^{K} |S_i| \right) \\
&= 2e^{-\epsilon^2 n} + n 2^{-k - H(p-\epsilon)n + n}
\end{aligned}$$

The theorem follows from the fact that for every $\epsilon, \delta > 0$ we can pick n_0 large enough so that for every $n \geq n_0$, it is the case that $2e^{-\epsilon^2 n} \leq \delta/2$ and $n 2^{-k - H(p-\epsilon)n + n} \leq \delta/2$ (assuming $k \geq (1 - H(p) + \epsilon)n$). ■

Recall that at the end of the previous lecture we showed that if we assumed the noiseless coding theorem is tight (i.e, has a converse) then the converse to the noisy coding theorem follows. While the theorem above does not imply a converse to the noiseless coding theorem, the proof technique is general enough to capture the noiseless coding theorem as well. This motivates the following exercise.

Exercise: Prove converse of the noiseless coding theorem (from Lecture 1).

2.2 Remarks on Shannon's Theorem

2.2.1 Discrete Memoryless Channels

Shannon's coding theorem is, of course, much more general that what we have presented. We only presented the result for the case of the Binary Symmetric Channel. For starters, the result can be generalized to the case of all "Discrete Memoryless Channels (DMCs)". Such channels are characterized by two finite sets — Σ representing the input alphabet of the channel and Γ representing the output alphabet of the channel — and a transition probability matrix $P = \{p_{\sigma\gamma}\}$, where $p_{\sigma\gamma}$ denotes that probability that the output alphabet is γ given than the input alphabet is σ. We require that $\sum_{\gamma\in\Gamma} p_{\sigma\gamma} = 1$ for every $\sigma \in \Sigma$, so that this definition makes sense. When we attempt to transmit a sequence of symbols from Σ over this channel, it behaves on each element of the channel independently and produces a sequence of elements from Γ, according to the transition probability matrix P.

Given such a channel, characterized by P, Shannon gave a procedure to compute the capacity of the channel. This capacity relates to the mutual information between two random variables.

Given a distribution D_Σ over Σ, let σ be a random variable chosen according to D_Σ. Pick γ at random from Γ with probability $p_{\sigma\gamma}$. Denote by $D_{\Sigma,\Gamma}$ the joint distribution on the pairs (σ, γ) so generated, and by D_Γ the marginal distribution of γ. Since D_Σ, D_Γ and $D_{\Sigma,\Gamma}$ are all distributions on finite sets, their entropy is well defined. Define the "mutual information" between variables σ and γ (or more correctly of the transition matrix P with initial distribution D_Σ) to be $H(D_\Sigma) + H(D_\Gamma) - H(D_{\Sigma,\Gamma})$.

Shannon's theorem showed that the capacity of the channel characterized by P, is the maximum over all distributions D_Σ, of the mutual information between σ and γ. He also gave a linear system whose solution gave the distribution that maximizes this information.

2.2.2 Markovian Channels

Shannon's theory extends even further. Natural scenarios of error may actually flips bits with some correlation rather than doing so independently. A subclass of such correlations is given by "Markovian channels", where the channel can be in one of several (finitely many) states. Depending on which state the channel is in, the probability with which it makes errors may be different.

Such models are useful in capturing, say, "burst error" scenarios. In this situation the channel makes sporadic sequences of many errors. One can model this source by a channel with two states (noisy/normal) with two different channel characteristics for the two states. (see Figure 2.1). When in the "noisy" state the channel flips every bit, say, with probability $\frac{1}{2}$ (and so a channel that is perpetually in a noisy state can transmit no information). When in the normal state, however, the channel flips bits with only a small probability, say p. Further, if the channel is in a given state at time t it tends to stay in the same state at time $t + 1$ with probability $1 - q$ and tends to flip its state with a small probability q. Is it possible to transmit information on such a channel? If so, at what rate? Working this out would be a good exercise!

Shannon's theory actually gives the capacity of such a channel as well. Deciphering what the theory says and unravelling the proofs would be a good topic for a term paper.

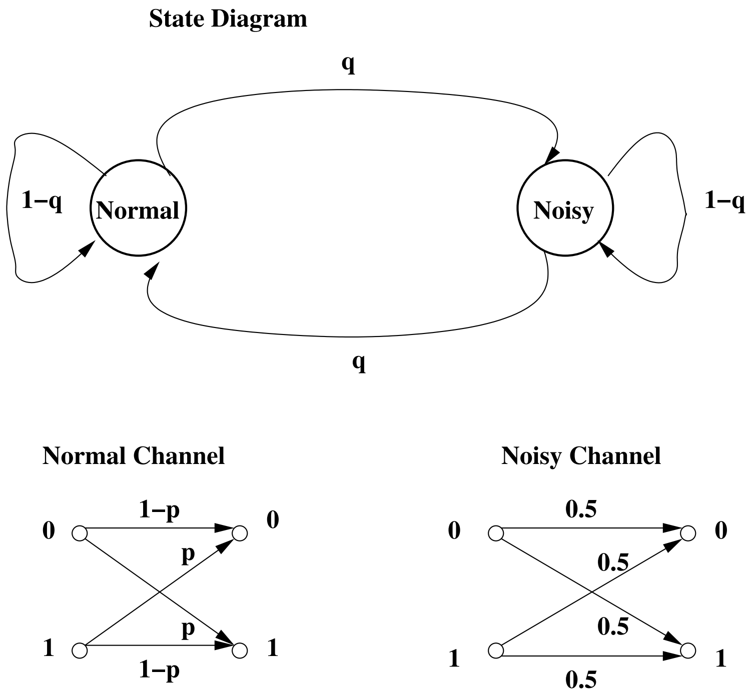


Figure 2.1: State diagram of a simple burst error channel

2.2.3 Zero Error Capacity of a channel

An interesting by product of the Shannon theory is the so called "Zero error capacity" of a channel. To motivate this notion, let us consider a "stuck typewriter". Suppose the keys in the typewriter are sticky and likely to produce the wrong symbol when you hit it. Further suppose that the error pattern is very simple. If you hit a letter of the keyboard, you get as output either the correct letter or the next letter of the alphabet. and that such an error happens with probability $\frac{1}{2}$ for every keystroke independently. In other words, typing A results in A or B, typing B results in B or C, etc. and typing Z results in Z or A (so we have a wrap around). (See Figure 2.2.)

Some analysis of this channel reveals that it has a channel capacity of $\log_2 13$. I.e., each keystroke is capable, on the average of conveying one of 13 possibilities. If the typewriter had not been stuck, it would have had a capacity of $\log_2 26$. Thus the error cuts down the number of possibilities per stroke by a factor of 2.

If we think hard (actually may be not even so hard) we can see a way of achieving this capacity. Simply don't use the even-numbered letters of the alphabet (B,D,F, etc.) and work only with the odd-numbered ones (A, C, E etc.). Since there is no possibility of confusion within the odd-numbered letters, there is no ambiguity in the message (if an A or B is received, A must have been the keystroke typed, if C or D is received, C is the keystroke etc.). The interesting aspect of this way of using the channel is that we acheive the capacity, *with zero error*! This motivates a general concept: The *Zero Error Capacity* of a channel is informally defined as the optimal rate of transmission that can be achieved while maintaining a zero probability of decoding error.

In the case of the stuck typewriter the zero error capacity equals the Shannon capacity. This is

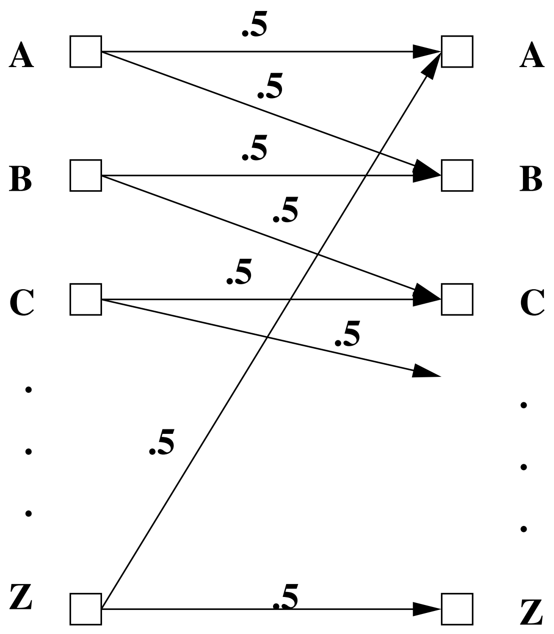


Figure 2.2: Channel for a stuck typewriter

not always the case. For example, the zero error capacity of the binary symmetric channel for any $p \neq 0, 1$ is zero! (Any string has positive probability of being corrupted into to any other string.) It is true that the zero error capacity is less than or equal to the Shannon capacity. Computing the Shannon capacity of even simple channels is non-trivial, and in general this function is not known to be computable.

A simple illustrative example is the zero error capacity of a stuck keyboard with only five keys (or in general an odd number of keys)! Then it is no longer clear what the zero error capacity of this channel is. In 1979, L. Lovász [71] wrote a brilliant paper that showed how to compute the Shannon capacity of several graphs (and in general give a lower bound on the Shannon capacity). In particular he shows that the Shannon capacity of a stuck typewriter with 5 keys is $\sqrt{5}$ - an irrational number! This paper has played a pioneering role in computer science leading to the notion of semi-definite programming and its consequences on combinatorial optimization.

Term paper topic: Study the zero error capacity of arbitrary channels and survey the work of Lovász and successors.

This will terminate our discussion of the Shannon based theory. As mentioned in the first lecture, Shannon's original paper [100] and the text by Cover and Thomas [26] are excellent sources for further reading.

2.3 Error correcting codes

Shannon's theory, while providing exact results for the rate at which one can communicate on a noisy channel, are unfortunately highly non-constructive. Specifically the two key ingredients: the *encoding* and *decoding* functions are totally non-constructive. In order to get some sense of how to make these results constructive, one has to examine the encoding function and see what properties about it are useful. Shannon noticed that a result of Hamming indeed does so, and that this may be a step in making his results more constructive. In retrospect it seems this was crucial in making Shannon's results constructive. This is the theory of error-correcting codes, as initiated by the work of Hamming [48]. Hamming focussed on the set of strings in the image of the encoding map, and called them "error-correcting codes". He identified the distance property that would be desirable among the codes and initiated a systematic study. We develop some notation to study these notions.

2.3.1 Notation

We consider codes over some alphabet Σ and reserve the letter q to denote the cardinality of Σ. It is often helpful to think of $\Sigma = \{0, 1\}$ and then the codes are termed binary codes.

We consider transmissions of sequences of n symbols from Σ from sender to receiver. Recall that for two strings $x, y \in \Sigma^n$, the Hamming distance between x and y, denoted $\Delta(x, y)$, is the number of coordinates where x differs from y. We note that the Hamming distance is indeed a metric: i.e., $\Delta(x, z) = \Delta(z, x) \leq \Delta(x, y) + \Delta(y, z)$ and $\Delta(x, y) = 0$ if and only if $x = y$.

A code C is simply a subset of Σ^n for some positive integer n. The minimum distance of a code C, denoted $\Delta(C)$, is given by $\Delta(C) = \min_{x,y \in C, x \neq y}\{\Delta(x, y)\}$. The Hamming theory focusses on the task of constructing (or showing the existence of) codes with large minimum distance and large cardinality.

There are four fundamental parameters associated with a code C:

- Its *block length*: n, where $C \subseteq \Sigma^n$.
- Its *message length*: $k = \log_q |C|$. (To make sense of this parameter, recall that we are thinking of the code as the image of an encoding map $E : \Sigma^k \to \Sigma^n$ and in this case $\log_q |C| = k$ is the length of the messages.)
- Its *minimum distance* $d = \Delta(C)$.
- Its *alphabet size*: $q = |\Sigma|$.

It is often customary to characterize a code by just the four parameters it achieves and refer to such a code as an $(n, k, d)_q$-code.

2.3.2 Broad Goals of Coding Theory

In a nutshell the broad goal of coding theory can be stated in one of the four ways below, where we fix three of the four parameters and try to optimize the fourth. The correct optimizations are:

- Given k, d, q find an $(n, k, d)_q$ code that minimizes n.
- Given n, d, q find an $(n, k, d)_q$ code that maximizes k.
- Given n, k, q find an $(n, k, d)_q$ code that maximizes d.
- Given n, k, d find an $(n, k, d)_q$ code that minimizes q.

The first three choices are self-explanatory. It is always desirable to have a small block length, large message length, and large distance. However it is not so immediate that minimizing q is the right thing to do (in particular, we don't have a monotonicity result). However empirically (and almost certainly) it seems to be the case that one can get get values of the parameters for larger values of q and getting good parameters for small values of q is the challenging part. Furthermore, building codes with large q and then trying to reduce q is a very clever way of getting good codes. So we will keep this version in mind explicitly.

2.3.3 Error Correcting Codes

Why are we interested in codes of large minimum distance? It may be worth our while to revisit Hamming's paper and see what he had to say about this. Hamming actually defined three related properties of a code C.

1. The minimum distance of C, which we have already seen.
2. The error detection capacity of C: A code C is e-error detecting if under the promise that no more than e errors occur during transmission, it is always possible to detect whether errors have occured or not, and e is the largest integer with this property. Hamming notes that a e-error correcting code has minimum distance $e + 1$.
3. The error correction capacity of C: A code C is t-error correcting if under the promise that no more than t errors occur during transmission, it is always possible to determine which locations are in error (information-theoretically, but not necessarily efficiently) and correct them, and if t is the largest integer with this property. Hamming notes that a t-error correcting code has minimum distance $2t + 1$ or $2t + 2$.

Thus the minimum distance of a code is directly relevant to the task of correcting errors and we will focus on this parameter for now. Later (in the second part of the course) we will turn our attention to the question - how can we make these error-detection and error-correction capabilities algorithmic.

2.3.4 Some simple codes

The famed Hamming codes are codes of minimum distance three. Even though Hamming's name is associated with distance-3 codes, he also gave, in the same paper. codes of distance two and four! Let us start even slower and describe the distance one codes first!

- $d = 1$: This is trivial. All we want is that the encoding function be injective. So the identity function works and gives the best possible $(n, n, 1)_q$ code.
- $d = 2$, This is already interesting. A simple way to achieve distance 2 is to append the parity of all the message bits to message and thus get a code with $n = k + 1$, i.e., an $(n, n-1, 2)_2$ code. For general q, one identifies Σ with $\mathbb{Z}_q$, the additive group of integers modulo q and uses (instead of the parity check bit) the check symbol that is the sum of all message symbols over $\mathbb{Z}_q$. This gives, for every q, a $(n, n-1, 2)_q$ code.
- $d = 3$, non-trivial interesting case.

Interpolating from the first two examples, one may conjecture that a $(n, n-d+1, d)_q$ code is always possible. This turns out not to be the case and in fact $d = 3$ already gives a counterexample to this conjecture. Hamming gave codes for this case and proved their optimality. We will describe the codes in the next lecture.

6.897 Algorithmic Introduction to Coding Theory September 12, 2001

Lecture Notes on Algebra

Lecturer: Madhu Sudan *Scribe: Madhu Sudan*

These notes describe some basic algebraic structures that we will encounter during this course, including:

- Finite fields of all sizes (and shapes).
- (Univariate and multivariate) polynomials over finite fields in one or more variables.
- Vector spaces over finite fields (or Linear algebra).

Unfortunately, there is no simple order in which one can present all these objects — their presentation is interleaved for essential reasons. Polynomials are typically defined with coefficients from fields. Fields are constructed by constructing polynomial rings and then reducing them modulo irreducible polynomials. Linear algebra needs to be based on fields. But it also provides convenient ways of looking at fields. We will try to describe all these connections below. Mostly we are interested in computational and combinatorial consequences. We would like to see how to represent fields so as to perform elementary manipulations efficiently. We would also like to know if some computational problems from linear algebra can be solved efficiently. We are also interested in combinatorial questions such as: How often can a polynomial evaluate to zero? How does one prove that this can not happen too often? The notes below present answers to such questions.

2.4 Main definition

Since we are interested in polynomials over fields, it would be nice to know the basic algebraic structures which unify both fields and polynomials. Commutative rings are such structures and we define them below.

Definition 2.2 (Commutative Rings and Fields) *A* commutative ring *is given by a triple $(R, +, \cdot)$, where R is an arbitrary set containing two special elements 0 and 1 and $+, \cdot$ are functions mapping $R \times R$ to R satisfying the following properties for every triple $a, b, c \in R$:*

Associativity: *Both $+$ and $\cdot$ are associative, i.e., $a+(b+c) = (a+b)+c$ and $a\cdot(b\cdot c) = (a\cdot b)\cdot c$.*

Commutativity *Both $+$ and $\cdot$ are commutative, i.e., $a + b = b + a$ and $a \cdot b = b \cdot a$.*

Distributivity: *$\cdot$ distributes over $+$, i.e., $a \cdot (b + c) = a \cdot b + a \cdot c$.*

Identities: *$a + 0 = a$ and $a \cdot 1 = a$.*

Additive Inverses: *For every $a \in R$, there exists an additive inverse $-a \in R$ such that $a + (-a) = 0$.*

If in addition, every non-zero element has a multiplicative inverse, then R is a field*. (I.e., for every $a \in R - \{0\}$, there exists an $a^{-1} \in R$ such that $a \cdot a^{-1} = 1$.)*

Often we will skip the operators $+$ and $\cdot$ and simply refer to the set R as the ring (with addition and multiplication being specified implicitly). Commutative rings form the foundation for much of the elegant results of algebra and algebraic geometry. Within the class of commutative rings, one can get nicer and nicer domains (rings with nicer and nicer properties) and this culminates with the notion of a field. Informally, rings allow the operations of addition, subtraction and multiplication, while a field also allows division. We will see some of the intermediate notions later. Right now we turn to polynomials.

2.5 Polynomial rings

Given any ring R, and a symbol t (usually referred to as an indeterminate, one can create a ring $R[t]$ of polynomials over R. Such a ring inherits most of the nice properties of the underlying ring. Below is a formal definition of the ring of polynomials.

Definition 2.3 *Given a commutative ring R and indeterminate t, the set $R[t]$ has as its elements finite sequences of R, with the sequence $f = \langle f_0, \ldots, f_l \rangle$ being interpreted as the formal sum $\sum_{i=0}^{l} f_i t^i$. Addition and multiplication over $R[t]$ are defined accordingly, i.e., if $f = \langle f_0, \ldots, f_l \rangle$ and $g = \langle g_0, \ldots, g_k \rangle$ with $l \leq k$ then $f + g = \langle f_0 + g_0, \ldots, f_l + g_l, g_{l+1}, \ldots, g_k \rangle$ and $f \cdot g = \langle h_0, \ldots, h_{l+k}$ where $h_i = \sum_{j=0}^{i} f_j g_{i-j}$. For a polynomial $f \in R[t]$, given by $f = \sum_{i=0}^{d} f_i t^i$, we define its* degree, *denoted* $\deg(f)$ *to be the largest index d' such that $f_{d'}$ is non-zero.*

Proposition 2.4 *For every commutative ring R and indeterminate t, $R[t]$ is a commutative ring.*

The most natural ring of polynomials that we will encounter are the ring of polynomials over some (finite) field F, say $F[x]$. Now we can adjoin a new indeterminate y to this ring to another ring $F[x][y]$. We will use the notation $F[x, y]$ to denote such a ring whose elements are simply polynomials in two variables x and y. In particular $F[y][x] = F[x][y] = F[x, y]$. Continuing this way, adjoining m variables $x_1, \ldots, x_m$ to F for some integer m, we get the space of m-variate polynomials $F[x_1, \ldots, x_m]$. It is also possible to define this ring directly and we do so in order to define various notions of degree associated with it.

Definition 2.5 (Multivariate polynomial rings) *Given a ring R and indeterminates $x_1, \ldots, x_m$ the m-variate polynomial ring over R, denoted $R[x_1, \ldots, x_m]$ has as its elements finite sequences indexed by d-tuple of non-negative integers $f = \langle f_{i_1, \ldots, i_m} \rangle_{0 \leq i_j \leq d_j}$. The element represents the formal sum $\sum_{i_1, \ldots, i_m} f_{i_1, \ldots, i_m} x_1^{i_1} \cdots x_m^{i_m}$. Addition and multiplication are interpreted appropriately. The* x_j-degree *of f, denoted* $\deg_{x_j}(f)$, *is the largest index d_j' such that there exist indices $i_1, \ldots, i_m$ such that $f_{i_1, \ldots, i_{j-1}, d_j, i_{j+1}, \ldots, i_m}$ is non-zero. The* total degree *of f is the largest sum $\sum_{j=1}^{m} i_j$, among tuples $i_1, \ldots, i_m$ for which $f_{i_1, \ldots, i_m}$ is non-zero.*

We will come back to multivariate polynomials later. Right now we move on to descriptions of fields and this will need univariate polynomials.

2.6 Finite Fields

Fields are the nicest of algebraic structures. that allow all sorts of manipulations efficiently. In particular we can not only define addition and multiplication, but also subtraction $(a - b = a + (-b))$

and division ($a/b = a \cdot b^{-1}$). The most familiar examples of fields are the field of rational numbers $\mathcal{Q}$ and the field of real numbers $\mathbb{R}$. For our purposes fields that have only a finite number of elements are much more important. The following theorem tells us what kind of finite fields exist.

Theorem 2.6 *For a positive integer q, a field F of cardinality q exists if and only if $q = p^l$ for a prime p and positive integer l.*

We use the notation $\mathbb{F}_q$ to denote the field with q elements. Since we eventually intend to use the fields computationally, we will need to know a little more about such fields. Specifically, given q how can one represent the elements of the field $\mathbb{F}_q$? Given (such representations of elements) $\alpha, \beta \in \mathbb{F}_q$, how can we compute (representation of) $\alpha + \beta$, $\alpha - \beta$, $\alpha \cdot \beta$ and α/β? We answer these questions below:

Prime fields. If $q = p$ for a prime p, then the field $\mathbb{F}_q$ is simply the field of arithmetic modulo p. Thus the natural way to represent the elements of $\mathbb{F}_q$ is using the integers $\{0, \ldots, p-1\}$. It is easy to carry out addition, multiplication, and subtraction in the field can be carried out in time $\text{poly}\log q$. Fermat's little theorem also tell us that if $\beta \neq 0$, then $\beta^{-1} = \beta^{p-2} (\text{mod} p)$ and by using a fast modular exponentiation algorithm, β^{-1} can also be computed in time $\text{poly}\log q$. (Actually addition, multiplication, and subtraction can be computed in time $O(\log q \text{poly} \log\log q)$.)

Before going on to describing fields of cardinality p^l, where $l \geq 2$, we need to define the notion of irreducible polynomials.

Definition 2.7 (Irreducible polynomials) *Given a ring $F[t]$ of polynomials over a field F, a polynomial $f \in F[t]$ is said to be* reducible *if there exist polynomials g and h in $F[t]$ of degree at least one such that $f = g \cdot h$. f is said to be irreducible if no such polynomial exists.*

We are now ready to describe the remaining finite fields.

Prime power fields. Let $q = p^l$ for prime p and positive integer l. Suppose f is an irreducible polynomial of degree l in $\mathbb{F}_p[t]$. Then $\mathbb{F}_q \cong \mathbb{F}_p[t]/(f)$, i.e., the ring of polynomials in t reduced modulo f. Specifically, the elements of $\mathbb{F}_p[t]/(f)$ are polynomials of degree strictly less than l. (Note that there are exactly p^l such polynomials.) Addition is straightforward polynomial addition. (Note that the degree of the sum is less than l if both polynomials have degree less than l.) Multiplication is performed modulo f, i.e., given g and h we compute $p = gh$ using regular polynomial multiplication and then compute the remainder when p is divided by f. This is a polynomial r of degree less than l and we define $g \cdot h$ to be r in the "field" $\mathbb{F}_p[t]/(f)$. Fermat's little theorem applied to groups shows that $g^{-1} = g^{q-2}$ in this field also.

Exercise: Verify that $\mathbb{F}_q$ as described above satisfies the definitions of a field.

The above construction is would not be very useful, if it weren't for the fact that irreducible polynomials exist and can be found efficiently.

Theorem 2.8 (cf. [102]) *For every prime p and positive integer l, there exists an irreducible polynomial of degree l over $\mathbb{F}_p[t]$. Furthermore such a polynomial can be found deterministically in time* $\text{poly}(l, p)$ *and probabilistically in expected time* $\text{poly}(l, \log p)$.

Given the above we see that we can pre-compute a representation of a field in expected time poly $\log q$ and then perform all field operations deterministically in time poly $\log q$. In certain scenarios it may be useful to have irreducible polynomial explicitly. In $\mathbb{F}_2[t]$ an infinite sequence of such polynomials is known (cf. [117, Theorem 1.1.28]).

Theorem 2.9 *For every $l \geq 0$, the polynomial $x^{2\cdot 3^l} + x^{3^l} + 1$ is irreducible over $\mathbb{F}_2[x]$.*

Thus we can construct fields of size $2^{2\cdot 3^l}$ for every integer l totally explicitly. Thus if we were interested in a field of size at least $q = 2^m$, and m is not of the form $2 \cdot 3^l$, we can find an m' of the right form with $m' < 3m$ and the resulting field would be of size less than q^3, which is only polynomially larger than our lower bound.

2.7 Evaluations of polynomials

We introduced polynomials merely as formal sums — syntactic expressions with no semantics associated with them. Evaluations associate some semantics to them.

Definition 2.10 *The* evaluation *of a polynomial $f = \sum_{i=0}^{d} f_i t^i \in R[t]$ at the point $\alpha \in R$, denoted $f(\alpha)$, is given by $\sum_{i=0}^{d} f_i \alpha^i$. Evaluations of multivariate polynomials are defined analogously; the evaluation of $f \in R[x_1, \ldots, x_m]$ and $\alpha = \langle \alpha_1, \ldots, \alpha_m \rangle$ is denoted $f(\alpha)$ or $f(\alpha_1, \ldots, \alpha_m)$.*

Evaluations carry natural semantics, i.e., $f(\alpha) + g(\alpha) = (f + g)(\alpha)$ and $f(\alpha) \cdot g(\alpha) = (f \cdot g)(\alpha)$. We are interested in knowing how often a polynomial can evaluate to zero. To answer this question, we first introduce the notion of division of polynomials.

Proposition 2.11 (Division Algorithm) *Given polynomials $f, g \in F[t]$ for some field F, there exists a unique pair of polynomials q and r (for quotient and remainder) in $F[t]$ satisfying $\deg(r) < \deg(g)$ and $f = g \cdot q + r$. Further the polynomial q satisfies $\deg(q) = \deg(f) - \deg(g)$ if $\deg(f) \geq \deg(g)$.*

The name of the proposition above is due to the fact that the proposition is proved by simply performing long division in the usual manner. Applying the above proposition with $g = t - \alpha$ for some $\alpha \in F$, we get that $f = q \cdot (t - \alpha) + r$ where r has degree zero and hence $r \in F$. Furthermore evaluating the expression above at α yields $f(\alpha) = q(\alpha) \cdot (\alpha - \alpha) + r$, and thus $r = f(\alpha)$. Thus we have $f = q \cdot (t - \alpha) + f(\alpha)$ for some q of degree $\deg(f) - 1$. The following proposition then follows.

Proposition 2.12 *The polynomial $t - \alpha$ divides f if and only if $f(\alpha) = 0$.*

Thus we get that if distinct elements $\alpha_1 \ldots, \alpha_k$ are all zeroes of a polynomial f (i.e., $f(\alpha_i) = 0$ for $i \in [k]$) then the polynomial $h = \prod_{i=1}^{k} (t - \alpha_i)$ divides f. Since the degree of h is k it follows that the degree of f is at least k. So we get:

Theorem 2.13 *For a field F, an element $f \in F[t]$ evaluates to zero on at most $\deg(f)$ points in F.*

We now move onto estimating the number of zeroes of multivariate polynomials. To do so, we need a variant of Theorem 2.13 for multivariate polynomials. We obtain such a result by expanding the scope of the theorem above. We first need a definition.

Definition 2.14 *A commutative ring R is an* integral domain *if it does not contain any zero divisors, i.e., there do not exist non-zero elements $a, b \in R$ such that $a \cdot b = 0$.*

Note that the rings $F[x_1, \ldots, x_m]$ are integral domains. Integral domains are of interest in that they are almost as nice as fields. Specifically, the following construction gives a field that contains any given integral domain.

Definition 2.15 *For an integral domain R, its* field of fractions, *denoted $\tilde{R}$, is the ring whose elements are pairs (a, b) with $a \in R$ and $b \in R - \{0\}$, modulo the equivalence $(a, b) \cong (c, d)$ if $a \cdot d = b \cdot c$. The element (a, b) is interpreted as the ratio a/b. Addition and multiplication are defined analogously with $(a, b) + (c, d) = (a \cdot d + b \cdot c, b \cdot d)$ and $(a, b) \cdot (c, d) = (a \cdot c, b \cdot d)$.*

The following proposition is easily verified.

Proposition 2.16 *For every integral domain R, $\tilde{R}$ is a field. Further, R is contained in $\tilde{R}$.*

The field of fractions of $F[x_1, \ldots, x_m]$ is usually denoted $F(x_1, \ldots, x_m)$ and its elements are the rational functions (ratios of polynomials) in $x_1, \ldots, x_m$.

The following lemma is now an easy consequence of the notions of integral domains, fields of fractions, and Theorem 2.13.

Lemma 2.17 *The polynomial $g(\mathbf{x}) \in F[\mathbf{x}]$ is a zero of the polynomial $f(\mathbf{x}, t) \in F[\mathbf{x}][t]$ (i.e., $f(\mathbf{x}, g(\mathbf{x})) = 0$) if and only if $t - g(\mathbf{x})$ divides the polynomial $f(\mathbf{x}, t)$. Hence the polynomial f has at most* $\deg_t(f)$ *zeroes in $F[x_1, \ldots, x_m]$.*

Proof The proof is simple. We simply view f as a polynomial in $K[t]$, where $K = F(x_1, \ldots, x_m)$ is the field of fractions of $F[x_1, \ldots, x_m]$. Since $g \in K$, it follows that $f(g) = 0$ iff $t - g$ divides f (from Proposition 2.12). Furthermore, Theorem 2.13, applied to $f \in K[t]$, says that f has at most $\deg_t(f)$ roots in K, which contains $F[x_1, \ldots, x_m]$. ■

The following theorem is now an easy inductive consequence of Lemma 2.17.

Theorem 2.18 *A non-zero polynomial $f \in \mathbb{F}_q[x_1, \ldots, x_m]$ is non-zero on at least $\prod_{i=1}^{m}(q - \deg_{x_i}(f))$ points in $\mathbb{F}_q^m$.*

Proof We start by viewing $\mathbb{F}_q[x_1, \ldots, x_m]$ as a polynomial in $\mathbb{F}_q[x_1, \ldots, x_{m-1}][x_m]$. By Lemma 2.17, there are at most $\deg_{x_m}(f)$ choices of $\alpha_m \in \mathbb{F}_q[x_1, \ldots, x_{m-1}]$ and hence in $\mathbb{F}_q$ such that $f(x_1, \ldots, x_{m-1}, \alpha_m) = 0$. For an α_m such that $f(x_1, \ldots, x_{m-1}, \alpha_m) \neq 0$ (and there exist $q - \deg_{x_m}(f)$ such α_m's), let $f_{\alpha_m}(x_1, \ldots, x_{m-1}) = f(x_1, \ldots, x_{m-1}, \alpha_m)$. Since f_{α_m} is a polynomial in $F[x_1, \ldots, x_{m-1}]$ with $\deg_{x_i}(f_{\alpha_m}) \leq \deg_{x_i}(f)$, it follows (by induction on m) that f_{α_m} is non-zero on at least $\prod_{i=1}^{m-1}(q - \deg_{x_i}(f))$ points in $\mathbb{F}_q^{m-1}$. The theorem follows. ■

We can derive other variants of the theorem above. One such variant that is quite popular in the CS community, often termed Schwartz's lemma or the DeMillo-Lipton-Schwartz-Zippel lemma [28, 98, 125], is the following:

Theorem 2.19 *A non-zero polynomial $f \in \mathbb{F}_q[\mathbf{x}]$ of total degree d is zero on at most a d/q fraction of the points in $\mathbb{F}_q^m$.*

Remark: Both Theorems 2.18 and 2.19 can also be extended to count the number of zeroes in some space of the form S^m for $S \subseteq \mathbb{F}_q$, but we don't do so here.

Proof The proof again goes by induction, but this time in the reverse order. Let d_m be the degree of f in x_m and let $f_m \in F[x_1, \ldots, x_{m-1}]$ be the coefficient of x^{d_m} in f, where we view f as an element of $F[x_1, \ldots, x_{m-1}][x_m]$. Note that the total degree f_m is at most $d - d_m$. For a random choice of $\alpha_1, \ldots, \alpha_{m-1} \in \mathbb{F}_q$, by induction we have that the probability that $f_m(\alpha_1, \ldots, \alpha_{m-1}) = 0$ is at most $(d - d_m)/q$. If this event happens, then we give up (and assume $f(\alpha_1, \ldots, \alpha_{m-1}, \alpha_m) = 0$ for all α_m). Else we get a polynomial $g(x_m) \stackrel{\text{def}}{=} f(\alpha_1, \ldots, \alpha_{m-1}, x_m)$ in one variable of degree d_m. By Theorem 2.13 a random choice of α_m is a zero of g with probability at most d_m/q. For $f(\alpha_1, \ldots, \alpha_m)$ to be zero, it must be the case that $f_m(\alpha_1, \ldots, \alpha_{m-1}) = 0$ or $g(\alpha_m) = 0$, Thus by the union bound, we find that $f(\alpha_1, \ldots, \alpha_m) = 0$ with probability at most $(d - d_m)/q + d_m/q = d/q$. ■

Finally we describe yet another variant that was the version used in the classical Reed-Muller codes.

Theorem 2.20 *If $f \in F_q[x_1, \ldots, x_m]$ has individual degree at most l in each variable and has total degree $d = lk + r$, then it is non-zero on at least $(1 - l/q)^k(1 - r/q)$ fraction of the inputs from $\mathbb{F}_q^m$.*

The proof of the theorem is a simple variant of the two proofs above and so we won't repeat it. It is more interesting to see a consequence. Suppose we have a multilinear polynomial of total degree k over $\mathbb{F}_2$. Then it is non-zero on at least a 2^{-k} fraction of the domain $\mathbb{F}_2^m$. This is exactly the kind of result that was used in the original Reed-Muller codes.

2.8 Vector spaces over fields

Here we relay some basic definitions and results about linear algebra that form the basis of linear codes.

We will be considering subspaces of the vector space $\mathbb{F}_q^n$, which is endowed with an addition operator "+" and a scalar-vector product "·", where if $\mathbf{x} = \langle x_1, \ldots, x_n \rangle$ and $\mathbf{y} = \langle y_1, \ldots, y_n \rangle$, then $\mathbf{x} + \mathbf{y} = \langle x_1 + y_1, \ldots, x_n + y_n \rangle$ and for $\alpha \in \mathbb{F}_q$, $\alpha \cdot \mathbf{x} = \langle \alpha \cdot x_1, \ldots, \alpha \cdot x_n \rangle$.

Definition 2.21 (Linear subspace) *A subset $L \subseteq \mathbb{F}_q^n$ is a linear subspace of $\mathbb{F}_q^n$ if for every $\mathbf{x}, \mathbf{y} \in L$ and every $\alpha \in \mathbb{F}_q$ it is the case that $\mathbf{x} + \mathbf{y} \in L$ and $\alpha \cdot \mathbf{x} \in L$.*

Definition 2.22 (Basis, Dimension) *The* span *of vectors $\mathbf{x}_1, \ldots, \mathbf{x}_k$, denoted* $\mathrm{span}(\mathbf{x}_1, \ldots, \mathbf{x}_k)$, *is the set $\{\sum_{i=1}^k \alpha_i \cdot \mathbf{x}_i \mid \alpha_1, \ldots, \alpha_k \in \mathbb{F}_q\}$. A set of vectors $\mathbf{x}_1, \ldots, \mathbf{x}_k$ is* linearly independent *if $\sum_{i=1}^k \alpha_i \cdot \mathbf{x}_i = \mathbf{0}$ implies $\alpha_1 = \cdots = \alpha_k = 0$. For a linear subspace $L \subseteq \mathbb{F}_q^n$, a set of vectors $\mathbf{x}_1, \ldots, \mathbf{x}_k \in L$ forms a* basis *if the vectors are linearly independent and their span equals L. The* dimension *of L, denoted* $\dim(L)$, *is the size of the largest basis for L.*

Playing around with the definitions, one can show easily that the span of a set of vectors is a linear subspace, that every linear subspace has a basis, and that all bases for a given subspace have the same size. One way to describe a linear subspace is to give its basis. A different way is to give constraints satisfied by elements of the subspace. We move towards this notion next. To do so, we need the notion of an dot product of vectors. For $\mathbf{x} = \langle x_1, \ldots, x_n \rangle$ and $\mathbf{y} = \langle y_1, \ldots, y_n \rangle$, the dot product of $\mathbf{x}$ and $\mathbf{y}$, denoted $\langle \mathbf{x}, \mathbf{y} \rangle$, equals $\sum_{i=1}^{n} x_i y_i$.

Definition 2.23 (Null Space) *For a linear subspace $L \subseteq \mathbb{F}_q^n$, it null space, denoted $L^\perp$, is the set $\{\mathbf{y} \in \mathbb{F}_q^n \mid \langle \mathbf{x}, \mathbf{y} \rangle = 0\}$.*

Proposition 2.24 *The null space of a linear subspace $L \subseteq \mathbb{F}_q^n$ is also a linear subspace of $\mathbb{F}_q^n$ and has dimension $n - \dim(L)$.*

A full proof of this assertion turns out to be somewhat complicated, and seems to involve proving the well-known but non-trivial fact that the row rank of a matrix equals its column rank. Instead of proving this, we will give a sense of how the proof goes, by essentially giving an algorithm to compute the basis of the null space $L^\perp$, given a basis of the space L. The description in the following paragraph is not self-contained — reading this paragraph is not suited for all audiences. A better approach may be to read a chapter on linear algebra from a text on algebra (e.g., [73]).

Suppose $\mathbf{x}_1, \ldots, \mathbf{x}_k$ form a basis for L. Let $\mathbf{G} \in \mathbb{F}_q^{k \times n}$ be the matrix whose ith row is $\mathbf{x}_i$. Since $\mathbf{x}_1, \ldots, \mathbf{x}_k$ are linearly independent, it follows that the rank of $\mathbf{G}$ is k. (Note that we didn't really define the rank of a matrix — there will be other such transgressions in this paragraph.) In particular this means there is an invertible $k \times k$ submatrix in $\mathbf{G}$. By permuting the columns of $\mathbf{G}$, we can write it as $\mathbf{G} = [\mathbf{A}|\mathbf{B}]$ where $\mathbf{A}$ is an invertible square matrix and $\mathbf{B}$ is the rest. Note that we are interested in a basis of the space $L^\perp = \{\mathbf{y} \mid \mathbf{y}\mathbf{G} = \mathbf{0}\}$. Writing all vectors $\mathbf{y} \in \mathbb{F}_q^n$ as $\mathbf{y} = (\mathbf{y}_A, \mathbf{y}_B)$ where $\mathbf{y}_A \in \mathbb{F}_q^k$ and $\mathbf{y}_B \in \mathbb{F}_q^{n-k}$, we get $\mathbf{y}_A\mathbf{A} + \mathbf{y}_B\mathbf{B} = \mathbf{0}$ for all $\mathbf{y} \in L^\perp$. This essentially yields $L^\perp = \{(-\mathbf{y}_B\mathbf{B}\mathbf{A}^{-1}, \mathbf{y}_B) | \mathbf{y}_B \in \mathbb{F}_q^{n-k}\}$, Taking $\mathbf{y}_B$ to be all the unit vectors gives $n - k$ vectors that generate $L^\perp$.

The correct way to think of the null space $L^\perp$ is that its members give linear constraints on the members of L. E.g., the vector $\mathbf{y} \in L^\perp$ enforces the constraint $\sum_{i=1}^{n} y_i x_i = 0$ on the vectors $\mathbf{x}$ in L. Since it suffices to satisfy the constraints given by any basis of $L^\perp$ (the other constraints get satisfied automatically), the basis of $L^\perp$ gives an alternate succinct representation of L.

We now move on to computational versions of the above results: Most of these computational results just build on the essential fact that Gaussian elimination works over any field (and not just rationals or reals or complexes).

Theorem 2.25 *Given a matrix $\mathbf{G}$ whose rows are the vectors $\mathbf{x}_1, \ldots, \mathbf{x}_k \in \mathbb{F}_q^n$, let $L = \mathrm{span}(\mathbf{x}_1, \ldots, \mathbf{x}_k)$. Then, the following problems can be solved in time $O((n+k)^3)$:*

1. *For $n = k$, compute the determinant of $\mathbf{G}$.*
2. *Compute the rank of $\mathbf{G}$.*
3. *If $\operatorname{rank} \mathbf{G} = k$, then a pseudo-inverse matrix $\mathbf{G}^{-1}$ such that $\mathbf{G}\mathbf{G}^{-1} = \mathbf{I}_k$, where $\mathbf{I}_k$ is the $k \times k$ identity matrix.*
4. *Given a vector $\mathbf{b} \in \mathbb{F}_q^n$, compute a vector $\mathbf{x}$ and a matrix $\mathbf{H}$ such that the set $\{\mathbf{z}_\mathbf{y} = \mathbf{x} + \mathbf{y}\mathbf{H} \mid \mathbf{y}\}$ is the set of solutions to $\mathbf{z}\mathbf{G} = \mathbf{b}$ if such a solution exists.*

2.9 Representing fields by matrices

We have already encountered one representation for element of a field $\mathbb{F}_q$, where $q = p^l$ for some prime (power) p — namely, the elements of $\mathbb{F}_q$ are polynomials over $\mathbb{F}_p$. However it is often to know about other representations. Here we describe two (actually one and a half) representations.

The first representation is only semi-adequate in that it describes how to do addition in $\mathbb{F}_q$, but not how to multiply. But it is useful to get to the second representation. Further it is often useful to think of the two simultaneously. We now start with the simple representation.

Fields as vector spaces. A simple way to think of $\mathbb{F}_{p^l}$ is as $\mathbb{F}_p^l$ - i.e., field elements are just vectors over $\mathbb{F}_p$ and field addition is just vector addition. Formally, there is an invertible transformation $L : \mathbb{F}_{p^l} \to \mathbb{F}_p^l$, that $\alpha \in \mathbb{F}_{p^l}$ is represented by the element $L(\alpha) \in \mathbb{F}_p^l$. such that $L(\alpha+\beta) = L(\alpha) + L(\beta)$. However this representation does not give a clue on how to do field multiplication.

Field elements as linear transformations. One way to think of a field element is that the element α defines a map $\beta \mapsto \alpha \cdot \beta$. Now if we represent β by its linear representation, then we get that α is a *linear* map from $\mathbb{F}_p^l$ to $\mathbb{F}_p^l$. In other words, if we fix the linear representation L, then corresponding to α, we can define a map $M_\alpha : \mathbb{F}_p^l \to \mathbb{F}_p^l$, with $M_\alpha(L(\beta)) = L(\alpha \cdot \beta)$. Note that this map satisfies $M_\alpha(L(\beta) + L(\gamma)) = M_\alpha(L(\beta)) + M_\alpha(L(\gamma))$. Since this is a linear map, this says there is a matrix $\mathbf{M}_\alpha \in \mathbb{F}_p^{k\times k}$ such that $M_\alpha(\mathbf{x}) = \mathbf{x}\mathbf{M}_\alpha$. Furthermore, in this case we have $\mathbf{M}_{\alpha_1\cdot\alpha_2} = \mathbf{M}_{\alpha_1} \cdot \mathbf{M}_{\alpha_2}$. and $\mathbf{M}_{\alpha_1+\alpha_2} = \mathbf{M}_{\alpha_1} + \mathbf{M}_{\alpha_2}$. Thus the transformation $\alpha \mapsto \mathbf{M}_\alpha$ maps $\mathbb{F}_{p^l}$ to $\mathbb{F}_p^{k\times k}$ and has the property that addition and multiplication in the field are just addition and multiplication of matrices! This representation (of α by $\mathbf{M}_\alpha$) can be quite useful at times.

2.10 Conclusions

Not all the descriptions above were intended to be complete. The idea is to list facts that (a) are true and (b) are assumed to be true in this course. Hopefully we won't use stuff that is not in these notes — but if we do we will try to make explicit note of this later. If you are planning to learn from this class you should either (a) be completely comfortable with assuming the facts stated, or (b) read appropriate algebra texts to review the material on fields and linear algebra. Some recommendations (this may be expanded later) include the text by Lidl and Neidereitter on finite fields [68] and by MacLane and Birkhoff on algebra in general [73].

Chapter 3

6.897 Algorithmic Introduction to Coding Theory	September 12, 2001
Lecture 3	
Lecturer: Madhu Sudan	*Scribe: Arian Shahdadi*

Today we will talk about Hamming codes and the Hamming bound. In order to talk about Hamming's codes, we will define linear codes. The lecture also contained material on some basic and linear algebra, which is deferred to a separate handout.

3.1 Linear Codes

There are many ways to specifiy a code given our established formalisms. One convenient way is to give an encoding function, thereby specifying how to create arbitrary codewords. This generalized mapping can be used to decribe classes of codes in a succinct manner. For a special class of codes called *linear codes*, other succinct options are available. We will define these codes here.

Linear codes are obtained when the alphabet Σ is associated with the field $\mathbb{F}_q$ for some finite q. In such cases, one can think of the "ambient space" (Σ^n), which contains the codewords and received words, as a vector space.

Recall that $L \subseteq \mathbb{F}_q^n$ is a *linear subspace* of the vector space $\mathbb{F}_q^n$ if for every pair $\mathbf{x}, \mathbf{y} \in L$ and $\alpha \in \mathbb{F}_q$, it is the case that $\mathbf{x}+\mathbf{y} \in L$ and $\alpha \cdot \mathbf{x} \in L$. (Here $+$ is vector addition and $\cdot$ is scalar-vector multiplication. Specifically, if $\mathbf{x} = \langle x_1, \ldots, x_n \rangle$ and $\mathbf{y} = \langle y_1, \ldots, y_n \rangle$, then $\mathbf{x} + \mathbf{y} = \langle x_1 + y_1, \ldots, x_n + y_n \rangle$ and $\alpha \cdot \mathbf{x} = \langle \alpha \cdot x_1, \ldots, \alpha \cdot x_n \rangle$.)

When a code $C \subseteq \mathbb{F}_q^n$ is a linear subspace of $\mathbb{F}_q^n$, then the code is called a *linear code*. Notationally, we express the fact that an $(n, k, d)_q$ code is linear by using square brackets and denoting it an $[n, k, d]_q$ code.

3.1.1 Generator matrix and Parity check matrix

Linear algebra gives us several succinct ways of representing codes. Recall that a linear subspace $C \subseteq \mathbb{F}_q^n$ of dimension k can be specified by giving a basis for the subspace, i.e., by giving an independent set of vectors $\mathbf{x}_1, \ldots, \mathbf{x}_k \subseteq \mathbb{F}_q^n$ such that $C = \{\sum_{i=1}^k \alpha_i \cdot \mathbf{x}_i \mid \alpha_1, \ldots, \alpha_k \in \mathbb{F}_q\}$. Note in particular that the code has q^k vectors and so we are not abusing notation here by using k to mean two different things — the dimension does equal the message length.

Using matrix notation, we can thus represent the code C by a *generator matrix* $\mathbf{G} \in \mathbb{F}_q^{k \times n}$, where the rows of $\mathbf{G}$ are the basis vectors $\mathbf{x}_1, \ldots, \mathbf{x}_k$. (Throughout this course we will denote vectors by row vectors, unless otherwise specified.) In this notation, $C = \{\alpha \mathbf{G} \mid \alpha \in \mathbb{F}_q^k\}$.

Another way of describing a linear subspace is by enumerating the constraints that vectors in the subspace obey, or equivalently, by describing its "null space". For $\mathbf{x}, \mathbf{y} \in \mathbb{F}_q^n$, let $\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{i=1}^n x_i \cdot y_i$ where the multiplications and summation denote field operations. Recall that for every linear subspace C of dimension k, one can find a linear subspace, denoted $C^\perp$, of dimension $n - k$, such that for every $\mathbf{x} \in C$ and $\mathbf{y} \in C^\perp$, $\langle \mathbf{x}, \mathbf{y} \rangle = 0$. Let $\mathbf{H}^T \in \mathbb{F}_q^{(n-k) \times n}$ be the generator matrix for $C^\perp$. The matrix $\mathbf{H} \in \mathbb{F}_q^{n \times (n-k)}$ is called the *parity check matrix* of C. Note that one can express $C = \{\mathbf{y} \mid \mathbf{yH} = \mathbf{0}\}$.

Note that under both representations, we can show trivially that the 0 word is a codeword for all linear codes. We will return to this fact later on.

Both the generator matrix and the parity check matrix representations give an $O(n^2 \log q)$ sized representation of the code C. These representations take $\text{poly}(n \log q)$ bits to describe and are thus fairly compact.

Furthermore these representations are equivalent computationally. With some rudimentary linear algebra, we can show that parity check matrices and generator matricies are essentially the same. That is, given G we can compute H and given H we can compute G.

Although we have shown that generator and parity check matrices are equivalent in both a mathematical and computational sense, do they work in the same manner? As we will see, parity check matrices have some properties that are very useful.

3.1.2 Applications of the parity check matrix

Error-detection: Suppose we are given a received vector $\mathbf{y}$ obtained from the transmission of a codeword from a code of minimum distance d. Further, suppose that we are guaranteed that that $d - 1$ or fewer errors have occurred. As we noted earlier, it is possible to tell, given unlimited computational resources, if an error has occured or not. Can we do this computation efficiently? It turns out that the answer, for linear codes C given by their parity check matrix $\mathbf{H}$, is "YES"! We simply compute the vector $\mathbf{yH}$. If the result is the all zeroes vector, then $\mathbf{y}$ is a codeword and hence no errors occurred during transmission. If $\mathbf{yH} \neq \mathbf{0}$, then $\mathbf{y}$ is not a codeword, by the definition of the parity check matrix. Thus we conclude that error-detection is "easy" (in polynomial time) for linear codes.

Another nice property of parity check matricies is that they give a sense of the minimum distance of a code. One important question is that of how we can determine the minimum distance of a linear code. Unfortunately, there are no known efficient methods of computing this — and the problem is known to be NP hard [118, 29]. Nevertheless parity check matricies offer some aid in designing codes with reasonable minimum distance as we will see shortly. First, we digress in order to define a new concept, the *Hamming weight* of a codeword.

Definition 3.1 *The* Hamming weight *of a vector* $\mathbf{x} \in \mathbb{F}_q^n$*, denoted* $\text{wt}(\mathbf{x})$*, is the number of non-zero* *coordinates in* $\mathbf{x}$*.*

Now, in order to find the minimum distance of a linear code, we can find a non-zero codeword $\mathbf{x}$ of smallest weight such that $\mathbf{x} \cdot \mathbf{H} = \mathbf{0}$. The weight of this codeword, $\mathrm{wt}(\mathbf{x})$, equals the minimum distance of the code. We can prove this assertion using elementary linear algebra.

Proposition 3.2 *The minimum distance of a linear code* C *with parity check* $\mathbf{H}$ *equals the smallest integer* d *such that there exist* d *rows of* $\mathbf{H}$ *that are linearly dependent.*

Proof First, note that the proposition is equivalent to the assertion that the minimum distance of C equals the weight of the non-zero vector $\mathbf{x}$ of smallest weight that satisfies $\mathbf{xH} = \mathbf{0}$. To see this, let $\mathbf{h}_i$ denote the ith row of $\mathbf{H}$. Now suppose $\mathbf{xH} = \mathbf{0}$ and let $x_{i_1}, \ldots, x_{i_d}$ be the non-zero elements of $\mathbf{x}$. Then $\sum_{j=1}^{d} x_{i_j} \mathbf{h}_{i_j} = \mathbf{0}$ and thus the rows $\mathbf{h}_{i_1}, \ldots, \mathbf{h}_{i_d}$ are linearly dependent. Similarly any collection of d linearly dependent rows lead to a vector $\mathbf{x}$ of weight d such that $\mathbf{xH} = \mathbf{0}$.

Now we turn to showing that $\min_{\{\mathbf{x} \neq \mathbf{0} | \mathbf{xH} = \mathbf{0}\}} \{\mathrm{wt}(\mathbf{x})\}$ is the minimum distance of the code. First note that if $\mathbf{xH} = \mathbf{0}$, then its weight is an upper bound on the minimum distance since its distance from $\mathbf{0}$ (which is also a codeword) is $\mathrm{wt}(\mathbf{x})$. On the other hand this quantity also lower bounds the minimum distance: If $\mathbf{y}$ and $\mathbf{z}$ are the nearest pair of codewords, then the vector $\mathbf{x} = \mathbf{y} - \mathbf{z}$ is a codeword whose Hamming weight equals the Hamming distance between $\mathbf{y}$ and $\mathbf{z}$. ■

3.1.3 Systematic Codes

Finally we introduce another side-effect of linearity, which is the notion of a *systematic code.*

Definition 3.3 *An* $(n, k, d)_q$ systematic code *is a code in which the encoding of a message consists of* k *message symbols followed by* $n - k$ *check symbols.*

In other words, in a systematic encoding, the message is recoverable without decoding if there are no errors. Systematic codes are very common in practical implementations of encoding and decoding schemes.

The term "systematic" is from the paper of Hamming [48]. Hamming observed that codes obtained from "linear" operations can be turned into systematic ones without loss of generality. (Hamming did not explicitly introduce linear codes, though linearity did play a dominant role in his codes.) To do so, note that if the generator matrix $\mathbf{G}$ is of the form $[\mathbf{I} \mid \mathbf{A}]$ where $\mathbf{I}$ is the $k \times k$ identity matrix, and $\mathbf{A}$ is a $k \times (n-k)$ matrix, then the code generated by $\mathbf{G}$ is systematic. So what can we do if $\mathbf{G}$ is not of this form? For simplicity, we describe the procedure when $q = 2$ and the code is binary. (The method also generalizes to other fields.) First we permute the columns and rows of $\mathbf{G}$ till the leftmost entry of the first row is a 1. (Permuting the rows does not alter the code, while permuting the columns only changes the naming of the coordinate indices.) Now we can arrange it so that all the leftmost entry of all remaining rows is 0. We do so by subtracting the first row from any given row if the row has a 1 in the leftmost entry. Again this process does not change the code — it only changes the basis we are using for the code. Repeating this process for the remaining indices, we arrive at a generator matrix with an identity matrix in the left part - as desired.

We conclude that every linear code is systematic. The converse need not be true.

3.2 Hamming Codes

The machinery of linear codes developed above allows us to arrive at the Hamming codes with minimum distance $d = 3$ naturally. In order to describe these codes, we construct their parity check matrix $\mathbf{H}$.

For simplicity, we will construct the matrix over the alphabet $\Sigma = \{0, 1\}$. This puts us in the field $\mathbb{F}_2$, where the addition and multiplication operations are just arithmetic modulo 2.

Notice our goal is to construct a matrix $\mathbf{H} \in \mathbb{F}_q^{n\times(n-k)}$ such that $\mathbf{xH} \neq \mathbf{0}$ for any vector $\mathbf{x}$ of weight 1 or 2. (Then the code will have distance ≥ 3. We will do so for as large an n as possible, given a fixed choice of $l = n - k$. Denote by $\mathbf{h}_i$ the ith row of this matrix. We will first determine what conditions $\mathbf{h}_i$ must satisfy.

First, consider a vector $\mathbf{x}$ of weight 1. Say this vector has a 1 in the ith position and is zero elsewhere. Then $\mathbf{xH} = \mathbf{h}_i$. Since we don't want $\mathbf{xH}$ to be zero, this imposes the constraint that $\mathbf{h}_i$ should not be zero. If $\mathbf{H}$ satisfies this property, then it gives the parity check matrix of a code with distance ≥ 2.

Now, consider a vector $\mathbf{x}$ of weight 2. Say, this vector has a 1 in the ith and jth coordinates and is zero elsewhere. Here, we get, $\mathbf{xH} = \mathbf{h}_i + \mathbf{h}_j$. If this result is to be non-zero, we need $\mathbf{h}_i \neq \mathbf{h}_j$, for every $i \neq j$. We conclude that $\mathbf{H}$ is the parity check matrix of a code of minimum distance ≥ 3 over $\mathbb{F}_2$ if and only if all its rows are distinct and non-zero.

Thus if $l = n - k$, then the largest n we can allow is the number of distinct non-zero vectors $\mathbf{h}_i \in \mathbb{F}_2^l$. This is a simple enumeration problem — there are exactly $2^l - 1$ such vectors and so we get $n = 2^l - 1$. Using our previously defined notation, we can now fully specify the parameters of the generalized Hamming codes:

Proposition 3.4 *For every positive integer l, there exists an $[2^l - 1, 2^l - l - 1, 3]_2$ code, called the* Hamming code.

Exercises:

- Describe the parameters and parity check matrix of the q-ary Hamming codes.
- Describe the generator matrix of the Hamming code. Put it in systematic form $[\mathbf{I} \mid \mathbf{A}]$ and describe the entries of the matrix $\mathbf{A}$.

To make some sense of the growth of parameters, notice that if $k = 2^l - l - 1$, then $l = \Theta(\log k)$. Thus the Hamming codes are obtained by appending $\Theta(\log k)$ parity check bits to the k-bit message. This gives a code of minimum distance 3, or equivalently, a 1-error-correcting code. The next section addresses the issue of whether one needs to expend so much redundancy to correct one error. But first we describe a simple error-correcting algorithm for the Hamming code.

3.2.1 Error-correction in the Hamming code

Given a received vector $\mathbf{y}$, where $\mathbf{y} = \mathbf{x} + \mathbf{e}$ for some codeword $\mathbf{x}$ and a vector $\mathbf{e}$ of weight one, how can we compute the location i such that $e_i = 1$?

The naive method for computing this would be by brute force. For $j = 1$ to n, we try flipping the jth bit of $\mathbf{y}$ and see if this gives us a codeword. We can check if such a vector $\mathbf{y}'$ is a codeword by computing $\mathbf{y}'\mathbf{H}$ and checking to see if it is zero. This gives an $O(n^3)$ algorithm.

Hamming wasn't satisfied with this approach and instead relied on the parity check matrix to simplify this computation. Since we know that $\mathbf{xH} = \mathbf{0}$, we have that $\mathbf{yH} = \mathbf{eH} = \mathbf{h}_i$, where i is the index of the error location. Furthermore, if we arrange $\mathbf{H}$ so that the ith row is simply the number i written in binary, then the vector $\mathbf{yH}$ simply gives the location of the error in binary. This speeds up our computation by a factor of n, giving an $O(n^2)$ algorithm to correct errors.

3.3 The Hamming bound and perfect codes

Is it necessary to add logarithmically many bits to correct 1 error? Hamming proved that this was the optimal result by introducing what is called the Hamming bound. We now derive this bound.

Let us recall some notation we have used earlier. Let $B(\mathbf{y}, r)$ denote the ball of radius r around vector $\mathbf{y}$ (where for this section the vectors will be from $\mathbb{F}_2^n$). Let $\mathrm{Vol}(r, n) = \sum_{i=0}^{r} \binom{n}{i}$ be the volume of such a ball. The Hamming bound gives the following:

Theorem 3.5 *If an $(n, k, d)_2$ code exists, then*

$$2^k \mathrm{Vol}\left(\left\lfloor \frac{d-1}{2} \right\rfloor, n\right) \leq 2^n.$$

Proof Let $C = (n, k, d)_2$ code and let $r = \lfloor \frac{d-1}{2} \rfloor$. The Hamming bound is based on the simple observation that if $\mathbf{x}$ and $\mathbf{y}$ are codewords of C, then $B(\mathbf{x}, r)$ and $B(\mathbf{y}, r)$ are disjoint. Since $\bigcup_{\mathbf{x} \in C} B(\mathbf{x}, r) \subseteq \{0,1\}^n$ and the sets on the LHS are all disjoint, we get: $\sum_{\mathbf{x} \in C} |B(\mathbf{x}, r)| \leq 2^n$. The bound follows immediately. ■

The Hamming bound generalizes naturally to q-ary alphabets. Let $\mathrm{Vol}_q(r, n) = \sum_{i=0}^{r} \binom{n}{i} (q-1)^i$. Then we have:

Theorem 3.6 *For every n, k, d and q, an $(n, k, d)_q$ code exists only if*

$$q^k \mathrm{Vol}_q\left(\left\lfloor \frac{d-1}{2} \right\rfloor, n\right) \leq q^n.$$

For the case $d = 3$ and $q = 2$, the above bound says that $2^k(n+1) \leq 2^n$, and this bound is met *exactly* by the Hamming codes (and this is true also for q-ary Hamming codes). The Hamming bound defines a class of codes known as *perfect* codes.

Definition 3.7 *An $(n, k, d)_q$ code is* perfect *if it meets the Hamming bound exactly: i.e.,*

$$q^k \mathrm{Vol}_q\left(\left\lfloor \frac{d-1}{2} \right\rfloor, n\right) = q^n.$$

An interesting fact here is that other than the Hamming codes and two specific codes proposed by Golay [37] no other perfect codes exist. This was supsected for long and finally proved by van Lint [116] and Tietavainen [113]. This proof is way beyond the scope of this class.

Converting odd minimum distance to even minimum distance

The Hamming bound is clearly "jerky" - it proves nothing interesting for $d = 2$, but jumps to proving that the Hamming codes for $d = 3$ are best possible. It does not improve for $d = 4$ but then jumps again when $d = 5$. Why does this discontinuity happen? Is it that the Hamming bound is too weak, or is it truly easier to construct codes of even minimum distance? Hamming showed, surprisingly enough, that the latter was the case. The argument is quite simple - the same that we used to construct distance 2 codes from distance 1 codes and we describe it next.

Proposition 3.8 *If an $(n, k, 2t + 1)_2$ code exists, then so does an $(n + 1, k, 2t + 2)_2$ code.*

Proof Let C be an $(n, k, 2t+1)_2$ code. We construct C', an $(n+1, k, 2t+2)_2$ code from C as follows: Let $\mathbf{x} = \langle x_1, \ldots, x_n \rangle \in C$. Corresponding to $\mathbf{x}$, C' contains the codeword $\mathbf{x}' = \langle x_1, \ldots, x_n, \sum_{i=1}^n x_i \rangle$. (All summations in this proof are sums modulo 2.) It is clear that C' is an $(n + 1, k, d)_2$ code for some d, and what we wish to show is that $d \geq 2t + 2$.

It suffices to show that if $\mathbf{x}, \mathbf{y} \in C$, then $\Delta(\mathbf{x}', \mathbf{y}') \geq 2t + 2$. Notice first that $\Delta(\mathbf{x}', \mathbf{y}') \geq \Delta(\mathbf{x}, \mathbf{y})$. So if the latter is at least $2t + 2$ then we are done. Thus it suffices to consider the case where $\Delta(\mathbf{x}, \mathbf{y}) = 2t + 1$. In this case, let us permute the coordinates of C so that $\mathbf{x}$ and $\mathbf{y}$ agree in the first $m = n - (2t + 1)$ locations and disagree in the last $2t + 1$ locations. Now consider the quantity $\sum_{i=1}^n (x_i + y_i)$. This quantity equals the parity of "($2*$ # 1's in $\mathbf{x}$ in the first m coordinates) $+(2t + 1)$". Clearly, this parity is odd (since the first expression as well as $2t$ are even and the last term, 1 is odd). We thus get that exactly one of $\sum_{i=1}^n x_i$ and $\sum_{i=1}^n y_i$ is one (modulo 2) and the other is zero. Thus $\mathbf{x}'$ and $\mathbf{y}'$ disagree in the last coordinate and so the distance between them is $2t + 2$ in this case also. ■

Exercise: Does the proposition above extend for other values of q?

Applying the proposition above to the distance 3 codes, Hamming got a family of distance 4 codes. Going from $d = 4$ to $d = 5$ is non-trivial. In particular the Hamming bound implies that one needs approximately $2 \log k$ check bits to get such a code. Getting such a code proved to be non-trivial and was finally discovered, independently by Bose and Ray-Chaudhuri [23], and Hocquenghem [49], in 1959. These codes, which are quite famous under the label "BCH codes", give binary codes of block length n and odd minimum distance d using $\frac{d-1}{2} \log n$ check bits. The Hamming bound shows this is tight to within lesser order terms.

Chapter 4

6.897 Algorithmic Introduction to Coding Theory	September 19, 2001
Lecture 4	
Lecturer: Madhu Sudan	*Scribe: Joe Aung*

Today's topics:

- Singleton bound and Maximum Distance Separable (MDS) codes.
- Reed Solomon codes.
- Reed Muller codes.
- Hadamard codes.

4.1 The Singleton Bound

Our first result is a simple lower bound on the block length of a codeword, given a fixed distance and message length. This bound is due to R. C. Singleton [106] and is hence named the Singleton bound. To motivate this result, recall that in lecture 2, we saw an $[n,n,1]_2$ code and an $[n,n-1,2]_2$ and wondered if we could generalize these results to a $[n,n-d+1,d]_2$ code (and the Hamming bound ruled this out for $d=3$ in the binary case). A more elementary question is why should we only ask for $k \leq n-d+1$ and not better! The Singleton bound shows that this is indeed the best possible, over *any* alphabet.

Theorem 4.1 ([106]) *If C is an $(n,k,d)_q$ code then $d \leq n-k+1$.*

Proof Let Σ be the q-ary alphabet of C. Consider the projection map $\pi : \Sigma^n \to \Sigma^{k-1}$ that projects every word in Σ^n to its first $k-1$ coordinates. Since the range of π has only q^{k-1} elements and $|C| = q^k > q^{k-1}$, we see that there must exist two distinct codewords $\mathbf{x}, \mathbf{y} \in C$ such that $\pi(\mathbf{x}) = \pi(\mathbf{y})$. Since $\mathbf{x}$ and $\mathbf{y}$ agree on their first $k-1$ coordinates, it follows that they may differ on at most all remaining $n-(k-1)$ coordinates, and thus we have $\Delta(\mathbf{x},\mathbf{y}) \leq n-k+1$. It follows that the minimum distance of C is at most $n-k+1$. ■

Codes that meet the Singleton bound, i.e., satisfy $k = n - d + 1$, are called *Maximum Distance Separable (MDS)* codes. Last time we defined *Perfect Codes* as codes that meet the Hamming bound, and we said that the only perfect codes were the Hamming codes and two codes discovered by Golay. MDS codes and perfect codes are incomparable: i.e., there exist perfect codes that are not MDS and MDS codes that are not perfect. Each meets an incomparable optimality criterion. Today we will see a simple but large family of MDS codes, namely the Reed-Solomon codes.

4.2 Reed Solomon codes

Reed-Solomon codes were introduced in a paper by Reed and Solomon in 1959 [92]. They are based on properties of univariate polynomials and in particular the following property of univariate polynomials as introduced in the last lecture (see Lecture Notes on Algebra).

Fact 4.2 *Two distinct polynomials p_1, $p_2 \in F_q[x]$ of degree strictly less than k, agree in strictly less than k points in F_q. I.e., there exist at most $k-1$ points $\alpha \in F_q$ s.t. $p_1(alpha)$=$p_2(alpha)$.*

4.2.1 Construction of Reed-Solomon Codes

We describe the Reed-Solomon codes by giving the encoding function for them. Note that the encoding function is not unique. Our choice is made simply to ease the exposition.

Given a prime power q and $n \leq q$, and $k \leq n$, a Reed-Solomon code $\mathrm{RS}_{q,n,k}$ is constructed as follows:

1. Generate the field $\mathbb{F}_q$ explicitly (say via an irreducible polynomial over the underlying prime).
2. Pick n distinct elements $\alpha_1, \dots, \alpha_n \in F_q$. Note this is where we need the property $n \leq q$.
3. To define the encoding, we first pick a convenient representation for the messages. Note that the message is k elements of $\mathbb{F}_q$, say $c_0, \dots, c_{k-1}$. We let the message define the polynomial $C(x) \stackrel{\text{def}}{=} \sum_{j=0}^{k-1} c_j x^j$.
4. The encoding of the message $C(x)$ is its evaluation at $\alpha_1, \dots, \alpha_n$, i.e., the sequence $\langle C(\alpha_1), \dots, C(\alpha_n) \rangle$.

4.2.2 Parameters acheived by the code $\mathrm{RS}_{q,n,k}$

First we note the the Reed Solomon codes are linear.

Proposition 4.3 *The Reed-Solomon code $\mathrm{RS}_{q,n,k}$ is linear.*

Proof Suppose we are given two codewords $\langle C(\alpha_1), \dots, C(\alpha_n) \rangle$ and $\langle D(\alpha_1), \dots, D(\alpha_n) \rangle$ and suppose $\beta \in \mathbb{F}_q$. We need to show that the sequences $\langle C(\alpha_1) + D(\alpha_1), \dots, C(\alpha_n) + D(\alpha_n) \rangle$ and $\langle \beta C(\alpha_1), \dots, \beta C(\alpha_n) \rangle$ are also codewords. Note that the former sequence is the evaluations of the polynomial $(C+D)(x)$ at the points $\alpha_1, \dots, \alpha_n$, while the latter is the evaluations of the polynomial βC at the same points. Further, note that if C, D are polynomials of degree at most $k-1$ then the polynomials $C + D$ and βC are also polynomials of degree at most $k - 1$. Thus the resulting sequences are also codewords of the Reed-Solomon code. ■

It is obvious from the construction that the Reed-Solomon code $\mathrm{RS}_{q,n,k}$ has block length n and message length k. The only parameter that does not follow by definition is the distance, but that is easily argued.

Proposition 4.4 *The Reed-Solomon code* $\mathrm{RS}_{q,n,k}$ *has distance* $n - k + 1$.

Proof By the Singleton bound we know that the distance $d \leq n - k + 1$. So it suffices to prove $d \geq n - k + 1$. Suppose we have two distinct polynomials $C(x)$ and $D(x)$ of degree at most $k - 1$. Then by Fact 4.2 we have that $C(x)$ and $D(x)$ agree on at most $k-1$ points of $\mathbb{F}_q$ and hence disagree on at least $n - k + 1$ points of the set $\{\alpha_1, \ldots, \alpha_n\}$. The distance follows. ■

As a result we get the following theorem.

Theorem 4.5 *For every prime power* q*, and every pair of positive integers* k, n *such that* $k \leq n \leq q$*, there exists an* $[n, k, n - k + 1]_q$ *code.*

4.2.3 Applications of the Reed-Solomon codes

By playing games with the alphabet size, we've managed to construct codes that meet the Singleton bound. But a natural question to ask at this stage is: "How useful is it to have a code over large alphabets?".

To answer the question, we first invoke empirical evidence! Reed-Solomon codes are possibly the most commonly used codes in practical applications. In particular they are used to store information/music/video in compact discs (CDs) and digital video discs (DVDs), making them the most deployed family of codes! How do these technologies end up using codes over large alphabets? We describe the basic idea below. (Warning: The numbers used below are mostly for example. They are close to, but not exactly equal to the numbers used in practice.)

CDs and DVDs store information as a sequence of bits. The actual sequence of bits is quite long. The usual error-correcting methods break this long sequence into a collection of small chunks and encode each chunk separately. For example, we may pick each chunk to contain 240 bytes each (where one byte equals 8 bits). This gives a message sequence of 240 bytes where we now interpret each byte as an element of $\mathbb{F}_{256}$, the field on 256 elements. Using $n = q = 256$, one may encode this sequence using a Reed-Solomon code $\mathrm{RS}_{256,256,240}$ to get a sequence of 256 bytes which are then recorded on the CD. Thus in actuality we have described a *binary error-correcting code* of message length 240×8 bits, and block length 256×8 bits. What is the distance of this code? To analyze the distance, first let us recall that the underlying Reed-Solomon code over $\mathbb{F}_{256}$ has distance 17 — i.e., we must change at least 17 bytes of the encoded message to get an encoding of some other message. In turn this implies that we need to flip at least 17 bits in the binary encodings to get from one codeword to another. Abstracting this idea for arbitrary n, k and q, we get the following implication for binary codes.

Proposition 4.6 *For every* $k \leq n \leq q$*, where* q *is a prime power, there exists a family of* $(n\lceil \log_2 q \rceil, k \log_2 q, n - k + 1)_2$ *code.*

Exercise: Show that if $q = 2^l$, then the above code construction can be made linear.

How good is such a code? Written slightly differently, and throwing away some floors and ceilings, we see that the above amounts to codes of the form $(K + (1 + o(1))d \log K, K, d)_2$ codes. In contrast the Hamming bound says a code with message length K and distance d must have block length at least $K + (1 - o(1))\frac{d}{2} \log K$, for any fixed d and $K \to \infty$. So these codes based on Reed-Solomon codes are not too bad compared to the impossibility result. As we mentioned last time, better codes are known. In particular for the same block length and distance, BCH codes could encode 248 bytes of data. But analyzing those codes is somewhat harder, which is why we don't present them here.

Still the complexity of analyzing the distance of the code can not possibly a reason not to use them in practice. So do people prefer to use a weaker Reed-Solomon code as opposed to a potentially better BCH code? The main reason is the nature of the error. Typical errors on storage devices tend to happen in bursts. So when, say, 30 bits on the chunk are flipped it is quite likely that these 30 bit errors are not distributed uniformly over the chunk, but are localized to five or six bytes. In such as case, the Reed-Solomon code can actually correct all these errors (since it can correct upto 8 byte errors)! This enhanced performance of the Reed-Solomon codes in case of bursty error patterns is the main reason why it is so commonly used.

4.3 Codes based on Multivariate Polynomials

The major bottleneck with the Reed Solomon codes is the restriction $q \geq n$. In this section, we use minor algebraic extensions of such codes to get codes which work over smaller alphabets, including one non-trivial family of codes over the binary alphabet.

4.3.1 Bivariate polynomials

We start by generalizing the idea behind Reed Solomon codes in a simple way using bivariate polynomials instead of univariate polynomials. This will already give codes over an alphabet of size $q = \sqrt{n}$. We proceed as follows:

For prime power q and integer $l < q$ the bivariate polynomial code $B_{q,l}$ is defined as follows:

- Messages consist of $(l+1)^2$ field elements which we view as an $(l+1) \times (l+1)$ matrix of coefficients $\langle m_{ij} \rangle_{i=0,j=0}^{l,l}$. We identify this message with the bivariate polynomial $M(x, y) = \sum_{i=0}^{l} \sum_{j=0}^{l} m_{ij} x^i y^j$.

- The encoding of a message corresponding to $M(x, y)$ is its evaluation at all field elements. Thus the encoding of M is $\langle M(\alpha, \beta) \rangle_{\alpha \in \mathbb{F}_q, \beta \in \mathbb{F}_q}$.

This gives us an $[n, k, d]_q$ code with $n = q^2$ and $k = (l+1)^2$. How much is the distance? It follows from Theorem 17 of Lecture 2.5 (Lecture notes on algebra) that its distance is $d = (q - l)^2$. In contrast, the Singleton bound allows $d = q^2 - (l+1)^2$. The difference, approximately, $2l(q - l)$, is the price we pay for the smaller alphabet size.

4.3.2 Multivariate polynomial codes: Reed-Muller codes

We now extend the generalization of the previous section fully, to multivariate polynomials with an arbitrary number of variables, say m. These codes are termed Reed-Muller codes after their discoverers: These codes were discovered by D. E. Muller [83] and then I. S. Reed gave a decoding procedure for them [91]. The codes as described here are generalized to a range of parameters that were not covered originally, which seems to have focussed on codes over $\mathbb{F}_2$ only. (In particular, the way we describe them, these will be strict generalizations of Reed-Solomon codes, while Reed-Solomon codes were actually discovered much later!)

Here we will work with the notion of the total degree of a polynomial. Recall this is the maximum, over all monomials with non-zero coefficients, of the total degree of the monomial, where the total degree of a monomial is the sum of the degrees of all variables in it. E.g. the total degree of the monomial $x^3y^4z^3$ is 10, and the total degree of the polynomial $3x^9+4y^8+2x^3y^4z^3$ is also 10. Recall Theorem 18 from Lecture 2.5 shows that a polynomial of total degree l is zero on at most l/q fraction of the inputs — we will use this fact below.

We will start by presenting a computer scientist's view of Reed-Muller codes, which only consider polynomials of degree $l < q$.

Reed-Muller Codes - Case 1. For positive integers m, l and prime power q with $l < q$, the Reed-Muller code $\mathrm{RM}_{m,l,q}$ is defined as follows:

- The message is a sequence of coefficients $\langle m_{i_1,\dots,i_m}\rangle_{i_1+\cdots+i_m\leq l}$. The message represents the polynomial
$$M(x_1,\dots,x_m) = \sum_{i_1+\cdots+i_m\leq l} m_{i_1,\dots,i_m} x_1^{i_1}\dots x_m^{i_m}.$$
- The encoding of a polynomial $M(\mathbf{x})$ is the sequence $\langle M(\alpha)\rangle_{\alpha\in\mathbb{F}_q^m}$.

It is obvious that the block length of the code $\mathrm{RM}_{m,l,q}$ is $n = q^m$. The message length equals the number of m-long sequences of non-negative integers that sum to at most l, and this number turns out to be $\binom{m+l}{m}$. Finally, from Theorem 18 of Lecture 2.5, the distance of the code is at least (actually exactly) $(1-\frac{l}{q})n$. We will summarize the properties of the code shortly, but before doing so, let us consider a choice of parameters which is somewhat illustrative of the powers of this code.

Sample setting of parameters: Suppose we wish to encode k elements of some alphabet. It seems reasonable to ask for codes of length $n = \mathrm{poly}(k)$ that have large minimum distance (say $n/2$) with as small an alphabet as possible. It turns out Reed-Muller codes can give such codes with alphabet size $\mathrm{poly}(\log k)$, by the following setting of parameters: We choose $m = \frac{\log k}{\log\log k}$ and $q = \log^2 k$ and l such that $\binom{m+l}{l} = k$. For this choice of parameters, we note that the code $\mathrm{RM}_{m,l,q}$ has block length $n = q^m = k^2$ which was one of our goals. To estimate the distance, note the $\binom{m+l}{l} \geq (l/m)^m$. Thus we have $l \leq mk^{1/m} = m\log k = \log^2 k/\log\log k = o(q)$ as $k \to infty$. Thus this family of codes has distance $(1-o(1))\cdot n$.

Reed-Muller codes - Case 2. Now we consider the case where the total degree $l > q$. In such case, we associate messages with polynomials of total degree at most l and individual degree at most $q-1$ in every variable. Let $S(m,l,q) = \{(i_1,\dots,i_m) | \sum_j i_j \leq l, 0 \leq i_j < q\}$ and let $K(m,l,q) = |S(m,l,q)|$. The Reed-Muller codes $\mathrm{RM}_{m,l,q}$ are described as follows:

- The messages are a sequence of $K(m,l,q)$ elements of $\mathbb{F}_q$ denoted $\langle m_{\mathbf{i}} \rangle_{\mathbf{i} \in S(m,l,q)}$. This message is associated with the polynomial

$$M(x_1, \ldots, x_m) = \sum_{\mathbf{i} \in S(m,l,q)} m_{\mathbf{i}} x_1^{i_1} \cdots x_m^{i_m}.$$

- The encoding of the message is its evaluation at all points $\alpha \in \mathbb{F}_q^m$, i.e., the sequence $\langle M(\alpha) \rangle_{\alpha \in \mathbb{F}_q^m}$.

This yields a code of block length q^m and message length $K(m,l,q)$. To estimate the distance of the code write $l = a(q-1) + b$ with $b < q-1$. Then by Theorem 19 of the Lecture 2.5, the distance of this code is at least $q^{m-a}(1 - b/q)$. Again the setting of parameters may be somewhat confusing. So we give an example setting of parameters to illustrate the power of this code:

Sample setting of parameters: (This is the original setting of the parameters in the papers of Reed and Muller.) We let $q = 2$ and pick $l < m$. Then $K(m,l,2) = \mathrm{Vol}(l,m)$ (that's right! - the volume of the Hamming ball in $\{0,1\}^m$ of radius l.) We may lower bound this quantity by $\binom{m}{l}$. The distance of the code is 2^{m-l}. Thus the $\mathrm{RM}_{m,l,2}$ code gives a $[2^m, \binom{m}{l}, 2^{m-l}]_2$ code!

4.4 Hadamard codes

Before concluding today's lecture we give one more example of codes, which again turn out to be special cases of Reed-Muller codes. (The presentation here is different from the way we did it in lecture.)

Jacques Hadamard was interested in some constructions of self-orthogonal matrices with all entries being $+1$ or -1. The ensuing constructions lead to nice error-correcting codes and we describe this connection here.

Definition 4.7 *An $n \times n$ matrix $\mathbf{H} = \{\mathbf{h}_{ij}\}$ is a* Hadamard matrix *if $\mathbf{h}_{ij} \in \{+1, -1\}$ for all i, j and $\mathbf{H}\mathbf{H}^T = n\mathbf{I}$, where $\mathbf{I}$ is the $n \times n$ identity matrix, and all arithmetic is regular integer arithmetic.*

Viewed appropriately, the rows of an $n \times n$ Hadamard matrix give a binary code of block length n, with n codewords (i.e., a message length of $\log n$). To get this view, note that every row of H is just a binary vector (where the binary alphabet just happens to be $\{+1, -1\}$ rather than the usual $\{0,1\}$). Thus clearly the rows form a binary code (of message length $\log n$ and block length n). The most interesting aspect of this code is the distance. The fact that $\mathbf{H}\mathbf{H}^T = n\mathbf{I}$ is equivalent to saying that the codewords are at distance exactly $n/2$ from each other! To see this, note that the (i,j)th entry of $\mathbf{H}\mathbf{H}^T$ is $\sum_{k=1}^n \mathbf{h}_{ik}\mathbf{h}_{jk}$. For any k, the quantity $\mathbf{h}_{ik}\mathbf{h}_{jk}$ is either $+1$ or -1, with -1 indicating $h_{ik} \neq h_{jk}$ and $+1$ indicating $h_{ik} = h_{jk}$. For $i \neq j$, we have $\sum_{k=1}^n \mathbf{h}_{ik}\mathbf{h}_{jk} = 0$, and this implies that exactly $n/2$ of the summands are $+1$ and exactly -1 of the summands are $+1$. In turn this yields that for that half the coordinates k, $\mathbf{h}_{ik} = \mathbf{h}_{jk}$ and so the codewords corresponding to the ith and jth rows agree on exactly $n/2$ coordinates.

There are several ways to augment this obvious code and all of these are interchangeably referred to as Hadamard codes. For our purpose, we will fix the following codes to be Hadamard codes based on an $n \times n$ Hadamard matrix $\mathbf{H}$.

Definition 4.8 *Given an $n \times n$ Hadamard matrix* $\mathbf{H}$*, the Hadamard code of block length* n*,* Had_n*, is the binary code whose codewords are the rows of* $\mathbf{H}$ *(with $+1$s replaced by 0s and -1s replaced by 1s), and the complements of the rows of* $\mathbf{H}$*.*

Proposition 4.9 *For every n such that an $n \times n$ Hadamard matrix exists, the Hadamard code* Had_n *is a $(n, \log(2n), \frac{n}{2})_2$-code.*

Proof The block length and message length follow by definition. We have also argued that two codewords that correspond to distinct rows of the Hadamard matrix differ in $n/2$ places. Now if $\mathbf{c}$ is a codeword corresponding to a row of the matrix and $\mathbf{c}'$ is a codeword corresponding to the complement of a row of the matrix, then if the corresponding rows are the same, then the codewords differ everywhere; and if the corresponding rows are different then the codewords disagree whenever the corresponding rows agree, but this also happens exactly $n/2$ times. The proposition follows. ∎

We now point to one family of codes with the above parameters that we actually know of! Note that if we take the Reed-Muller code $\mathrm{RM}_{m,1,2}$ with m variables and total degree 1 over a binary alphabet, then we get a $[2^m, m+1, 2^{m-1}]_2$ code, which is of the form $[n, \log(2n), n/2]_2$. So it is worthwhile asking if this is a Hadamard code, i.e., is there an underlying Hadamard matrix.

It turns out that the $\mathrm{RM}_{m,1,2}$ codes do come from an underlying Hadamard matrix. To do so recall that the messages of the Reed-Muller code correspond to coefficients $c_0, \ldots, c_m$ representing the polynomial $C(\mathbf{x}) = c_0 + \sum_{i=1}^{m} c_i x_i$. Now if we consider only those codewords corresponding to $c_0 = 0$, then we get a collection of codewords that differ in exactly half the places, and using them as the rows yields the Hadamard matrix.

As an aside note that the usual construction of Hadamard matrices with n being a power of 2 is inductive, with $\mathbf{H}_m$, the $2^m \times 2^m$ Hadamard matrix being defined as:

$$\mathbf{H}_1 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad \text{and} \quad \mathbf{H}_{m+1} = \begin{bmatrix} \mathbf{H}_m & \mathbf{H}_m \\ \mathbf{H}_m & -\mathbf{H}_m \end{bmatrix}.$$

4.5 Summary

We've seen a number of different codes with incomparable merits. The Reed-Solomon codes have optimal distance to message length behaviour but require large alphabets. Hadamard codes work over a binary alphabet, but have very poor relationship between message length and block length. Hamming codes have a good relationship between message and block length, but only offer a distance of 3. In going forward we will look for families of codes which maintain a constant ratio between message length and block length, while also maintaining a constant ratio between distance and block length.

6.897 Algorithmic Introduction to Coding Theory	September 19, 2001
Lecture On BCH Codes	
Lecturer: Madhu Sudan	*Scribe: Madhu Sudan*

In Lecture 4, we spoke about BCH codes in vague terms. Here we give a self-contained account of the codes with full proofs of its parameters. Note that these may not be the most general possible BCH codes, but it shows the specific version that is interesting.

To remind the reader where we are going, recall our definition of a q-ary BCH code. In these notes, we will focus only on the case where the block length of the code equals q^m for some integer m.[1]

Definition 4.10 *For prime power q, integer m and integer d, the* BCH code $\mathrm{BCH}_{q,m,d}$ *is obtained as follows: Let $n = q^m$ and let $\mathbb{F}_{q^m}$ be an extension of $\mathbb{F}_q$ and let C' be the (extended) $[n, n-(d-1), d]_{q^m}$ Reed-Solomon code obtained by evaluating polynomials of degree at most $n-d$ over $\mathbb{F}_{q^m}$ at all the points of $\mathbb{F}_{q^m}$. Then the code $\mathrm{BCH}_{q,m,d}$ is the $\mathbb{F}_q$-subfield subcode of C'. (In other words, $\mathrm{BCH}_{q,m,d} = C' \cap \mathbb{F}_q^n$.)*

It is evident from the definition that the codes have distance *at least* d. (The code could have a larger distance, so the literature in coding theory refers to d as the *designed distance* of the code.) Our goal is to lower bound the dimension of this code as a function of n and d (and m and q). The reader may not realize immediately why this code even has positive dimension for small d. We will explain why this is so — and in doing so, we will lower bound the dimension by $n - m(d-1)$. The rest of the writeup will focus on a *mild* improvement by adding roughly $\frac{1}{q}m(d-1)$ to the dimension. The interest in this mild improvement is that in the binary case, this brings us close to the Hamming bound (for small d) as opposed to the Gilbert-Varshamov bound. So these shows that the BCH codes *beat* the Gilbert-Varshamov bound (as do every other family of codes we know).

Let us start with a simple argument showing this code has dimension at least $n - m(d-1)$. First recall that every function from $\mathbb{F}_{q^m}$ to $\mathbb{F}_q$ is a polynomial over $\mathbb{F}_{q^m}$ of degree at most $n-1$. Thus the space of polynomials from $\mathbb{F}_{q^m}$ to $\mathbb{F}_q$ is a $\mathbb{F}_q$-linear space of dimension exactly n. We wish to know what is the dimension of the subspace of polynomials of degree at most $n-d$. But now note that the restriction that a polynomial $f(x) = \sum_{i=0}^{n-1} f_i x^i$ has degree at most $n-d$ is equivalent to saying that the coefficients f_i must equal zero, for $i \in \{n-(d-1), \ldots, n-1\}$. Each such condition is a single linear constraint over $\mathbb{F}_{q^m}$, but this translates into a block of m linear constraints over $\mathbb{F}_q$. Since we have $d-1$ such blocks of constraints, the restriction that the functions have degree at most $n-d$ places at most $m(d-1)$ linear constraints. Thus the resulting space has dimension at least $n - m(d-1)$. The argument above is informal, but easy to formalize (and in fact some of the ingredients are even included later in these notes). Formalizing the argument leads to the following bound.

Proposition 4.11 (Weak dimension bound) *The code $\mathrm{BCH}_{q,m,d}$ has dimension at least $q^m - m(d-1)$.*

To improve the bound we will argue that not all the constraints above are independent. In particular we will show that the condition that $f_i = 0$ for $i = (q^m - 1) - j$ implies that coefficient $f_{i'} = 0$ for

[1] Strictly speaking, the classical BCH codes fix $n = q^m - 1$, but I see no reason to do so. If you want the classical codes, just puncture the codes we obtain and you'll get a code matching the classical parameters!

$i' = (q^m - 1) - qj$ under the restriction that f maps $\mathbb{F}_{q^m}$ to $\mathbb{F}_q$. This allows us to recover some of the lost dimensions, and gives us the required bound. To do so we first need to understand the role that $\mathbb{F}_q$ plays within $\mathbb{F}_{q^m}$.

4.6 $\mathbb{F}_q$ versus $\mathbb{F}_{q^m}$

We have seen how we can go from the field $\mathbb{F}_q$ to its extension $\mathbb{F}_{q^m}$. (See Lecture Notes of Algebra - Lecture 2.5). In Section 6 there, we described a canonical representation of $\mathbb{F}_{q^m}$ as an m-dimensional vector space over $\mathbb{F}_q$, which contains the original field $\mathbb{F}_q$ as a one-dimensional subspace.

Now we invert the question. Suppose someone gives us the field $\mathbb{F}_{q^m}$ and asks for the elements corresponding to the base field $\mathbb{F}_q$. Can we obtain this subfield without referring to our construction? What is the structure of a subfield $\mathbb{F}_q$ in the bigger field $\mathbb{F}_{q^m}$? Are these well-defined objects inside a field, or does a field $\mathbb{F}_{q^m}$ contain many copies of $\mathbb{F}_q$ floating around inside it? These are questions that no self-respecting algebraist would ask — so we will. (A reader comfortable with finite fields can safely skip this section.)

Recall that every element $\alpha \in \mathbb{F}_q$ satisfies the property $\alpha^q = \alpha$. We can look for elements in $\mathbb{F}_{q^m}$ satisfying this property. Turns out there are exactly q of these and these form the subfield $\mathbb{F}_q$ in $\mathbb{F}_{q^m}$. We will prove this below, but first let us see why it is reasonable that the roots of $x^q - x = 0$ should form a field. It is evident that if $\alpha^q = \alpha$ and $\beta^q = \beta$, then $(\alpha\beta)^q = \alpha\beta$ and so the roots of $x^q - x$ are closed under multiplication. But why is $(\alpha + \beta)^q = \alpha + \beta$? We prove this in a proposition that will be several times in this lecture.

Proposition 4.12 *Let $\mathbb{F}_q$ be a field of characteristic p. In other words, p is a prime and $q = p^r$. Then for every non-negative ℓ, and $\alpha, \beta \in F_q$, it is the case that $(\alpha + \beta)^{p^\ell} = \alpha^{p^\ell} + \beta^{p^\ell}$.*

Proof We start with an even more elementary fact. For every $\gamma \in \mathbb{F}_q$, $p \cdot \gamma = 0$ (where $p \cdot \gamma$ is just shorthand for $\gamma + \cdots + \gamma$, with p copies of γ in there). The simplest way to see this fact is that addition in $\mathbb{F}_q$ is just vector addition in $\mathbb{F}_p^r$ and the sum of p copies of any vector in $\mathbb{F}_p^r$ is zero.

Now we move to proving the proposition. Note that it is trivial for $\ell = 0$. So, we prove the propostion for the case $\ell = 1$. In this case $(\alpha + \beta)^p = \sum_{i=0}^{p} \binom{p}{i} \alpha^{p-i}\beta^i$. Now note that p divides $\binom{p}{i}$ for every $i \in [p-1]$. Since $p\gamma = 0$ for every $\gamma \in \mathbb{F}_q$, we have $\binom{p}{i}\alpha^{p-i}\beta^i = 0$ for every $i \in [p-1]$ and we are only left with the cases $i \in \{0, p\}$ which gives: $(\alpha + \beta)^p = \alpha^p + \beta^p$.

The proposition now follows by induction on ℓ. For $\ell \geq 2$, we have

$$\begin{aligned}
(\alpha + \beta)^{p^\ell} &= ((\alpha + \beta)^p)^{p^{\ell-1}} \\
&= (\alpha^p + \beta^p)^{p^{\ell-1}} \\
&= (\alpha^p)^{p^{\ell-1}} + (\beta^p)^{p^{\ell-1}} \\
&= \alpha^{p^\ell} + \beta^{p^\ell}.
\end{aligned}$$

■

Now we argue formally that the roots of $x^q - x$ do form the subfield $\mathbb{F}_q$ of $\mathbb{F}_{q^m}$.

Proposition 4.13 *There are exactly q elements $\alpha \in \mathbb{F}_{q^m}$ satisfying $\alpha^q = \alpha$ and these form the subfield $\mathbb{F}_q$ of $\mathbb{F}_{q^m}$.*

Proof We've already argued that the roots are closed under addition and multiplication. Thus the roots of $x^q - x$ form a field. It suffices to argue that the cardinality of this field is q.

It is easy to see there are at most q elements α in $\mathbb{F}_{q^m}$ satisfying $\alpha^q - \alpha = 0$, since these are roots of the degree q polynomial $x^q - x$. To see that this polynomial has q roots, note that every element of $\mathbb{F}_{q^m}$ is a root of $x^{q^m} - x$. Furthermore we have $x^{q^m} - x = (x^q - x) \cdot h(x)$, where $h(x) = x^{q^m - q} + x^{q^m - 2q} + \cdots + x^q + 1$ is a polynomial of degree $q^m - q$. Thus every element of $\mathbb{F}_{q^m}$ is a either a root of the polynomial $x^q - x$ or of $h(x)$. Since at most $q^m - q$ elements can be roots of $h(x)$, we have that $h(x)$ has exactly $q^m - q$ roots, and $x^q - x$ has exactly q roots. ■

We conclude that the roots of $x^q - x$ in $\mathbb{F}_{q^m}$ define the subfield $\mathbb{F}_q$ of $\mathbb{F}_{q^m}$ uniquely. Thus the operation of restricting a code in $\mathbb{F}_{q^m}^n$ to $\mathbb{F}_q^n$ is not arbitrary (such as say, as restricting the code to S^n for some arbitrary subset S of $\mathbb{F}_{q^m}$). Next we extend our understanding of subfields to functions mapping $\mathbb{F}_{q^m}$ to $\mathbb{F}_q$.

4.7 Functions from $\mathbb{F}_{q^m}$ to $\mathbb{F}_q$

We first extend Propositions 4.12 and 4.13 to the case of functions.

Proposition 4.14 *For functions $f, g : \mathbb{F}_{q^m} \to \mathbb{F}_{q^m}$, the following hold:*

1. $(f + g)^q = f^q + g^q$.
2. $f^q = f$ *if and only if the range of f is contained in $\mathbb{F}_q$.*

We omit the proof since it is straightforward given Propositions 4.12 and 4.13. We now move to the proposition that is the crux of the improved dimension analysis.

Proposition 4.15 *Suppose $f(x) = \sum_{i=0}^{n-1} f_i x^i$ is a polynomial over $\mathbb{F}_{q^m}$ mapping $\mathbb{F}_{q^m}$ to $\mathbb{F}_q$. Suppose $j, \ell \in \{1, \ldots, n-2\}$ are such that $\ell = qj (\mathrm{mod}\ (n-1))$. Then $f_\ell = 0$ if $f_j = 0$. Furthermore $f_{n-1} \in \mathbb{F}_q$.*

Proof Since f maps $\mathbb{F}_{q^m}$ to $\mathbb{F}_q$, we must have $f^q = f$ (Proposition 4.14, Part 2). Using Proposition 4.14, Part 1, we get that $f(x)^q = \sum_{i=0}^{n-1} (f_i)^q x^{iq}$. Since this expansion has terms of degree larger than $n-1$, we reduce terms above using the identity $x^n = x$ to get that f^q takes the form $\sum_{i=0}^{n-2} (f_i)^q x^{iq (\mathrm{mod}\ (n-1))} + (f_{n-1})^q x^{n-1}$. (The coefficient f_{n-1} needs to be treated separately since we don't have $x^{n-1} = 1$, but only $x^n = x$. Nevertheless each term can be verified to be correct.) Now using the fact that the functions f^q and f are identical over $\mathbb{F}_{q^m}$, we get that $f_0^q = f_0$, $f_{n-1}^q = f_{n-1}$ and $f_{qj (\mathrm{mod}\ (n-1))} = f_j^q$. The proposition follows as a special case. ■

Before going on to the final claim giving the better bound on the dimension of BCH codes, let us just ensure that the constraints we are looking at are all *linear*. We already insisted that the collection of functions mapping $\mathbb{F}_{q^m}$ to $\mathbb{F}_q$ is linear, but in what representation? The natural representation of such a function $f : \mathbb{F}_{q^m} \to \mathbb{F}_q$ is $\mathbf{f} = \langle f(\alpha)\rangle_{\alpha \in \mathbb{F}_{q^m}}$. But, in such a case, are constraints of the form $f_i = 0$ linear in this representation, (where f_i is the coefficient of x^i in the polynomial representation of f)? The answer is affirmative, and the proposition below asserts this.

Proposition 4.16 *Let $f = \sum_{i=0}^{n-1} f_i x_i$ be a polynomial mapping $\mathbb{F}_{q^m}$ to $\mathbb{F}_q$. Then the constraint $f_i = 0$ is a conjunction of at most m linear constraints on the the vector $\mathbf{f} = \langle f(\alpha)\rangle_{\alpha \in \mathbb{F}_{q^m}}$. The condition $f_{n-1} = 0$ is one linear constraint on the vector $\mathbf{f}$.*

Proof The first part is easy if we recall the right relationships between $\mathbb{F}_q$ and $\mathbb{F}_{q^m}$, and in particular that elements of $\mathbb{F}_{q^m}$ form a m-dimensional vector space over $\mathbb{F}_q$. Viewing $\mathbf{f}$ as a vector from $\mathbb{F}_{q^m}^n$ and recalling that interpolation is linear, we obtain that f_i is a $\mathbb{F}_{q^m}$ is a linear form in $\mathbf{f}$. But since $\mathbb{F}_q^m$ can be embedded in $\mathbb{F}_q^m$ preserving linearity, we get that f_i is given my m $\mathbb{F}_q$-linear forms in $\mathbf{f}$. The condition $f_i = 0$ just forces all these linear forms to be zero.

For the furthermore part, recall by Proposition 4.15 that f_{n-1} is already an element of $\mathbb{F}_q$ which is a one-dimensional subspace of $\mathbb{F}_q^m \cong \mathbb{F}_{q^m}$. Thus the $f_{n-1} = 0$ forces one additional linear constraint on this space, and thus on the vector $\mathbf{f}$. ■

4.8 Bounding the dimension of BCH codes

We are now ready to prove a stronger bound on the dimension of BCH codes.

Lemma 4.17 *The dimension of the code $\mathrm{BCH}_{q,m,d}$ is at least $q^m - 1 - m\left\lceil \frac{(d-2)(q-1)}{q} \right\rceil$.*

Proof The proof is straightforward given all our work so far. The basic idea is to consider the space of all functions from $\mathbb{F}_{q^m}$ to $\mathbb{F}_q$, which forms a $\mathbb{F}_q$-vector space of dimension n. Viewing these functions as polynomials from $\mathbb{F}_{q^m}[x]$, we then restrict them to have zero as the coefficients of x^i for $i \in \{n-(d-1), \ldots, n-1\}$.

The condition that the coefficient of x^{n-1} is zero imposes one linear constraint and reduces the dimension of the space to $n-1$. The remaining conditions, corresponding to coefficients of x^i for $i \in \{n-(d-1), \ldots, n-2\}$, lead to at most m conditions each. However, we don't need to impose all such conditions. In particular we can skip every qth condition (starting at $n-2$ and going down) since these are exponents of the form $\ell = (n-1) - qj$, where j is a positive integer. Since $\ell = q(n-1-j)(\mathrm{mod}\ (n-1))$, by Proposition 4.15, the condition that the coefficient of x^ℓ equals zero is implied by the condition that the coefficient of x^{n-1-j} equals zero. Thus the constraints corresponding to the coefficients of x^i for $i \in \{n-(d-1), \ldots, n-2\}$, lead to at most $m\left\lceil \frac{(d-2)(q-1)}{q} \right\rceil$ linear constraints. Thus the dimension of the remaining space is at least $n - 1 - m\left\lceil \frac{(d-2)(q-1)}{q} \right\rceil$ as claimed. ■

We conclude with the following theorem summarizing the properties of the BCH codes.

Theorem 4.18 ([23, 49, 42]) *For prime power q, integers m and d, the code $\mathrm{BCH}_{q,m,d}$ is an $\left[n, n-1-m\lceil\frac{(d-2)(q-1)}{q}\rceil, d\right]_q$ code, for $n = q^m$.*

Of particular interest is the case $q = 2$ and even d. In this case, the messy ceilings and floor disappear and we the following nice corollary:

Corollary 4.19 *For every integer m and t, the code* $\mathrm{BCH}_{2,m,2t}$ *is a* $[n, n - 1 - (t-1)\log n, 2t]$*-code, for* $n = 2^m$.

As pointed out earlier, this is particularly nice, since it is very close to the Hamming bound, for constant d, and in particular matches the Hamming code of distance 4 (and by puncturing matches parameters of the Hamming code of distance 3 as well).

Bibliographic notes

Binary BCH codes were discovered independently by Hocquenghem [49] and Bose and Ray-Chaudhuri [23]. The extension to the general q-ary case is due to Gorenstein and Zierler [42].

The proofs in these notes are original. This proof was inspired by a simple self-contained description of BCH codes by Rüdiger Urbanke [115]. (The only short and self-contained description I could find!) These description here is different in that it does not go to the Fourier transform (something I am allergic to). An earlier version of these notes had a more elaborate proof. It was also buggy! Thanks to Amnon Ta-Shma for pointing out the bug. Fixing the bug resulted in the final (simpler) proof.

Chapter 5

6.897 Algorithmic Introduction to Coding Theory	September 24, 2001
Lecture 5	
Lecturer: Madhu Sudan	*Scribe: Mohammad Mahdian*

Today we will talk about:

- Asides on Reed-Solomon codes.
- Asymptotics of codes.
- Random codes.

5.1 Reed-Solomon codes

There are two equivalent ways to look at Reed-Solomon codes:

- Evaluation of polynomials: This is the definition that we have seen in the previous lecture. It's usually more convenient to prove theorems using this definition.
- Coefficients of polynomials: Some special cases of Reed-Solomon codes $\mathrm{RS}_{q,n,k}$ (q is the size of the alphabet (field), n is the block length, and k is the message length) can be described as follows:
  - **Generator polynomial:** A polynomial $g(x) \in \mathbb{F}_q[x]$ of degree $n-k$.
  - **Message:** coefficients of a polynomial $M(x)$ of degree less than k.
  - **Encoding:** coefficients of $g(x)M(x)$.

This definition gives us various codes depending on which g we pick. If g is a polynomial that divides x^n-1, then we get a general family of codes called "cyclic codes". For some further restrictions, we get BCH codes, which in turn contain the special cases of RS codes. To define the generator polynomial for these RS codes, we first need to introduce the notion of a primitive element of $\mathbb{F}_q$.

Definition 5.1 *An element $\alpha \in \mathbb{F}_q$ is a* primitive element *if $\alpha, \alpha^2, \ldots, \alpha^{q-1}$ are all the nonzero elements of the field.*

It is well-known that every field has many primitive elements. Given a primitive element α, we can define the generator polynomial g for $RS_{q,n,k}$ as follows:

$$g(x) := \prod_{i=1}^{n-k} (x - \alpha^i).$$

Using this polynomial with $n = q - i$ and $\alpha_i = \alpha^i$, we get an RS code $\mathrm{RS}_{q,n,k}$. (Recall that $\alpha_1, \ldots, \alpha_n$ are the set of points on which the polynomial in the first definition is evaluated.) A proof of this fact can be found at the end of this lecture.

5.2 Alternant Codes

Given n distinct elements $\alpha_1, \ldots, \alpha_n$ and n nonzero elements $\beta_1, \ldots, \beta_n$ of $\mathbb{F}_q$, the *Alternant Code* is defined as follows:

- **Message:** Polynomial $M(x)$ of degree less than k.
- **Encoding:** $\langle \beta_1 M(\alpha_1), \ldots, \beta_n M(\alpha_n) \rangle$.

In terms of the minimum distance, it is clear that alternant codes are equivalent to the RS codes. In particular a given coordinate of an encoding a given message in the alternant code is non-zero if and only if the same coordinate of the RS encoding of the same message is non-zero. However, the alternant code might have different properties in terms of its sub-codes. A sub-code of a code is defined as follows:

Let $q = 2^k$. We know that $\mathbb{F}_2$ is a subfield of $\mathbb{F}_q$. Consider an $[n, k, d]_q$ code C_1. The $\mathbb{F}_2$ sub-code of C_1, denoted C_2, is an $[n, k', d']_2$ code that consists of all codewords of C_1 that are also in $\mathbb{F}_2^n$ ($C_2 = C_1 \cap \mathbb{F}_2^n$). Such an operation can be carried out in general for any pair of field $\mathbb{F}^{(1)} \subseteq \mathbb{F}^{(2)}$, and the resulting codes are called sub-field sub-codes of the original code.

The resulting codes have minimum distance at least as much as that of the original code. However their message length may be much smaller. In fact, a priori it is unclear as to why the sub-field sub-code should contain any non-zero vector. However their performance is not as bad as it looks! Many interesting families of codes can be obtained as sub-field sub-codes of Alternant codes. BCH codes form one such example, for some clever choice of $\alpha_1, \ldots, \alpha_n, \beta_1, \ldots, \beta_n$.

Readers more interested in this material can find it in [74, Chapter 12].

5.3 Asymptotics of Codes

So far we have seen a variety of codes:

Hamming code These codes have a good relationship between k and n, but d is small.

RS codes They meet the Singleton bound, but need large alphabet size.

Hadamard code $d = n/2$, but $k = \log n$, i.e., encoding increases the length of the message exponentially.

We sense that none of these codes is "good enough", but we have not defined a concept of "good" codes. To do so, we need to study the *asymptotics* of codes. To do so, we will consider infinite families of codes $\mathcal{C} = \{(n_i, k_i, d_i)_{q_i}\}_{i=1}^{\infty}$, with $\lim_{i\to\infty}\{n_i\} = \infty$.

Definition 5.2 *The* (message) rate *of a family of codes* $\mathcal{C} = \{(n_i, k_i, d_i)_{q_i}\}_{i=1}^{\infty}$*, denoted* $R(\mathcal{C})$*, is defined to be* $\liminf_{i\to\infty}\left\{\frac{k_i}{n_i}\right\}$*. The* relative distance *of* $\mathcal{C}$*, denoted* $\delta(\mathcal{C})$*, is defined to be* $\liminf_{i\to\infty}\left\{\frac{d_i}{n_i}\right\}$.

Definition 5.3 *A family of codes* $\mathcal{C}$ *is* asymptotically good *if* $R(\mathcal{C}), \delta(\mathcal{C}) > 0$.

When the family $\mathcal{C}$ is clear from context, we will skip the argument and just refer to R and δ.

One of the early "holy grails" of coding theory was to construct asymptotically good codes. This was achieved early on. We will see in this lecture that such codes do exist, and in the next lecture we will show how to construct a family of asymptotically good codes.

Every result in coding theory tends to have an asymptotic interpretation, and often the asymptotic version is much more succinct. For example, the Singleton bound ($n - k + 1 \geq d$) implies

$$\delta \leq 1 - R.$$

Similarly, the Hamming bound has an asymptotic interpretation. Recall that this bound says that for binary codes, $2^k \mathrm{Vol}(\frac{d-1}{2}, n) \leq 2^n$. Using the approximations $\frac{d-1}{2} \approx \frac{\delta n}{2}$ and $\mathrm{Vol}(pn, n) \approx 2^{H(p)n}$, we have

$$\text{For } q = 2, \qquad R + H\left(\frac{\delta}{2}\right) \leq 1.$$

The above bounds are shown in Figure 5.1. In the rest of this lecture, we will show that random binary codes satisfy $R \geq 1 - H(\delta)$. We don't know of an explicit construction for a code satisfying this bound.

5.4 The Gilbert-Varshamov bound

The Gilbert-Varshamov bound says that there is an infinite family of codes $\mathcal{C}$ satisfying $R(\mathcal{C}) \geq 1 - H(\delta(\mathcal{C}))$. We will present three proofs for this fact. These proofs are due to:

- Gilbert [36], who showed essentially that a random code has this property
- Varshamov [119] who showed that random linear codes have this property.
- Wozencraft [124] who constructed a small space of linear codes most of whose members meet the Gilbert-Varshamov bound.

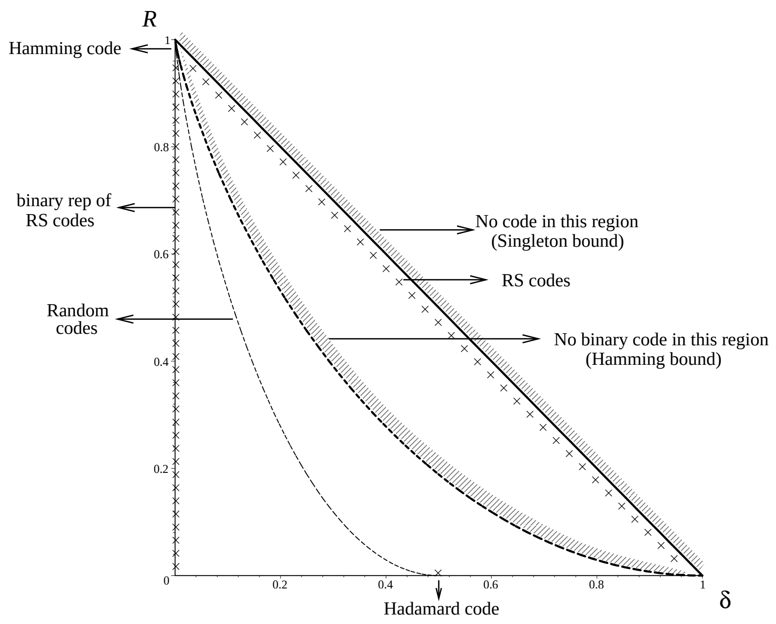


Figure 5.1:

5.4.1 Gilbert's code (Greedy code)

Gilbert showed the family of codes $\mathcal{C}$ with its nth element picked greedily according to the following procedure satisfies the bound $R(\mathcal{C}) \geq 1 - H(\delta(\mathcal{C}))$. Later we will view the result as showing that a randomized procedure leads to good codes with high probability.

- Greedy(n, d)
- Intitialize: $S \leftarrow \{0,1\}^n, C \leftarrow \emptyset$.
- Iterate until $S = \emptyset$:
  1. Pick $\mathbf{x} \in S$ and add $\mathbf{x}$ to C.
  2. Delete $B(\mathbf{x}, d)$ (ball of radius d around $\mathbf{x}$) from S. (See Figure 5.2.)

Lemma 5.4 *Fix $0 < \delta < \frac{1}{2}$ and $\epsilon > 0$ and let $R \geq 1 - H(\delta) - \epsilon$. Then for all sufficiently large n, the procedure* Greedy$(n, \lceil \delta n \rceil)$ *produces a code with at least 2^{Rn} codewords.*

Proof Let n be large enough so that $\mathrm{Vol}(d, n) \leq 2^{(H(\delta)+\epsilon)n}$.

Assume the algorithm picks K codewords. At every step, the greedy algorithm deletes at most $\mathrm{Vol}(d, n)$ elements from S. Therefore, since S started with 2^n elements, we have

$$K \geq \frac{2^n}{\mathrm{Vol}(d, n)} \geq 2^{(1-H(\delta)-\epsilon)n} = 2^{Rn}.$$

■

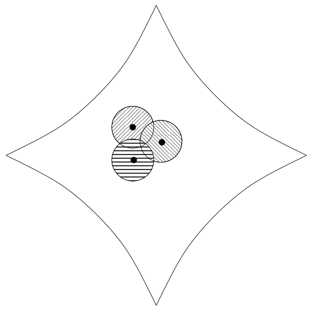


Figure 5.2: Hypercube and the greedy algorithm

The following theorem follows:

Theorem 5.5 *There exists a family of codes $\mathcal{C}$ with $R(\mathcal{C}) \geq 1 - H(\delta(\mathcal{C}))$.*

An alternate way to get a similar result is to follow the following probabilistic procedure.

1. Pick a subset $C' \subset \{0,1\}^n$ of size 2^k at random.
2. Let $B \subseteq C'$ be the set $\{\mathbf{x} \in C' | \exists \mathbf{y} \in C' - \{\mathbf{x}\} \text{ s.t. } \Delta(\mathbf{x}, \mathbf{y}) \leq d\}$.
3. Let $C = C' - B$. Output C.

We argue (informally) that for an appropriate setting of parameters, C' still has $\Theta(2^k)$ codewords and that its distance is at least d. To do so, note that the expected number of neighbours from $C' - \mathbf{x}$ that a vector $\mathbf{x}$ has of distance at most d is approximately $2^{k-n}\text{Vol}(d, n)$. For appropriate setting of parameters (still maintaining $R \approx 1 - H(\delta)$) this expected number of neighbours can be made arbitrarily small, say γ. Thus the probability that $\mathbf{x}$ belongs to B is at most γ. By Markov's inequaltiy, we have that the event that half of the elements of C' are in B occurs with probability at most 2γ. Deleting this set still leaves us with a C of size 2^{k-1} and this happens with probability at least $1 - 2\gamma$.

5.4.2 Varshamov's linear code

Varshamov's linear codes are constructed using the following probabilistic procedure:

> Random-Linear(n, k)
>
> Pick the entries of the $k \times n$ generator matrix $\mathbf{G}$ uniformly and independently at random from $\mathbb{F}_2^{k \times n}$. Let $C = \{\mathbf{y}\mathbf{G} : \mathbf{y} \in \mathbb{F}_2^k\}$.

Lemma 5.6 *Let $0 < \delta < \frac{1}{2}$ and $\epsilon > 0$. Let $R = 1 - H(\delta) - \epsilon$. For sufficiently large n and $k = \lceil Rn \rceil$, the procedure* Random-Linear(n, k) *produces a code with 2^k codewords and distance at least δn, with high probability.*

Proof We really need to prove two separate assertions here: First, that the matrix $\mathbf{G}$ has full column rank k so that the code does have 2^k codewords. Next, we need to show that no pair of distinct codewords in the code generated by $\mathbf{G}$ are within distance δn of each other. We will combine the two steps into one and simply argue that for every non-zero vector $\mathbf{y}$, it is the case that $\mathbf{yG}$ does not lie in $B(\mathbf{0}, \delta n)$. The first part (rank of $\mathbf{G}$) is implied by the fact that $\mathbf{yG} \neq \mathbf{0}$ for any non-zero $\mathbf{y}$. The second part follows, since we would have proved that no codeword has Hamming weight less than δn and we know that the the minimum distance of a linear code equals the minimum weight of a non-zero codeword.

Suppose n is large enough so that $\mathrm{Vol}(\delta n, n) \leq 2^{(H(\delta)+\epsilon/2)n}$. Let $d = \delta n$. For every fixed $\mathbf{y} \neq \mathbf{0}$ in $\mathbb{F}_2^k$, it is easy to see that $\mathbf{yG}$ is a random vector in $\{0,1\}^n$, and therefore,

$$\begin{aligned} \Pr[\mathrm{wt}(\mathbf{yG}) \leq d] &= \Pr[\mathbf{yG} \in B(\mathbf{0}, d)] \\ &= \frac{\mathrm{Vol}(d,n)}{2^n} \\ &\leq 2^{(H(\delta)+\epsilon/2-1)n}. \end{aligned}$$

Therefore, by the union bound, the probability that there is a $\mathbf{y}$ with $\mathrm{wt}(\mathbf{yG}) \leq d$ is at most $2^k 2^{(H(\delta)+\epsilon/2-1)n}$. If $R = \frac{k}{n} \leq 1 - H(\delta) - \epsilon$, then this probability is at most $2^{-(\epsilon/2)n}$ which goes to zero as $n \to \infty$. Therefore with high probability, the random procedure outputs a linear code with minimum distance at least δn. ■

5.4.3 Wozencraft construction

Varmashov's construction gives an algorithm of running time 2^{kn} for constructing the code. Wozencraft uses a clever idea to reduce this running time to 2^n. The idea is to find a family of disjoint sets $S_1, S_2, \ldots, S_t \subseteq \{0,1\}^n - \bar{\mathbf{0}}$ such that for every i, $S_i \cup \{\bar{\mathbf{0}}\}$ is a linear subspace of $\{0,1\}^n$.

Claim 5.7 *If such a family exists and $t \geq \mathrm{Vol}(d,n)$, then there is an i such that $S_i \cup \{\bar{\mathbf{0}}\}$ is a linear code with distance at least d.*

Proof Every vector $\mathbf{x} \in B(\mathbf{0}, d) - \{\mathbf{0}\}$ lies in at most one of the S_i's (since the S_i's are disjoint). Since $t > \mathrm{Vol}(d,n) - 1$, it follows that at least one of the S_i does not contain any of the elements of $B(\mathbf{0}, d) - \{\mathbf{0}\}$. Such an S_i has minimum weight at least d, and since $S_i \cup \{\bar{\mathbf{0}}\}$ is linear, it has distance at least d. ■

Furthermore, if we can construct this partition with the additional property that all $|S_i|$'s are equal, we will get a linear code of size $2^n/t$. Such a construction will be presented in the next lecture.

5.5 Appendix: Equivalence of Reed Solomon code

Here we show that the two definitions of Reed-Solomon codes coincide for appropriate choice of parameters. To be explicit let us reintroduce the two definitions (with more parameters).

Definition 5.8 *For prime power q, integers $k \le n \le q$ and a vector $\alpha = \langle \alpha_1, \ldots, \alpha_n \rangle \in \mathbb{F}_q^n$ of distinct elements of $\mathbb{F}_q$, the Reed-Solomon code $\mathrm{RS}_{q,n,k,\alpha}$ is the collection of vectors $\{\langle M(\alpha_1), \ldots, M(\alpha_n)\rangle | M \in \mathbb{F}_q[x], \deg(M) < k\}$.*

The second definition below is via the coefficients of polynomials.

Definition 5.9 *For a prime power q, a primitive element $\alpha \in \mathbb{F}_q$ and integer k, the alternate Reed-Solomon code $\mathrm{RS}'_{q,k,\alpha}$ is the collection of vectors $\{\langle c_0, \ldots, c_{q-2}\rangle | g(x)$ divides $C(x) = \sum_{i=0}^{q-2} c_i x^i\}$, where $g(x) = \prod_{j=1}^{q-1-k}(x - \alpha^j)$.*

The following proposition gives the equivalence:

Proposition 5.10 *For every prime power q, primitive element α and $k \le q-1$ and for $n = q-1$ and $\alpha = \langle \alpha^0, \ldots, \alpha^{n-1}\rangle$, $\mathrm{RS}_{q,n,k,\alpha} = \mathrm{RS}'_{q,k,\alpha}$.*

Proof Since both codes have the same number of codewords, it suffices to prove that every codeword according to the first definition is a codeword according to the second definition.

Consider a vector $\langle c_0, \ldots, c_{n-1}\rangle$ and the associated polynomial $C(x) = \sum_{i=0}^{n-1} c_i x^i$. To prove this vector is a codeword according to the second definition, it suffices to prove that $C(\alpha^j) = 0$ for every $j \in [n-k]$ (since this implies that $\prod_j (x - \alpha^j)$ divides $C(x)$).

Consider a message $\mathbf{m} = \langle m_0, \ldots, m_{k-1}\rangle$ and the associated polynomial $M(x) = \sum_{l=0}^{k-1} m_l x^l$. Let $\mathbf{c} = \langle M(\alpha^0), \ldots, M(\alpha^{n-1})\rangle$ be the encoding of $\mathbf{m}$ according to the first definition. Let $C_{\mathbf{m}}(x)$ be the polynomial with $\mathbf{c}$ as its coefficients, i.e., the coefficient of x^i is $M(\alpha^i)$. We show below that $C_{\mathbf{m}}(\alpha^j) = 0$ for every $j \in [n-k]$.

$$
\begin{aligned}
C_{\mathbf{m}}(\alpha^j) &= \sum_{i=0}^{n-1} M(\alpha^i)(\alpha^j)^i \\
&= \sum_{i=0}^{n-1}\sum_{l=0}^{k-1} m_l (\alpha^i)^l (\alpha^j)^i \\
&= \sum_{l=0}^{k-1} m_l \sum_{i=0}^{n-1} (\alpha^l \alpha^j)^i \\
&= \sum_{l=0}^{k-1} m_l \sum_{i=0}^{q-2} \gamma_{j,l}^i
\end{aligned}
$$

where $\gamma_{j,l} = \alpha^{j+l}$. Notice that for every j, l s.t. $j + l \ne q-1$, $\gamma_{j,l} \ne 1$. Notice further that for every such $\gamma_{j,l}$ the summation $\sum_{i=0}^{q-2} \gamma_{j,l}^i = 0$[1]. Since $l \in \{0, \ldots, k-1\}$, we find that $\gamma_{j,l} \ne 1$ for every $j \in [n-k]$. Thus for every $j \in [n-k]$, we find that $C_{\mathbf{m}}(\alpha^j) = 0$. This concludes the proof. ■

[1]This identity is obtained as follows: Recall that Fermat's little theorem asserts that $\gamma^{q-1} - 1 = 0$ for every non-zero γ in $\mathbb{F}_q$. Factoring the left hand side, we find that either $\gamma - 1 = 0$ or $\sum_{i=0}^{q-2} \gamma^i = 0$. Since $\gamma \ne 1$, the latter must be the case.

Chapter 6

6.897 Algorithmic Introduction to Coding Theory	September 26, 2001
Lecture 6	
Lecturer: Madhu Sudan	*Scribe: Nicole Immorlica*

Today we will talk about:

- Wozencraft construction continued
- Building codes from other codes
  - Parity check bit
  - Puncturing
  - Restriction
  - Direct Product
  - Concatenation
- Forney codes
- Justesen codes

6.1 Wozencraft construction (continued)

The Wozencraft construction gives a $2^{O(n)}$ time algorithm for constructing $[n, k, d]_2$ codes. We pick up where we left off in the last lecture. Recall our goal is to construct a family of sets $S_1, S_2, \ldots, S_t \subseteq \{0,1\}^n - \mathbf{0}$ such that

1. The sets are pairwise disjoint.
2. $\forall i$, $S_i \cup \{\mathbf{0}\}$ is a linear subspace of $\{0,1\}^n$.
3. $t \geq \mathrm{Vol}(d, n)$.

4. $\forall i, j : |S_i| = 2^k - 1$.

We saw last lecture that if we can construct such a family of sets, one of these sets will yield a $[n, k, d]_2$ code. Today we will see Wozencraft's construction of such a family of sets. We will show the construction only $n = 2k$. It is fairly simple to generalize it to a construction for $n = ck$ for any integer c.

We will use the correspondence between fields and vector spaces that preserves addition (see Lecture Notes on Algebra, Section 6). In particular we will view $\mathbb{F}_2^k$ as $\mathbb{F}_{2^k}$ and $\mathbb{F}_2^n$ as $\mathbb{F}_{2^k}^2$. The sets we will construct will be indexed by $\alpha \in \mathbb{F}_{2^k}$, with S_α defined as follows: $S_\alpha = \{(x, \alpha x) \mid x \in F_{2^k} - \{0\}\}$. We now verify that the S_α's satisfy the above conditions for $t = 2^k$ and d such that $\mathrm{Vol}(d, n) \leq t$.

1. S_α's are pairwise disjoint: In particular, For every $(x, y) \in F_{2^k}^2$, there is at most one α such that $(x, y) \in S_\alpha$, namely $\alpha = xy^{-1}$ provided y is non-zero and $\alpha = 0$ if $y = 0$. (If $x = 0$ then $(x, y) \notin S_\alpha$ for any α.)
2. $S_\alpha \cup \{0\}$ is linear: Clearly each S_α is a linear subspace of $F_{2^k}^2$ and is generated by the matrix $[1\alpha]$. Since the correspondence between $\mathbb{F}_2^k$ and $\mathbb{F}_{2^k}$ respects addition, it follows that $S_\alpha \cup \{0\}$ are linear over $\mathbb{F}_2$ as well.
3. There are clearly $t = 2^k$ of the S_α's. The condition $t \geq \mathrm{Vol}(d, n)$ follows from the definition of d.
4. It is also obvious that $|S_\alpha| = 2^k - 1$.

Taking the ratios k/n and d/n we note that the codes S_α always have a rate of $\frac{1}{2}$. Further if we fix any $\epsilon > 0$, and set $d = (H^{-1}(\frac{1}{2}) - \epsilon)n$ then for all sufficiently large n we have $\mathrm{Vol}(d, n) \leq 2^{n/2}$ and thus the family above gives a code of rate $\frac{1}{2}$ and relative distance approaching $H^{-1}(\frac{1}{2})$.

By a slightly more careful argument we can actually verify that most codes in the family achieve the Gilbert-Varshamov bound. Specifically, we can prove:

Theorem 6.1 *For every $\epsilon > 0$ and for all sufficiently large even numbers n, Wozencraft's construction with parameter n gives a family of $2^{n/2}$ codes with all but ϵ fraction of which are $[n, \frac{1}{2}n, (H^{-1}(\frac{1}{2}) - \epsilon)n]_2$-codes.*

Remarks:

1. Furthermore, for all such n, given an index i of a code from the family with parameter n, any specific entry of the generator matrix of the ith code can be computed in time polynomial in n.
2. If n is of the form $4 \cdot 3^t$, then the computation can be carried out in $O(\log n)$ space. This part follows from the fact that the irreducible polynomial for such $\mathbb{F}_{2^k}$ where $k = n/2$ is known explicitly and this polynomial is sparse. (Thanks to Dieter van Melkebeek (dieter@ias.edu) for pointing out this use of sparsity.)

Exercise: Extend the argument above to construct for every integer c, every $\epsilon > 0$, and all sufficiently large k, an ensemble of $2^{(c-1)k}$ codes such that all but an ϵ-fraction of the ensemble are $[ck, k, (H^{-1}(1 - \frac{1}{c}) - \epsilon)(ck)]_2$-codes. Your construction should take time $2^{O(ck)}$.

References: The Wozencraft ensemble of codes do not appear in any paper by Wozencraft. They are alluded to in a monograph by Massey [78, Section 2.5]. The actual family as described above is from Justesen's paper [56]. The extension asked for in the exercise is from the paper of Weldon [123].

6.2 Building codes from other codes

In the previous section we saw that asymptotically good codes exist. However, we had no explicit construction for them. The second holy grail of coding theory is to construct in polynomial time binary codes that meet the GV-bound. No one knows how to do this yet. One approach to this problem is to create new codes from existing ones. We look at five ways of getting new codes from old codes. Four of them don't improve the asymptotics of the code. The fifth leads to constructions of families of asymptotically good codes. (However, they do not meet the GV-bound.)

6.2.1 Parity check bit

We recall a construction of Hamming (see notes for Lecture 3). Given a code $C = [n, k, d]_2$, create a new code $C' = [n+1, k, d']_2$ as follows. First encode the message using C to get a codeword $\mathbf{c}$ of length n. Then, add an extra bit which is the parity of the bits of c. This new codeword, $\mathbf{c}'$ has length $n+1$. Furthermore, as argued in Lecture 3, if n is odd, the new distance $d' = d+1$. Otherwise the distance may remain d.

The parity check bit operation does improve relative distance for codes of odd length but not for codes of even length. Furthermore, the rate suffers. So we can not repeat this method to obtain really great codes.

6.2.2 Puncturing

Given a code $C = [n, k, d]_q$, create a new code $C' = [n-t, k, d']_q$ by simply deleting t coordinates. The new distance d' will be $d - t \leq d' \leq d$. For $t = 1$ we can think of the puncturing operation as achieving the effect of the inverse of the parity check bit operation (in a very loose sense).

This operation has the benefit of decreasing the encoding length thereby improving the rate. But at the same time it sacrifices the minimum distance of the code and thus decreases the relative distance.

While this operation does not yield a generic construction method for good codes, it turns out to be very useful in special cases. Often the best known code for a specific choice of, say n and k, might be a code obtained from puncturing a well-known code of longer block length. In such cases, special features of the code are often used to show that the distance is larger than the proven bound. Note further that all linear codes are punctured Hadamard codes! So obviously puncturing can lead to good codes. The question remains: When does it work? and what part of the codes should be punctured?

6.2.3 Restriction

Given a code $C = (n, k, d)_q$ over an alphabet Σ, create a new code $C' = (n-1, k', d)_q$ by choosing $\alpha \in \Sigma$ and $i \in [n]$ and retaining only those codewords $\mathbf{c}$ in which the ith coordinate of the codeword is α. The code C' is then obtained by deleting the ith coordinate from all remaining codewords.

The resulting code has block length n. If we pick α so that it is the most common letter in the ith coordinate (among codewords of C) then at least q^k/q messages will remain in C'. Since codewords differed in d positions to start with, and the only codewords that remain agreed in the deleted coordinate, the new codewords are still at Hamming distance at least d.

Restriction does improve the relative distance, but not necessarily the rate.

6.2.4 Direct Product

Given a codes $C_1 = [n_1, k_1, d_1]_q$ and $C_2 = [n_2, k_2, d_2]_q$, the direct product of C_1 and C_2, denoted $C_1 \otimes C_2$, is an $[n_1 n_2, k_1 k_2, d_1 d_2]_q$ constructed as follows. View a message of $C_1 \otimes C_2$ as a k_2 by k_1 matrix $\mathbf{M}$. Encode each row of $\mathbf{M}$ by the code C_1 to obtain an k_2 by n_1 intermediary matrix. Encode each column of this intermediary matrix with the C_2 code to get an n_2 by n_1 matrix representing the codeword encoding $\mathbf{M}$. This process works generally - for linear as well as non-linear codes C_1 and C_2. We first show that the resulting code has distance at least $d_1 d_2$ in either case. Then we show that if C_1 and C_2 are linear, then the resulting code is also linear, and furthermore is the same as the code that would be obtained by encoding the columns with C_2 first and then encoding the rows with C_1.

We prove this new code has distance at least $d_1 d_2$. Consider two distinct message matrices $\mathbf{M}_1$ and $\mathbf{M}_2$. Let $\mathbf{N}_1$ and $\mathbf{N}_2$ be the intermediate matrices obtained after the first step of the encoding process. Let $\mathbf{C}_1$ and $\mathbf{C}_2$ be the final codewords obtained from these matrices. Suppose $\mathbf{M}_1$ and $\mathbf{M}_2$ differ on the ith row. Then $\mathbf{N}_1$ and $\mathbf{N}_2$ must differ on at least d_1 coordinates on the ith row. In particular they differ on at least d_1 columns. Say $j_1, \ldots, j_{d_1}$ are indices of d_1 such columns where $\mathbf{N}_1$ and $\mathbf{N}_2$ differ. Then the column-by-column encoding results in codewords $\mathbf{C}_1$ and $\mathbf{C}_2$ which differ on at least d_2 coordinates on each of these d_1 columns. Thus $\mathbf{C}_1$ and $\mathbf{C}_2$ differ on at least $d_1 d_2$ entries.

Next we show that $C_1 \otimes C_2$ is linear if C_1 and C_2 are linear, and the encoding functions used are linear functions.

Claim 6.2 *Let $\mathbf{R}_1 \in \mathbb{F}_q^{k_1 \times n_1}$ generate the code C_1 and let $\mathbf{R}_2 \in \mathbb{F}_q^{k_2 \times n_2}$ generate the code C_2. Then the direct product code $C_1 \otimes C_2$ is a linear code that has as its codewords $\{{\mathbf{R}_2}^T \mathbf{M} \mathbf{R}_1 \mid \mathbf{M} \in \mathbb{F}_q^{k_2 \times k_1}\}$.*

Remark: As a consequence, we note that it does not matter if we encode the rows first and then the columns as above or vice versa.

Proof The proof follows easily from the fact that the intermediate matrix equals $\mathbf{M}\mathbf{R}_1$ and thus the final matrix equals ${\mathbf{R}_2}^T(\mathbf{M}\mathbf{R}_1)$. The interchangeability follows from associativity of matrix multiplication. The linear follows from the fact that the matrix ${\mathbf{R}_2}^T \mathbf{M}_1 \mathbf{R}_1 + {\mathbf{R}_2}^T \mathbf{M}_2 \mathbf{R}_1$ is just the encoding of $\mathbf{M}_1 + \mathbf{M}_2$ and the matrix $\alpha {\mathbf{R}_2}^T \mathbf{M}_1 \mathbf{R}_1$ is the encoding of $\alpha \mathbf{M}_1$, where $\alpha \in \mathbb{F}_q$. ∎

Exercise: In general the direct product of two codes depends on the choice of the encoding function.

Prove that this is not the case for linear codes. Specifically, prove that if $\mathbf{R}_1$ and $\mathbf{R}'_1$ generate C_1 and $\mathbf{R}_2$ and $\mathbf{R}'_2$ generate C_2, then $\{\mathbf{R}_2{}^T\mathbf{M}\mathbf{R}_1 \mid \mathbf{M}\} = \{\mathbf{R}'_2{}^T\mathbf{M}\mathbf{R}'_1 \mid \mathbf{M}\}$.

Again, the direct product does not help in the construction of asymptotically good codes. E.g. if we started with codes C_1 and C_2 of rate and relative distance $\frac{1}{10}$, then the resulting code is weaker and has rate and relative distance of only $\frac{1}{100}$.

So far all the operations on codes have been ineffective in getting to asymptotically good codes. In retrospect one may say that this is because all these operations fixed the alphabet and tried to play around with the other three parameters. A simply but brilliant idea, due to Forney [54], showed how to extend the game to include the alphabet size in the parameters altered/expoited by the operations on codes. This operation is that of "concatenating codes". This method turns out to have profound impact on our ability to construct asymptotically good binary codes. We describe this method an its consequences in the next section.

6.3 Concatenation of codes

To motivate the notion of concatenation, let us recall the example using Reed-Solomon codes on CD players. Reed-Solomon codes were defined on large alphabets, while CD players work with the binary alphabet. However, given an $[n, k, d]_{2^r}$ Reed-Solomon code, we interpreted this code as an $[nr, kr, d]_2$ *binary* code by naively representing the alphabet of the RS code, elements of $\mathbb{F}_{2^r}$, as binary strings of length r. The main idea of concatenation is to focus on this "naive interpretation" step and to generalize it so that elements of $\mathbb{F}_{2^r}$ can be represented by binary strings of length larger than r. Note that the main loss in performance is due to the fact that in going from strings of length n (over $\mathbb{F}_{2^r}$) to binary strings of length nr, we did not increase the minimum distance of the code, and so lost in terms of the relative distance. A careful choice of the encoding in the second step ought to be able to moderate this loss, and this is exactly what the method of concatenation addresses.

As in the case of direct product codes, it is best to explain concatenation of codes in terms of the encoding functions. First we define the l-fold concatenation of a single encoding function.

Definition 6.3 *For positive integer l, linearity preserving bijective map $\pi : \mathbb{F}_{q^k} \to \mathbb{F}_q^k$ and encoding function $E : \mathbb{F}_q^k \to \mathbb{F}_q^n$ the* l-fold concatenation *of E is the function $\diamond_l^\pi E : \mathbb{F}_{q^k}^l \to \mathbb{F}_q^{nl}$ given by $\langle \mathbf{x}_1, \ldots, \mathbf{x}_l \rangle \mapsto \langle E(\pi(\mathbf{x}_1)), \ldots, E(\pi(\mathbf{x}_l)) \rangle$, where $\mathbf{x}_i \in \mathbb{F}_{q^k}$ for $i \in [l]$.*

Typically the exact map $\pi : \mathbb{F}_{q^k} \to \mathbb{F}_q^k$ is irrelevant so we will simply ignore it. Further if l is clear from context, we will ignore it and simply refer to the map $\diamond E$. We now define the concatenation of two codes.

Definition 6.4 *For encoding functions $E_1 : \mathbb{F}_{q^{k_2}}^{k_1} \to \mathbb{F}_{q_2^k}^{n_1}$ and $E_2 : \mathbb{F}_q^{k_2} \to \mathbb{F}_q^{n_2}$ (and some implicit bijection $\pi : \mathbb{F}_{q^{k_2}} \to \mathbb{F}_q^{k_2}$), the concatenation of E_1 and E_2 is the function $E_1 \diamond E_2 : \mathbb{F}_q^{k_1 k_2} \to \mathbb{F}_q^{n_1 n_2}$ given by*

$$\mathbb{F}_q^{k_1 k_2} \xrightarrow{\pi^{-1}} \mathbb{F}_{q^{k_2}}^{k_1} \xrightarrow{E_1} \mathbb{F}_{q^{k_2}}^{n_1} \xrightarrow{\pi} \left(\mathbb{F}_q^{k_2}\right)^{n_1} \xrightarrow{\diamond E_2} \left(F_q^{n_2}\right)^{n_1} \longrightarrow F_q^{n_1 n_2}.$$

In the message $\langle \mathbf{x}_1, \ldots, \mathbf{x}_{k_1} \rangle$ is mapped to the vector $\diamond_{n_1}^\pi E_2(E_1(\langle \pi^{-1}(\mathbf{x}_1), \ldots, \pi^{-1}(\mathbf{x}_{k_1}) \rangle))$.

If the encoding functions E_1, E_2 are linear maps giving linear codes C_1 and C_2 respectively, then $E_1 \diamond E_2$ is a linear map whose image is denoted by $C_1 \diamond C_2$. It may be verified that $C_1 \diamond C_2$ is a function of C_1 and C_2 alone and not dependent on E_1, E_2 or π. It is customary to call the code C_1 the *outer code* and the code C_2 the *inner code*, and $C_1 \diamond C_2$ is the concatenated code.

The next proposition verifies the distance properties of concatenated codes.

Proposition 6.5 *If C_1 is an $[n_1, k_1, d_1]_{q^{k_2}}$-code and C_2 is an $[n_2, k_2, d_2]_q$-code then $C_1 \diamond C_2$ is an $[n_1 n_2, k_1 k_2, d_1 d_2]_q$-code.*

Proof The only part that needs to be verified is the distance. To do so consider the encoding of a non-zero message. The encoding by E_1 leads to an intermediate word from $\mathbb{F}_{q^{k_2}}^{n_1}$ that in non-zero in d_1 coordinates. The n_1-fold concatenation of E_2 applied to the resulting codeword produces d_2 non-zero symbols in every block where the outer encoding produced a non-zero symbol. Thus we end up with at least $d_1 d_2$ non-zero symbols in the concatenated encoding. ■

If we ignore the non-trivial behavior with respect to the alphabet size, then the concatenation operator has essentially the same parameters as the direct product operator. However the concatenation operator allows the outer code to be over a larger alphabet and we have seen that it is easier to construct good codes over large alphabets. Thus the concatenation operator is strictly better than direct product. Below we show an example of non-trivial results it yields.

Example - RS $\diamond$ Hadamard: Suppose we concatenate an outer code that is an $[n, k, n-k]_n$-Reed-Solomon code with a $[n, \log n, \frac{n}{2}]_2$-Hadamard code. (Assume for this example that n is a power of 2.) Then the concatenated codes is an $[n^2, k \log n, \frac{n}{2}(n-k)]_2$-code. Depending on our choice of rate k/n of the outer code, we get a family of binary codes of constant relative distance and an inverse polynomial rate $R = \frac{k \log n}{n^2}$. This is a new range of parameters that we have not seen in the codes so far.

While it is possible to employ multiple levels of concatenation to improve the dependence of the block length n on the message length k making n closer and closer to being linear in n, we can never get an asymptotically good code this way. Informally, to get an asymptotically good family, we need both the inner code and outer code to be asymptotically good. In what follows, we will describe two approaches at getting constructions of asymptotically good codes using concatenation.

6.3.1 Forney codes/Zyablov bound

The first family of codes we describe are due to Forney [54], who described the basic idea of the codes, but did not stress the choice of parameters that would optimize the tradeoff between rate and relative distance. (Forney was after bigger fish, specifically an algorithmic version of Shannon's theorem. We will get to this when we get to algorithms.) The actual bounds were worked out by Zyablov [126] and are usually referred to as the Zyablov bounds.

The idea to get a polynomial time constructible family of asymptotically good codes is a simple one. As an outer code we will use a Reed-Solomon code over an n-ary alphabet, say an $[n, k, n-k]_n$-code. For the inner code, we will search for the best linear code in, say, Wozencraft's ensemble of codes. This takes exponential time in the block length of the inner code, but the block length of the inner code only needs to be linear in the message length and the message length of the inner code is only $\log n$. Thus the time it takes to find the best code in Wozencraft's ensemble is only polynomial in n.

Getting a little more specific, to construct a code of relatve distance δ, we pick δ_1 and δ_2 so that $\delta_1 \delta_2 = \delta$. For the outer code we pick an $[n, (1-\delta_1)n, \delta_1 n]_n$-RS-code. For the inner code we search

Wozencraft's ensemble to obtain an $[n', (1 - H(\delta_2))n', \delta_2 n']_2$-code with $(1 - H(\delta_2))n' = \log n$. The resulting code has block length $nn' = O(n \log n)$, relative distance δ and rate $(1 - \delta_1)(1 - H(\delta_2))$. Thus we obtain the following theorem:

Theorem 6.6 *For every $\delta \in (0, \frac{1}{2})$, there exists an infinite family of polynomial time constructible codes $\mathcal{C}$ with rate R and relative distance δ satisfying*

$$R \geq \max_{\delta \leq \delta_2 < \frac{1}{2}} \left\{ (1 - H(\delta_2)) \cdot \left(1 - \frac{\delta}{\delta_2}\right) \right\}. \tag{6.1}$$

The bound (6.1) above is the Zyablov bound.

6.3.2 Explicit constructions

We take a brief digression to discuss what it means to construct a code explicitly. It is clear that this ought to be a complexity-theoretic definition, since a code is a finite set and one can obviously enumerate all finite sets to see if one of them gives, say, an (n, k, d)-code. The constructions of Gilbert took exponential time, while Varshamov's is a randomized polynomial time construction that possibly returns an erroneous solution (to the task of finding an $[n, k, d]$ code). We asserted that Forney's construction is somehow explicit, and yet this is not satisfactory to many mathematicians. Here we enumerate some criteria for explicit constructions for the case of codes (though similar criteria apply to constructions of all combinatorial objects).

Let $\{\mathcal{C}_{R,\delta}\}_{(R,\delta)}$ be a collection of families of codes, where the family $\mathcal{C}_{R,\delta}$ has rate R and relative distance δ. The following are possible notions of $\mathcal{C}$ being explicitly constructible:

Polytime For every $0 < R < 1$ and $0 < \delta < 1$, there exists a polynomial p such that generator matrix of the ith element of the family $\mathcal{C}_{R,\delta}$, with block length n_i, is constructible in time $p(n_i)$, if such a family exists.

Uniform polytime There exists a polynomial p such that for every $0 < R < 1$ and $0 < \delta < 1$, generator matrix of the ith element of the family $\mathcal{C}_{R,\delta}$, with block length n_i, is constructible in time $p(n_i)$, if such a family exists.

The difference between polytime constructibility and uniform polytime constructibility is relatively small. This distinction can be made in the remaining definitions too, but we will skip the extra quantifiers, and simply focus on what makes a code $\mathcal{C}$ constructible (leaving it to the reader to find a preference within uniform and nonuniform time bounds).

Logspace The generator matrix of the ith member of $\mathcal{C}$ is constructible in logarithmic space. (This implies that $\mathcal{C}$ is polynomial time constructible.)

Locally Polytime Constructible [1] Here we will require that a specific entry, say the j, lth entry, of the generator matrix of the ith member of the code $\mathcal{C}$ be computable in time polynomial in the size of the binary representation of i, j, l. (Note this representation has size logarithmic in n and so this notion is much more explicit than earlier notions.)

Locally Logspace Constructible The j, lth entry of the generator matrix of the ith code is logspace constructible in the length of the binary representations of i, j and l.

[1] Actually, this notion does not have a name and I had to generate one on the fly. Thanks to Anna Lysyanskaya for suggesting this name.

As noted, the requirements get more stringent as we go down the list above. The notion of Locally Logspace Constructible is about as strong a requirement we can pose without getting involved with machine-dependent problems. (What operations are allowed? Why? etc.)

Forney's codes, as described above, are polytime constructible, but not uniform polytime or logspace constructible. The next family of codes we will describe are locally logspace constructible, making them as explicit as we could desire (define?).

6.3.3 Justesen Codes

The principal barrier we seem to face in producing codes explicitly is that we know how to construct smaller and smaller ensembles of good codes, but we don't know how to get our hands on any particular good one. In fact in the ensembles we construct almost all codes are good. Is there any way to use this fact? Justesen's idea [56] is a brilliant one — one that "derandomizers" should take note of: On the one hand we can produce a small sample space of mostly good codes. On the other hand we need one good code that we wish to use repeatedly — n_1 times in the concatenation. Do we really need to use the same code n_1 times? Do they all have to be good? The answer, to both questions, is NO! And so, surprisingly enough, the ensemble of codes is exactly what suffices for the construction. Specifically, we take an $[n_1, k_1, d_1]_{q^{k_2}}$-outer code with encoding function E_1 and an ensemble consisting of n_1 inner codes with the ith member denoted $E_2^{(i)}$. We encode a message $\mathbf{m}$ by first applying the outer encoding function to get $E_1(\mathbf{m})$ and then applying the ith inner encoding function to the ith coordinate of $E_1(\mathbf{m})$, getting the vector $\langle E_2^{(1)}((E_1(\mathbf{m}))_1), \ldots, E_2^{(n_1)}((E_1(\mathbf{m}))_{n_1})\rangle$.

The above definition can be formalized to get a notion of concatenating an $[n_1, k_1, *]_{q^{k_2}}$-outer code with an ensemble containing n_1 $[n_2, k_2, *]_q$-inner codes ($*$ representing the fact that the distances are unknown, or possibly not all the same). Denoting the outer code by C_1, and the inner ensemble by $\overline{C_2}$, we extend the notation for concatenation and use $C_1 \diamond \overline{C_2}$ to denote such concatenations. The following proposition shows how the parameters of the concatenated codes relate to those of the outer code and inner ensemble.

Proposition 6.7 *Let C_1 be an $[n_1, k_1, d_1]_{q^{k_2}}$ code. Let $\overline{C_2}$ be an ensemble of n_1 $[n_2, k_2, *]_q$-codes of which all but ϵ-fraction have minimum distance d_2. Then the concatenated code $C_1 \diamond \overline{C_2}$ is an $[n_1 n_2, k_1 k_2, (d_1 - \epsilon n_1) d_2]_q$ code.*

Proof The proof follows from the fact that the first level encoding of a non-zero message leaves at least d_1 coordinates that are non-zero. At most ϵn_1 of the inner codes do not have minimum distance d_2. Thus at least $d_1 - \epsilon n_1$ coordinates, when encoded by $\overline{C_2}$ result in d_2 non-zero zymbols each. The distance follows. ■

Note that it is not entirely trivial to find an ensemble with just the right parameters: To use every element of the ensemble at least once, we need the inner ensemble size to be no larger than the outer block length. To use an RS code at the outer level, we need the outer block length to be no larger than the outer alphabet size. To use concatenation, we need the number of outer alphabet size to be no larger than the number of inner codewords. Putting it all together, we need an ensemble with no more members than codewords per member of the ensemble. Fortunately enough, this is exactly what is achieved by Wozencraft's ensemble, so we can use it. Consequenntly we get one fully explicit (locally logspace constructible) family of error-correcting codes on the Zyablov bound. In particular the code is asymptotically good.

Theorem 6.8 *For every $0 < \delta < H^{-1}(\frac{1}{2})$, there exists a locally logspace constructible infinite family of codes $\mathcal{C}$ that has relative distance δ and rate $\frac{1}{2}\left(1 - \frac{\delta}{H^{-1}(\frac{1}{2})}\right)$.*

The code above is obtained by concatenating a Reed-Solomon code of appropriate rate with the Wozencraft ensemble. We note that to get local logspace constructibility, we need the inner code length to be $4 \cdot 3^l$ for some integer l so that we can use the explicit construction of fields of size $2 \cdot 3^l$.

Chapter 7

6.897 Algorithmic Introduction to Coding Theory	October 1, 2001
Lecture 7	
Lecturer: Madhu Sudan	*Scribe: Adam Smith*

Today we'll describe a new family of codes, called algebraic-geometry codes. These codes generalize Reed-Solomon codes and yield surprisingly good codes over constant sized alphabets. We'll motivate the codes from two points of view. First we'll show how they challenge the "conjectured converse of the Gilbert-Varshamov bound". We'll then motivate the codes from the algebraic perspective.

The reader is warned that this lecture is not self-contained. It describes the "nature" of the codes without showing how to construct them, or why they even exist.

7.1 Motivation 1: Getting Better Parameters

Gilbert-Varshamov Bounds In order to motivate the construction of AG codes, we first recall the Gilbert-Varshamov (GV) bound for q-ary codes. Let the define the q-ary entropy function to be:

$$H_q(p) = p \log_q \frac{q-1}{p} + (1-p) \log_q \frac{1}{1-p}$$

The q-ary entropy function serves as an analogue to the binary entropy function when we deal with Hamming distance over q-ary alphabets. In particular, we get a similar volume approximation to the binary case. Let $B_q(\mathbf{0}, r)$ be the ball of radius r about $\mathbf{0}$ in $\mathbb{F}_q^n$. Further, let $\mathrm{Vol}_q(r,n) = |B_q(\mathbf{0}, r)|$ be the volume of this ball. For fixed $0 < p < 1$, we have

$$\mathrm{Vol}_q(pn, n) \approx q^{nH_q(p)}.$$

(Strictly, the approximation is really only any good when we're considering the logarithm of the volume, i.e. $Vol_q(pn,n) = q^{nH_q(p)(1-o(1))}$.)

Given this notation, we can state the q-ary GV bound: There exists an infinite family of codes $\mathcal{C}$ with rate R and relative distance δ satisfying:

$$R \geq 1 - H_q(\delta).$$

(Note that the random linear code will also attain this bound with high probability.)

In order to get a better feeling for that bound, we will fix δ, and let q tend to ∞. We get

$$\begin{aligned}
H_q(\delta) &= \delta \log_q(q-1) + \frac{1}{\log q} H_2(\delta) \\
&= \delta + \frac{H_2(\delta)}{\log q} + O(\frac{1}{q \log q}) \\
&\qquad \left(\text{using the fact that } \frac{\log(q-1)}{\log q} = 1 - \frac{\log q - \log(q-1)}{\log q} \approx 1 - 1/(q \log q) \right) \\
&\approx \delta + O(\frac{1}{\log q})
\end{aligned}$$

This means that for fixed δ, random codes will more or less achieve

$$R = 1 - \delta - O(1/\log q).$$

(Note: All logarithms are base 2 unless noted otherwise).

Note that the Singleton bound shows that no code can achieve $R > 1 - \delta$. The above bound shows that random (linear) codes approach the Singleton bound with an inverse logarithmic deficit in the alphabet size. Is this the best deficit we can hope for? We recall a familiar family which seems to do better.

Reed-Solomon Codes Recall that Reed-Solomon codes met the Singleton bound exactly and did so with an alphabet size of exactly n (for infinitely many choices of n). So Reed-Solomon codes seem to perform much better, although in this case one cannot really talk about q and n separately. With RS codes, we must have $q \geq n$, and so q must go to ∞ with n. Nonetheless, we know that $R = 1 - \delta - O(1/n)$ for RS codes (for any $q \geq n$), and so we can wave our hands and claim that we get

$$R = 1 - \delta - O(1/q).$$

So in effect the difference between $1 - \delta$ and R is growing inversely in q, rather than inversely in the logarithm of q. This motivates the question - can we somehow turn the Reed-Solomon intuition into a formal proof where we actually get to fix q and let n go to infinity and see the behavior of R vs. δ. Algebraic-geometry (AG) codes turn out to do exactly this.

AG codes The constructions of AG codes in fact yield

$$R = 1 - \delta - \frac{1}{\sqrt{q} - 1} = 1 - \delta - O(1/\sqrt{q}),$$

for every even prime power q (i.e., q must equal p^ℓ for prime p and even integer ℓ). These codes do not require that q scale with n i.e. in our "analysis" we may fix δ and q, and let $n \to \infty$; then we can let q increase and see how the parameters scale with q. While the codes do not achieve a deficit of an inverse in q, they do get a polynomial decay in this deficit as a function of q. So it becomes clear that as q grows this family of codes will outperform the Gilbert-Varshamov bound. Since the deficit functions are quite explicit, it is possible to compare them exactly and note that the function $\delta + 1/(\sqrt{q} - 1)$ is smaller than $H_q(\delta)$ for $\delta = \frac{1}{2}$ and $q \geq\approx 44$. The smallest square larger than this number is 49 and so we get that for $q \geq 49$ the algebraic-geometry codes outperform random codes!

7.2 Motivation 2: Generalizing Previous Constructions

Recall that in previous classes we got codewords by taking multivariate polynomials and evaluating them at all points in $\mathbb{F}_q^m$ (RS codes were the univariate case $m = 1$). Consider the univariate and bivariate cases with degree ℓ:

$$\begin{aligned} \text{Univariate case:} &\quad \text{Yields } [q^2, \ell^2, q^2 - \ell^2]_{q^2}\text{-code.} \\ \text{Bivariate case:} &\quad \text{Yields } [q^2, \ell^2, (q - \ell)^2]_q\text{-code.} \end{aligned}$$

Thus, reducing the alphabet size from q^2 to q cost us a reduction in the distance of $2\ell(q - \ell)$. Where does this difference come from? Intuitively, this is because in the bivariate plane $\mathbb{F}_q \times \mathbb{F}_q$, there are many small subspaces that encode quite inefficiently. For example, if we take any axis-parallel line in the plane. Knowing that a codeword is 0 on $\ell + 1$ of the points means it must be 0 at all q points on the line. Yet this code may still be non-zero elsewhere. Thus these q zeroes of the codeword only lead to $\ell + 1$ linear constraints on the codeword - a deficit roughly of $q - \ell$.

Another example of such a subspace is the circle $x^2 + y^2 = c$ for some constant c. One can prove that if the polynomial is 0 on 2ℓ of the points, then it must 0 everywhere on that circle—this could be up to $2q$ points, depending on c and on the field size.

The big idea: One idea for improving the performance of the bivariate code is to find a *subset* of points in the plane to use as evaluation points. If the subset is chosen carefully, then we might be able retain the distance properties of bivariate polynomials, while not picking too many points from any one dimensional curve. How does one go around trying to pick subsets with small intersection any one-dimensional curve? It turns out pick a different curve is a good way to minimize the intersection. Concentrating on algebraic curves (also known as "varieties") yields the basic idea for AG codes. Of course, once one starts playing around with the idea, one starts going to arbitrarily polynomials in many variables, and picks curves of low dimension in this high dimensional space to evaluate them. We'll get to the constructions briefly, but first we give some history behind these constructions.

7.3 History of AG codes

- AG codes were conceived by V.D. Goppa, a Russian coding theorist around 1975. When he published his first paper on this topic [41], it was not clear that the resulting codes would lead to new asymptotic results — in particular, the necessary algebra had not been studied yet. His paper motivated the study of the associated algebraic questions and eventually led to the breakthrough results.
- The first family of AG codes meeting the bound $R \geq 1 - \delta - \frac{1}{\sqrt{q}-1}$ were discovered by Tsfasman, Vladuts and Zink [114]. There underlying algebra was quite involved, and the constructions were very complicated. Manin and Vladuts [77] put some effort into showing that these codes were actually polynomial time constructible (with an $O(n^{30})$ construction time!).
- In a sequence of works Garcia and Stichtenoth [32, 33] simplified both the constructions and the proofs significantly. The resulting codes were built on curves that were completely explicit in the specification. The proofs involved in showing some of the properties are also significantly simpler. (One could even say these are "elementary", as works on algebraic geometry go.)

- Recent works by Shum et al. [104, 103] clarifies the Garcia-Stichtenoth papers further, eventually getting some codes with $\tilde{O}(n^3)$ construction time (the notation $\tilde{O}(\cdot)$ means ignoring polylog factors). The eventual hope is that these families will become completely explicit.

7.4 An Example in 2-D

We now return towards the task of describing algebraic-geometry codes. We will start by giving an example of a very *concrete* algebraic-geometry code — specifically a $[19, 6, 13]_{13}$ code. We will then attempt to show how the construction generalizes.

Our example will be a code based on the "plane" $\mathbb{F}_{13} \times \mathbb{F}_{13}$. We want to choose a subset of the plane with lots of points on which to evaluate low-degree bivariate polynomials in order to get codewords. We know that if we choose something that intersects too much with lines or circles, then we will have the same problem that we did with the whole plane — there will be subspaces don't contain much information.

Goppa's insight was to use an algebraic curve: pick a polynomial $R(x, y)$ of small degree, and consider the subset

$$S = V(R) = \{(x, y) \ : \ R(x, y) = 0\}.$$

In order to avoid intersecting too much with lines, circles and their other small-degree friends, we can choose R so that it's *irreducible* (see Bezout's theorem below). This, together with a judicious choice of which polynomials to use, will yield the desired properties.

So in our example, we will use:

- $q = 13$, i.e. $\mathbb{F} = \mathbb{Z}_{13}$
- $S = V(R)$ given by $R(x, y) = y^2 - 2(x-1)x(x+1)$.
- The polynomials we will use as codewords are linear combinations of the 6 basis polynomials $\{1, x, y, x^2, xy, x^3\}$. Notice that we aren't taking all polynomials of a given degree, but a carefully chosen subspace.

The parameters given by this code are described below:

- $q = 13$: By choice.
- $n = 19$: This parameter is typically verified by exhaustive search. In this specific case, it maybe verified that $S = \{(0,0), (\pm 1, 0), (2, \pm 5), (3, \pm 3), (4, \pm 4), (6, \pm 2), (7, \pm 3), (9, \pm 6), (10, \pm 2), (11, \pm 1)\}$.
- $k = 6$: A message is a polynomial of the form $a_0 + a_1 x + a_2 x^2 + a_3 x^3 + b_0 y + b_1 xy$ which is given by six coefficients, thus giving a message length of 6.
- $d = 13$, as we will argue below.

In general finding the block length (n) is non-trivial task, however the distance can be argued algebraically. In this special case, we do so by an ad-hoc argument tuned to give the best possible result. Later we will mention a slightly more general argument that is more illustrative of the principle behind the construction.

Claim 7.1 *Any non-zero polynomial $f(x,y) = a_0 + a_1x + a_2x^2 + a_3x^3 + b_0y + b_1xy$ is zero on at most six points in S.*

Proof We divide the analysis into two cases:

Case 1: $b_0 + b_1x$ does not divide $a_0 + a_1x + a_2x^2 + a_3x^3$.

Consider any common zero (α, β) of $f(x,y)$ and $R(x,y)$. Such a zero must also be a zero of any polynomial of the form $f \cdot g + R \cdot T$, for any polynomials $g(x,y)$ and $T(x,y)$. If we choose $g(x,y) = y(b_0 + b_1x) - (a_0 + a_1x + a_2x^2 + a_3x^3)$ and $T(x,y) = -(b_0 + b_1x)^2$, then the resulting polynomial $f \cdot g + R \cdot T$ is independent of y and equals $U(x) = 2(b_0+b_1x)^2(x-1)x(x+1) - (a_0+a_1x+a_2x^2+a_3x^3)^2$, a polynomial of degree 6 is x. Since (α, β) should be a root of any such polynomial we conclude that α is a root of $U(x)$ and thus there are at most six possible choices for α.

Next we note that $b_0 + \alpha b_1 \neq 0$, since in such a case $a_0 + a_1\alpha + a_2\alpha^2 + a_3\alpha^3$ would also have to be zero, which contradicts the assumption for this case. So we can now use the relation $f(x,y) = 0$ to conclude that $\beta = -(a_0 + a_1\alpha + a_2\alpha^2 + a_3\alpha^3)/(b_0 + b_1\alpha)$ and thus the number of pairs (α, β) that satisfy both f and R is at most six.

Case 2: $b_0 + b_1x$ divides $a_0 + a_1x + a_2x^2 + a_3x^3$.

In this case $f(x,y) = (b_0 + b_1x)(y + c_0 + c_1x + c_2x^2) = f_1(x)f_2(x,y)$. Since every zero of f is a zero of f_1 or of f_2, we can divide this analysis into two parts.

Note first that $f_1(x)$ and $R(x,y)$ have at most two common zeros (α, β), with $\alpha = -b_0/b_1$ and β satisfying $\beta^2 = 2(\alpha - 1)\alpha(\alpha + 1)$.

Next eliminating y from $f_2(x,y)$ and $R(x,y)$ we find that any common zero (α, β) must satisfy

$$(c_0 + c_1\alpha + c_2\alpha^2)^2 = 2(\alpha - 1)\alpha(\alpha + 1),$$
$$\text{and } \beta = -(c_0 + c_1\alpha + c_2\alpha^2).$$

Again, we conclude that there are at most four choices of α satisfying the first condition, every such choice leads to one β satisfying the second condition. Thus f_2 and R have at most four common zeroes.

Putting the two parts together, we see that in this case also f and R have at most six common zeroes.

■

Hence, we get a $[19, 6, 13]_{13}$ code. In contrast a Reed-Solomon code could give a slight increase in the distance, to 14, for a big increase in the alphabet size, to 19. This demonstrates, non-asymptotically, some of the tradeoffs that become possible with AG codes.

Bezout's Theorem Before going on to describing AG codes in their full generality, we mention one general principle that can be used to determine the distance, for AG codes in the plane. Of course, the trick in getting codes of large minimum distance is to pick the right curve R, and the right basis set of polynomials. A guide in this choice is Bezout's theorem, which gives us some idea of where to look:

Theorem 7.2 (Bezout, a long, long time ago) *If $A(x,y), B(x,y)$ are polynomials of degree d_1, d_2 respectively, then if they share more than d_1d_2 zeroes, they must share a common factor.*

A proof of this fact can be found in most texts on algebraic geometry or algebraic curves (cf. [120, Theorem 3.1]). The rough idea is to eliminate one of the variables y by finding polynomials $C(x, y)$ and $D(x, y)$ such that $A \cdot C + B \cdot D$ is a function of x alone. The fact that such a polynomial exists is not trivial, but not too hard to prove either. Once one gets this polynomial, it limits the number of choices in x and in turn one can limit the number of y's for every such x.

Returning to our previous example, the key to the example was ensuring that no polynomial in the subspace spanned by the basis elements could have a common factor with $R(x, y)$. We established this by ensuring R was irreducible, then by restricting the y degree to being just 1.

7.5 A General Result

Generalizing the idea of the previous example to more than two variables and one polynomial relation among them, one builds AG codes as follows:

1. Pick $m-1$ polynomial constraints on m variables:

$$\begin{aligned} P_1(x_1, ..., x_m) &= 0 \\ &\vdots \\ P_{m-1}(x_1, ..., x_m) &= 0 \end{aligned}$$

2. Let $S = V(P_1, ..., P_{m-1}) = \{\mathbf{x} \ : \ P_1(\mathbf{x}) = \cdots = P_{m-1}(\mathbf{x}) = 0\}$ be the set of common zeroes of $P_1, \ldots, P_{m-1}$.
3. Choose a linear subspace of polynomials which can't agree too often when restricted to S.

Of course, once again everything depends on how one chooses the polynomials $P_1, \ldots, P_{m-1}$ and then the basis of polynomials to evaluate at the set S. Specifically, one tries to pick polynomials $P_1, \ldots, P_{m-1}$ so that $|S|$ is large, while there exists a large collection of polynomials which don't agree too often on S.

Somewhat surprisingly, algebraic-geometers had been considering exactly this problem for a long time. A collection of polynomials is associated with a "curve" that consists of all zeroes of the polynomials over the algebraic closure, $\overline{\mathbb{F}_q}$, of $\mathbb{F}_q$. Such a curve consists of infinitely many points, but only finitely many are rational, i.e., from $\mathbb{F}_q^m$ (not surprising, since $\mathbb{F}_q^m$ is finite). To every curve they associate two integer parameters - its "genus" and the number of "rational points" lying on the curve. Both concepts are algebraic abstractions of analogous topological terms. Genus of a curve is a non-negative integer indicating the "twistedness" of the curve - the higher the genus, the more twisted the curve. From the point of view of establishing distance of codes, the best curves are the least twisted ones. However to get many rational points one needs twisted curves. This follows from a fundamental result in algebraic-geometry, first due to Hasse and Weil, and then improved by Drinfeld and Vladuts. The latter bound says that the number of rational points is at most the genus times $(\sqrt{q}-1)$. The curves used by the AG codes are the "examples" showing the tightness of this bound. Once one finds curves matching this bound, a second "fundamental" result of algebra, known as the Riemann-Roch theorem, is invoked to show that a large basis of "polynomials" exists over this curve. We'll get to this part later. First we'll say something about curves of small genus with many rational points.

The first family of curves meeting the Drinfeld-Vladuts bound were found by Tsfasman, Vladuts and Zink [114]. Analyzing these curves was significantly hard. Subsequently much more elementary families were discovered by Garcia and Stichtenoth [32, 33]. We describe a family developed by them below.

Example ([32])

1. We assume $q = r^2$ for some prime power r.
2. The curves are described by $2m - 1$ polynomial equations over $2m$ variables $x_1, \ldots, x_m, y_1, \ldots, y_m$.
3. The polynomial equations are the following:

$$\begin{aligned} x_i^{r+1} &= y_i^r + y_i \qquad (i = 1, ..., m) \\ x_i x_{i+1} &= y_i \qquad (i = 1, ..., m-1) \end{aligned}$$

A relatively simple inductive argument shows that there are roughly q^m rational points (in this $2m$ dimensional space) giving the set S. The genus of this curve, determined by a so-called "Hurwitz's genus formula" is then established to be at most $|S|/(\sqrt{q} - 1)$. To choose the right space of polynomials for encoding, one then uses the notion of "order" of a polynomial. We'll omit its definition (along with so many others) but explain what properties it satisfies, since that will be useful in understanding how to work with the codes (for solving some algorithmic tasks).

The order of a polynomial behaves similarly to degree:

- $\mathrm{ord}(\alpha f + \beta h) \leq \max\{\mathrm{ord}(f), \mathrm{ord}(h)\}$ where $\alpha, \beta \in \mathbb{F}_q$. Furthermore, $\mathrm{ord}(\alpha f + \beta h) = \mathrm{ord}(h)\}$ if $\beta \neq 0$ and $\mathrm{ord}(f) < \mathrm{ord}(h)$.
- $\mathrm{ord}(f \cdot h) = \mathrm{ord}(f) + \mathrm{ord}(h)$
- If f is zero on $\mathrm{ord}(f) + 1$ points on S, then $f \equiv 0$ on S.

By the properties above, it is clear that the set of polynomials of order at most t for a vector space. However unlike the case of univariate polynomials over $\mathbb{F}_q$, one need not have polynomials of every order. The Riemann-Roch theorem shows, however that there do exist polynomials of all but g values of the order, where g is the genus of the curve. (This is why we like curves of small genus). Applying this theorem to the curves of Garcia and Stichtenoth one now gets the family of AG codes as claimed.

Specifically, let $n = |S|$ be the number of rational points on the curve S (fixed once q and m are fixed). By the fact that these curves meet the Drinfeld-Vladuts bound, we get that its genus $g \leq n/(\sqrt{q} - 1)$. For any distance parameter d, let $\mathcal{P}_d$ be the set of all polynomials of order $n - d$. Notice that the evaluations of polynomials in L gives a linear code of distance d. By the Riemann-Roch theorem, we get that this space has dimension at least $n - d - g + 1 \geq n - d - n/(\sqrt{q} - 1)$. We obtain:

Theorem 7.3 (Very Good AG Codes Exist) *For every even power of a prime q, and every parameter $\delta < 1 - \frac{1}{\sqrt{q}-1}$, there exists an infinite family of q-ary linear codes of relative distance δ and rate $R \geq 1 - \delta - \frac{1}{\sqrt{q}-1}$. Further a generator matrix for such a code can be constructed in $\tilde{O}(n^3)$ time.*

Chapter 8

6.897 Algorithmic Introduction to Coding Theory	October 3, 2001
Lecture 8	
Lecturer: Madhu Sudan	*Scribe: Shien Jin Ong*

Today we will discuss upper bounds on the rate of any family code, given a lower bound on its relative distance. Specifically we will present the Plotkin bound and the Elias-Bassalygo bounds. En route we will also encounter a different bound, called the Johnson bound. The unifying theme for the lecture is that of finding upper bounds on the rate of codes by geometric arguments. In particular, we will embed Hamming space into Euclidean space and use the embeddings, in combination with geometric facts, to derive our proofs.

These notes include extensions of various proofs to q-ary cases - the lecture only covered the binary case. Throughout this lecture, we will use R to denote the rate of some (unspecified) family of codes and δ to denote the relative distance of the same family.

8.1 Embedding Hamming spaces in Euclidean spaces

To motivate our first bound, let us recall our current state of knowledge, for binary codes. On the one hand we have the Singleton and Hamming upper bounds on codes, with the latter dominating the former and showing $R \leq 1 - H(\delta/2)$. The best existence result, the Gibert-Varshamov (GV) bound, shows there exists a family with $R \geq 1 - H(\delta)$. For any $\delta > 0$, the bounds are far away from each other. However to get a qualitative sense of the gap, consider the largest distance that these bounds suggest are feasible for codes of positive rate. The Hamming bound rules out a relative distance of 1 for codes of positive rate. One the other hand, the GV bound only finds codes of positive rate with relative distance close to $\frac{1}{2}$. Clearly there is a qualitative gap here — and we address this gap first.

It is reasonably easy to guess which of these bounds is closer to the truth. Over a binary alphabet, random words have a relative distance of $\frac{1}{2}$ from each other and it seems quite impossible to construct codes with better distance. The Hamming bound on the other seems quite weak around these parts. We just need a way to formalize our intuition, and we will do so geometrically, by embedding the binary Hamming space into Euclidean space. We develop the embedding below.

Definition 8.1 (Embedding) *The embedding function* Embed : $\{0,1\} \to \mathbb{R}$, *mapping bits to the reals is given by* $\text{Embed}(0) = +1$ *and* $\text{Embed}(1) = -1$. *For* $n \geq 1$, *the* n*-dimensional embedding function extends the embedding above, with* Embed : $\{0,1\}^n \to \mathbb{R}^n$ *being given by*

$$\text{Embed}(\langle b_1, \ldots, b_n \rangle) = \langle \text{Embed}(b_1), \ldots, \text{Embed}(b_n) \rangle.$$

The property of this embedding is that Hamming distances are preserved as Euclidean distances, or in inner products. We recall some familiar definitions for vector spaces over the reals.

Definition 8.2 *For vectors* $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$, *the inner product between* $\mathbf{x}$ *and* $\mathbf{y}$, *denoted* $\langle \mathbf{x}, \mathbf{y} \rangle$, *equals* $\sum_{i=1}^n x_i y_i$. *The norm of a vector* $\mathbf{x}$, *denoted* $\|\mathbf{x}\|$, *is* $\sqrt{\langle \mathbf{x}, \mathbf{x} \rangle}$, *The Euclidean distance between* $\mathbf{x}$ *and* $\mathbf{y}$, *is simply the norm of* $\mathbf{x} - \mathbf{y}$.

The following proposition lists some of the elementary properties of our embedding from Hamming to Euclidean spaces. The proof is easily verified from the definitions and hence omitted.

Proposition 8.3 *For* $\mathbf{b}, \mathbf{c} \in \{0,1\}^n$, *the function* Embed *satisfies* $\langle \text{Embed}(\mathbf{b}), \text{Embed}(\mathbf{c}) \rangle = n - 2\Delta(\mathbf{b}, \mathbf{c})$, *where* $\Delta(\cdot, \cdot)$ *is the Hamming distance function. Hence, we have*

$$\|\text{Embed}(\mathbf{b})\|^2 = n, \text{ and } \|\text{Embed}(\mathbf{b}) - \text{Embed}(\mathbf{c})\|^2 = 4\Delta(\mathbf{b}, \mathbf{c}).$$

The embedding above thus allows us to transform questions about Hamming space into questions about Euclidean space. We will then appeal to our geometric intuition backed by linear algebra for proofs of coding-theoretic statements.

8.2 The Plotkin bound

Theorem 8.4 (Plotkin bound [89])

1. *An* $(n, k, d)_2$ *code with* $d \geq \frac{n}{2}$ *has at most* $2n$ *codewords. In other words,* $k \leq \log 2n$.
2. *If an* $(n, k, d)_2$ *code exists, then* $k \leq n - 2d + \log(4d)$.

Proof The first part is the harder part and the second part follows easily by using restrictions. We start with the first part.

Let $\mathbf{c}_1, \ldots, \mathbf{c}_m$ be all the codewords of the $(n, k, d)_2$ code. Let $\mathbf{x}_1, \ldots, \mathbf{x}_m$ be their embeddings, i.e., $\mathbf{x}_i = \text{Embed}(\mathbf{c}_i)$. By Proposition 8.3, we have that the inner product between $\mathbf{x}_i$ and $\mathbf{x}_j$, for $i \neq j$, is equal to $n - 2\Delta(\mathbf{c}_i, \mathbf{c}_j) \leq 0$ from the fact that the code has distance $d \geq n/2$. In Lemma 8.6 we show that in $\mathbb{R}^n$ there can be at most $2n$ vectors such that every pair has an non-positive inner product. The first part of the theorem follows.

To see the second part, let us write $n = 2d + \ell$ and suppose C is an $(n, k, d)_2$ code. Then by restricting C to the most commonly occuring pattern in the first ℓ coordinates and deleting these coordinates, we get a $(2d, k - \ell, d)_2$ code. By the first part of the theorem, we have $k - \ell \leq \log(4d)$. ∎

Before stating or proving the critical Lemma 8.6, we state the asymptotic version of Plotkin's bound.

Corollary 8.5 *For any family of binary codes $\mathcal{C}$ with rate R and relative distance δ, it is the case that $R \leq 1 - 2\delta$.*

We now move on to proving Lemma 8.6,

8.3 Geometric assertions, Linear-algebraic proofs

Our first goal here is to prove a geometric fact: In n dimensions there exist at most $2n$ vectors that pairwise subtend an angle of at least $\frac{\pi}{2}$ at the origin. We start with an intuitive, inductive proof. However the proof actually uses a fair bit of intuition about Euclidean spaces that we haven't (or won't) prove. We will then give an alternate, linear-algebraic proof that only uses the fact that the norm of a vector is non-negative, and that any $n+1$ vectors in n dimensions are linearly dependent.

Before proving the lemma, let us see why it is proving the right fact. We already know that the Hadamard code matches the Plotkin bound (or the first part of it) and so its embedding should match the lemma below tightly. But we can come up with a simpler example (geometrically the same, actually!) which shows that the lemma is tight. Take $\mathbf{x}_1, \ldots, \mathbf{x}_n$ to be the unit vectors along the coordinate axes, and let $\mathbf{x}_{n+i} = -\mathbf{x}_i$ for $i \in [n]$. This gives $2n$ non-zero vectors that are mutually at an angle of at least $\pi/2$.

Lemma 8.6 *If $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \mathbb{R}^n$ are non-zero and satisfy $\langle \mathbf{x}_i, \mathbf{x}_j \rangle \leq 0$ for every $i \neq j \in [m]$, then $m \leq 2n$.*

Proof We prove the lemma by induction on n. Without loss of generality we may assume that the vector $\mathbf{x}_m$ is the unit vector $\langle 1, 0, \ldots, 0 \rangle$. (The fact that this assumption follows without loss of generality is intuitively obvious, but would require some work if we decided to prove it!) Write $\mathbf{x}_i = \langle \alpha_i, \mathbf{y}_i \rangle$, where $\mathbf{y}_i \in \mathbb{R}^{n-1}$. Since we know that all other vectors have a non-positive inner product with $\mathbf{x}_m$, we find that $\alpha_i = \langle \mathbf{x}_i, \mathbf{x}_m \rangle \leq 0$. It follows that for distinct $i, j \in [m-1]$, we have

$$\langle \mathbf{y}_i, \mathbf{y}_j \rangle = \langle \mathbf{x}_i, \mathbf{x}_j \rangle - \alpha_i \alpha_j \leq \langle \mathbf{x}_i, \mathbf{x}_j \rangle \leq 0.$$

So we have $2m - 1$ vectors in $n - 1$ dimensions, as so we should be able to apply the inductive hypothesis — right? Well, that would be too strong and would yield $m \leq n$ — a bit better than some of the examples we have. Why does this happen. Well, the inductive hypothesis assumes $\mathbf{y}_i$'s are non-zero, and we didn't prove this yet. So to complete the lemma, we note that at most one of the $\mathbf{y}_i$'s may be zero. (If two, say $\mathbf{y}_1$ and $\mathbf{y}_2$ are zero, then their inner product would be positive!) We delete the zero vector and then we are left with $m-2$ non-zero vectors in $n-1$ dimensions with a pairwise non-positive inner product. This allows us to apply induction and the lemma is proved. ■

What if we were actually given that pairwise the vectors have a stricly negative inner product of say $-\alpha$? Could we improve the bound? The reader may try modifying the proof above to show that it this case the number of vectors is at most $1 + \frac{1}{\alpha}$, a bound independent of the number of dimensions. But the proof also starts to get more tedious. Motivated by such tasks, we now state a stronger lemma and give a self-contained proof. In particular, the proof is easier to verify (though possibly harder to conceive).

Lemma 8.7

1. *If α is a positive number and $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \mathbb{R}^n$ are unit vectors i.e., $\|\mathbf{x}_i\| = 1$, that satisfy $\langle \mathbf{x}_i, \mathbf{x}_j \rangle \leq -\alpha$, for every distinct pair $i, j \in [m]$, then $m \leq 1 + \frac{1}{\alpha}$.*
2. *If $\mathbf{y}, \mathbf{x}_1, \ldots, \mathbf{x}_m \in \mathbb{R}^n$ satisfy $\langle \mathbf{x}_i, \mathbf{x}_j \rangle \leq 0$ for distinct i, j, while $\langle \mathbf{y}, \mathbf{x}_i \rangle > 0$, then $m \leq n$.*
3. *If $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \mathbb{R}^n$ are non-zero and satisfy $\langle \mathbf{x}_i, \mathbf{x}_j \rangle \leq 0$ for every $i \neq j \in [m]$, then $m \leq 2n$.*

Proof We prove the three parts in order.

1. Let $\mathbf{z} = \mathbf{x}_1 + \cdots + \mathbf{x}_m$. On the one hand, we have $\langle \mathbf{z}, \mathbf{z} \rangle \geq 0$. On the other, we have

$$\begin{aligned} \langle \mathbf{z}, \mathbf{z} \rangle &= \sum_{i=1}^{m} \langle \mathbf{x}_i, \mathbf{x}_i \rangle + \sum_{i \neq j \in [m]} \langle \mathbf{x}_i, \mathbf{x}_j \rangle \\ &\leq m \cdot 1 + m(m-1) \cdot (-\alpha) \\ &= m \cdot (1 - (m-1)\alpha). \end{aligned}$$

Putting the two together, we have $1 - (m-1)\alpha \geq 0$, implying $m \leq 1 + \frac{1}{\alpha}$.

2. For this part, assume for the sake of contradiction that $m \geq n+1$. Then there must exist a linearly dependent set of vectors among the $\mathbf{x}_i$'s. Specifically there exist disjoint sets $S, T \subseteq [m]$ and positive λ_i, for $i \in S \cup T$, such that $\sum_{i \in S} \lambda_i \mathbf{x}_i = \sum_{j \in T} \lambda_j \mathbf{x}_j$. It is not necessary that both S and T be non-empty, but at least one is non-empty. Assume without loss of generality that S is non-empty. Let $\mathbf{z} = \sum_{i \in S} \lambda_i \mathbf{x}_i = \sum_{j \in T} \lambda_j \mathbf{x}_j$. Our analysis divides into two cases depending on whether $\mathbf{z} = \mathbf{0}$ or not.

Case: $\mathbf{z} \neq \mathbf{0}$: Here we obtain the following contradication:

$$\begin{aligned} 0 &< \langle \mathbf{z}, \mathbf{z} \rangle \\ &= \langle \sum_{i \in S} \lambda_i \mathbf{x}_i, \sum_{j \in T} \lambda_j \mathbf{x}_j \rangle \\ &= \sum_{i \in S} \sum_{j \in T} \lambda_i \lambda_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle \\ &\leq 0, \end{aligned}$$

where the last inequality uses the fact that S and T are disjoint and so $\langle \mathbf{x}_i, \mathbf{x}_j \rangle \leq 0$ for every $i \in S$ and $j \in T$.

Case: $\mathbf{z} = \mathbf{0}$: Here we use the existence of the vector $\mathbf{y}$ and obtain a contradiction as follows:

$$\begin{aligned} 0 &= \langle \mathbf{y}, \mathbf{0} \rangle \\ &= \langle \mathbf{y}, \mathbf{z} \rangle \\ &= \langle \mathbf{y}, \sum_{i \in S} \lambda_i \mathbf{x}_i \rangle \\ &= \sum_{i \in S} \lambda_i \langle \mathbf{y}, \mathbf{x}_i \rangle \\ &> 0. \end{aligned}$$

The last inequality is strict since $S \neq \emptyset$, $\lambda_i > 0$ and $\langle \mathbf{y}, \mathbf{x}_i \rangle > 0$.

3. Finally we move to Part (3) which is exactly the same statement as that of Lemma 8.6. Our new proof follows easily from Part (2) of the current lemma. Pick a vector $\mathbf{y}$ in general position, i.e., so that $\langle \mathbf{y}, \mathbf{x}_i \rangle \neq 0$ for any $i \in [m]$. At least half the vectors $\mathbf{x}_i$ must have a positive inner product with either $\mathbf{y}$ or $-y$. Assume without loss of generality that $\mathbf{x}_1, \ldots, \mathbf{x}_{\lceil m/2 \rceil}$ have a positive inner product with $\mathbf{y}$. Applying Part(2) to these vectors and $\mathbf{y}$, we get $\lceil m/2 \rceil \leq n$.

■

Exercise: (A) Give an example showing the result in Part (1) of Lemma 8.7 is tight. (B) Interpret this part as a coding-theoretic bound. (C) Give codes that make this interpretation tight.

The exercise above gives the natural motivation for Part (1) of the lemma. Part (3) was already motivated by the Plotkin bound. What motivates us to study Part (2). On the one hand it provides a simple proof of Part (3). But it actually turns out to be even more important on its own and we will use it several times in the rest of these notes.

8.4 The Elias-Bassalygo bound

We now return to the task of bounding the rate of a code, given its relative distance. The current picture of the upper bounds involves two incomparable bounds: the Hamming bound is stronger for smaller δ and the Plotkin bound is stronger for larger δ. Our next bound unifies the two techniques and thus gets a bound which is always stronger, though the bound is very close to the Hamming bound for small δ.

To motivate this bound, we introduce a new notion of error-correction. Later we will refer to this notion as that of "list-decoding". Currently, we will use a terminology that is more reminiscent of the notion of "t-error-correcting codes" of Hamming.

Definition 8.8 **((t, ℓ)-error-correcting code)** *A code $C \subseteq \Sigma^n$ is a (t, ℓ) error correcting code if for every received word $\mathbf{y} \in \Sigma^n$, the ball of radius t around $\mathbf{y}$, $B(\mathbf{y}, t)$, contains at most ℓ codewords of C.*

For a code C and integer ℓ, we refer to the largest t for which C is a (t, ℓ)-error-correcting code to be the list of ℓ error-correcting radius of C.

Recall that Hamming's notion of a t-error-correcting code becomes a $(t, 1)$-error-correcting code in this new definition. Let us take a peek back at Hamming's proof of the Hamming bound for binary codes. The crux of the proof was that the balls $B(\mathbf{c}, t)$ around the codewords of a t-error-correcting code are disjoint. Thus if the code has 2^k codewords we get $2^k \mathrm{Vol}(t, n) \leq 2^n$, where $\mathrm{Vol}(t, n) = |B(\mathbf{c}, t)|$. Note that we didn't say exactly this when we proved the Hamming bound. Instead we considered balls of radius $(d-1)/2$ around codewords, where d was the minimum distance, and implicitly used the fact that such a code is a $(d-1)/2$-error-correcting code. But when we generalize the Hamming bound it will be better to explicit with notion of t-error-correcting codes.

Proposition 8.9 *Suppose a $(n, k, d)_2$ code C is a (t, ℓ)-error-correcting code. Then $2^k \mathrm{Vol}(t, n) \leq \ell \cdot 2^n$.*

Proof The proposition follows easily. If we consider the balls of radius t around the codewords, then any word in $\{0, 1\}^n$ is considered at most ℓ times. Thus the sum of the volumes of these balls is at most $\ell \cdot 2^n$. ■

Of course, the proposition does not immediately translate into new asymptotic relationships between rate and relative minimum distance. To get such relationships we have to relate the minimum distance of a code to its list of ℓ-error-correcting radius for non-trivial values of ℓ. Any $\ell > 2$, but less than $2^{\epsilon n}$ would be of interest. We will study such a bound next. Such bounds are closely related to bounds studied by S. Johnson [52, 53] and are termed the Johnson bounds.

Theorem 8.10 (Johnson bound [52]) *Every $(n, k, \delta n)_2$ code is also a $(\tau n - 1, n)$-error-correcting code for $\tau = \frac{1}{2} \cdot (1 - \sqrt{1 - 2\delta})$.*

Proof As usual we turn the problem into a geometric one by using the embedding function Embed. Let $\mathbf{c}_1, \ldots, \mathbf{c}_m$ be codewords of a code of minimum distance $d = \delta n$ that are within a Hamming ball of radius $t = \tau n - 1$ from a received vector $\mathbf{b}$. We wish to show $m \leq n$.

We will embed the vectors into Euclidean space, scaling them by a factor of $1/\sqrt{n}$ to get vectors of unit norm. For $i \in [m]$, let $\mathbf{x}_i = \frac{1}{\sqrt{n}}\text{Embed}(\mathbf{c}_i)$ and let $\mathbf{y} = \frac{1}{\sqrt{n}}\text{Embed}(\mathbf{b})$. By the properties of the embedding function (Proposition 8.3), we get: (1) $\|\mathbf{x}_i\| = \|\mathbf{y}\| = 1$. (2) $\langle \mathbf{x}_i, \mathbf{x}_j \rangle \leq 1 - 2\delta$, if $i \neq j$. (3) $\langle \mathbf{y}, \mathbf{x}_i \rangle > 1 - 2\tau$. In other words, we have a collection of unit vectors $\mathbf{x}_i$ whose pairwise inner product is small, but which have a common, large inner product with $\mathbf{y}$. Notice the syntactic similarity to Part (2) of Lemma 8.7. In fact we will reduce our problem to exactly this case. How? We will just shift our origin to a new vector $\mathbf{v}$ so that from this vector, the vectors $\mathbf{x}_i$ mutually subtend an angle of at least $\pi/2$. And how do we find such a vector $\mathbf{v}$? Well the most natural idea is to shift the origin closer to the scene of action — namely, towards $\mathbf{y}$. Specifically, we will move to some point $\alpha\mathbf{y}$ and inspect our world from there. The following claim asserts that we will see what we hope to see.

Claim 8.11 *There exists an α such that for every $i \neq j \in [m]$, $\langle \mathbf{x}_i - \alpha\mathbf{y}, \mathbf{x}_j - \alpha\mathbf{y} \rangle \geq 0$, while for every i, $\langle \mathbf{x}_i - \alpha\mathbf{y}, \mathbf{y} - \alpha\mathbf{y} \rangle > 0$.*

Proof We will not specify α yet only that it will lie in the interval $0 \leq \alpha < 1$. For such α, note that

$$\langle \mathbf{x}_i - \alpha\mathbf{y}, \mathbf{x}_j - \alpha\mathbf{y} \rangle \leq 1 - 2\delta - 2\alpha(1 - 2\tau) + \alpha^2 = (1 - \alpha)^2 + 4\alpha\tau - 2\delta.$$

The right-hand side is minimized at $\alpha = 1 - 2\tau$. For this setting, the RHS above equals $4\tau - 4\tau^2 - 2\delta$. Recall we set $\tau = \frac{1}{2}(1 - \sqrt{1 - 2\delta})$, and so we have $(1 - 2\tau)^2 = 1 - 2\delta$, which in turn implies $4\tau - 4\tau^2 - 2\delta = 0$. We conclude that for this setting $\langle \mathbf{x}_i - \alpha\mathbf{y}, \mathbf{x}_j - \alpha\mathbf{y} \rangle \geq 0$, as desired.

To conclude, we note that for the same setting $\alpha = 1 - 2\tau$, we have

$$\langle \mathbf{x}_i - \alpha\mathbf{y}, (1 - \alpha)\mathbf{y} \rangle > (1 - \alpha)(1 - 2\tau) - (\alpha)(1 - \alpha) = 0,$$

which yields the other part of the claim. ■

We are now in a position to apply Lemma 8.7, Part (2), to the vectors $\{\mathbf{x}_i - \alpha\mathbf{y}\}_i$ and $\mathbf{y} - \alpha\mathbf{y}$ to conclude that $m \leq n$. This concludes the proof of the theorem. ■

Combining Proposition 8.9 with Theorem 8.10 gives us the Elias-Bassalygo upper bound on the rate of a family of codes with relative distance δ.

Theorem 8.12 (Elias-Bassalygo bound [12, 101]) *If $\mathcal{C}$ is an infinite family of binary codes with rate R and relative distance δ, then $R \leq 1 - H(\frac{1}{2} \cdot (1 - \sqrt{1 - 2\delta}))$.*

Proof The theorem follows essentially from Proposition 8.9 and Theorem 8.10. The missing ingredients are dry exercises showing that one can pick n large enough so as to overcome all problems posed by identities which hold only asymptotically. (The volume of Hamming balls is not exactly related to the binary entropy function; the Johnson bound only lower bounds the (t, ℓ)-error-correcting radius of codes when ℓ is a growing function of n. And the bound only allows a list-decoding radius of $\tau n - 1$ and not τn.) We'll spare the reader the details. ■

Before concluding the section, let us digress briefly to understand when and why the Elias-Bassalygo bound is better. To do so, let us recall two of the previous bounds that we have worked with. The Hamming upper bound says $R \leq 1 - H(\delta/2)$, while the GV lower bound says $R \geq 1 - H(\delta)$. The Elias-Bassalygo bound shows $R \leq 1 - H(\tau(\delta))$, for $\tau(\delta) = \frac{1}{2} \cdot (1 - \sqrt{1 - 2\delta})$. First note that $\delta/2 \leq \tau(\delta) \leq \delta$ and H_2 is monotone decreasing, and so the Elias-Bassalygo bound is always between the Hamming bound and the GV bound. Further if $\delta > 0$, then $\tau(\delta) > \delta/2$ and so the Elias-Bassalygo bound is strictly better than the Hamming bound. However if δ is close to zero then $\delta/2$ is a very good approximation to $\tau(\delta)$ and so for small values of δ the Elias-Bassalygo bound is not much better than the Hamming bound. However for large values of δ, $\tau(\delta)$ starts to get closer to δ. In particular, if $\delta = \frac{1}{2} - \epsilon$, then $\tau(\delta) = \frac{1}{2} - \sqrt{\epsilon/2}$ and so τ approaches $\frac{1}{2}$ as δ approaches $\frac{1}{2}$. So as $\delta \to \frac{1}{2}$, the Elias-Bassalygo bound really starts to get better and approaches the GV bound!

What else could we hope for? While the Elias-Bassalygo bound gives us the right bound for $\delta = \frac{1}{2}$, it does not quite have the right growth around this point. In particular, the GV bound shows that one can find codes of rate $O(\epsilon^2)$ and relative distance $\frac{1}{2} - \epsilon$, as $\epsilon \to 0$. The Elias-Bassalygo bound only rules out codes of rate $\Omega(\epsilon)$ at this distance. Which bound is closer to the truth? Turns out the GV bound is correct here, and the E-B bound is too weak. In the next lecture we will describe a different upper bound, called the Linear Programming (LP) bound, which ends up showing the tightness of the GV bound.

8.5 q-ary bounds

We now extend the results of earlier section to codes over general alphabets. We start with q-ary embeddings. The definition does not extend the previous definition, but in fact, gives an alternate embedding which works as well.

Fix an arbitrary bijection $\text{ind} : \mathbb{F}_q \to [q]$ be any bijection between $\mathbb{F}_q$ and $[q]$. For $1 \leq i \leq n$, Let $\mathbf{e}_{i,n}$ be the unit vector along the ith coordinate direction in $\mathbb{R}^n$. We now define our q-ary embeddings.

Definition 8.13 (Embedding q-ary space in Euclidean space) *The embedding function q-Embed : $\mathbb{F}_q \to \mathbb{R}^q$ is defined as follows:*

$$q\text{-}\mathit{Embed}(\alpha) = \mathbf{e}_{\text{ind}(\alpha),q}.$$

For $n \geq 1$, the n-dimensional embedding function extends the embedding above, with q-Embed : $\mathbb{F}_q^n \to \mathbb{R}^{qn}$ being given by

$$q\text{-}\mathit{Embed}(\langle \alpha_1, \ldots, \alpha_n \rangle) = \langle q\text{-}\mathit{Embed}(\alpha_1), \ldots, q\text{-}\mathit{Embed}(\alpha_n) \rangle.$$

Proposition 8.14 *For vectors $\alpha, \beta \in \mathbb{F}_q^n$, the embedding q-Embed satisfies:*

$$\|q\text{-}\mathit{Embed}(\alpha)\|^2 = n, \ \langle q\text{-}\mathit{Embed}(\alpha), q\text{-}\mathit{Embed}(\beta) \rangle = n - \Delta(\alpha, \beta).$$

It will be preferrable to index our qn-dimensional space by two indices i, j with $i \in [n]$ and $j \in [q]$. Let H_i denote the hyperlane in $\mathbb{R}^{qn}$ given by $\sum_{j=1}^{q} x_{ij} = 1$. Note that the q-ary embedding lies in the affine subspace $\mathcal{H}$ given by the intersection of the hyperplanes H_i, i.e., $\mathcal{H} = \cap_{i=1}^{n} H_i$. Let $\mathbf{Q}_n \in \mathbb{R}^{qn}$ be the vector $\langle \frac{1}{q}, \ldots, \frac{1}{q} \rangle$. Then Q_n also lies in $\mathcal{H}$ and will play the role of the origin in $\mathcal{H}$. We will use the following proposition to tighten our results.

Proposition 8.15 *The vectors $\{q\text{-Embed}(\mathbf{x}) - \mathbf{Q}_n | \mathbf{x} \in \mathbb{F}_q^n\}$ lie in an $(q-1)n$-dimensional vector space over $\mathbb{R}$.*

We are now ready to prove some bounds. We start with the q-ary Plotkin bound.

Theorem 8.16 (q-ary Plotkin bound) *If C is an $(n, k, d)_q$ code, then $k \leq n - \frac{q}{q-1} \cdot d + \log_q \left(\frac{q^2}{q-1} d \right)$.*

Proof It suffices to prove the theorem for $d \geq \frac{q-1}{q} n$. The remaining cases follow by the restriction argument. For the case $d \geq \frac{q}{q-1} n$, we need to show that the number of codewords is at most qn. (This bound is met by Reed-Muller codes of degree 1.) It would be slightly easier to prove a bound of $2(q-1)n$. This may satisfy mere mortals, but since we're superhuman, we'll prove the correct result.

Let C be an $(n, k, d)_q$ code with $d \geq \frac{q-1}{q} n$. Let α be the *least* commonly occurring symbol in the first coordinate among codewords of C. Let C' be the code obtained by throwing away from C all codewords that have an α in the first coordinate position. Note that $|C'| \geq \frac{q-1}{q} |C|$. Thus it will suffice to prove that $|C'| \leq (q-1)n$ (to get $|C| \leq qn$). We will do so below.

Let $\mathbf{c}_1, \ldots, \mathbf{c}_m$ be the codewords of C'. Let $\mathbf{x}_1, \ldots, \mathbf{x}_m \in \mathbb{R}^{qn}$ be the vectors given $\mathbf{x}_i = q\text{-Embed}(\mathbf{c}_i) - Q_n$. Let $\mathbf{y} = -(\langle q\text{-Embed}(\alpha), \mathbf{Q}_{n-1} \rangle - \mathbf{Q}_n)$. We will show below that the following are true: (1) $\langle \mathbf{x}_i, \mathbf{x}_j \rangle \leq 0$, (2) $\langle \mathbf{x}_i, \mathbf{y} \rangle > 0$, and (3) $\mathbf{x}_i, \mathbf{y}$'s are contained in a $(q-1)n$ dimensional real vector space. Applying Lemma 8.7 to these vectors, we get that $m \leq (q-1)n$, as desired. Thus it suffices to verify the three conditions above to prove the theorem. We do so below.

1. Note that

$$\begin{aligned} \langle \mathbf{x}_i, \mathbf{x}_j \rangle &= \langle q\text{-Embed}(\mathbf{c}_i) - \mathbf{Q}_n, q\text{-Embed}(\mathbf{c}_j) - \mathbf{Q}_n \rangle \\ &\leq n - \Delta(\mathbf{c}_i, \mathbf{c}_j) - \frac{n}{q} \\ &\leq 0. \end{aligned}$$

2. We need to show $\langle \mathbf{x}_i, \mathbf{y} \rangle > 0$. Since $\mathbf{y}$ is zero on all but the first q coordinates, it suffices to consider the contribution to the inner product from the first q terms. Let the first coordinate of $\mathbf{c}_i = \beta$. Then $\langle \mathbf{x}_i, \mathbf{y} \rangle = \langle q\text{-Embed}(\beta) - \mathbf{Q}_1, -q\text{-Embed}(\alpha) + \mathbf{Q}_1 \rangle$. Since $\alpha \neq \beta$, the first inner product is zero, while the others are $\frac{1}{q}$. Summing them all up, we get $\langle \mathbf{x}_i, \mathbf{y} \rangle = \frac{1}{q} > 0$.

3. The final part follows from Proposition 8.14 and the fact that $\mathbf{y}$ also lies on the intersection of hyperplanes in which each of the n blocks of q coordinates sum to zero.

■

We move on to the Johnson bound for q-ary codes.

Theorem 8.17 (q-ary Johnson bound) *Every $(n, k, \delta n)_q$ code is also a $(\tau n - 1, (q-1)n)$-error-correcting code for $\tau = \frac{q-1}{q} \cdot (1 - \sqrt{1 - \frac{q}{q-1}\delta})$.*

Proof Let $d = \delta n$ and $t = \tau n - 1$. Let C be an $(n, k, d) - q$ code. For a received vector $\mathbf{b}$, let $\mathbf{c}_1, \ldots, \mathbf{c}_m$ be codewords of C within a Hamming distance of t from $\mathbf{b}$. Define $\mathbf{x}_i = q\text{-Embed}(\mathbf{c}_i) - \mathbf{Q}_n$, for $i \in [m]$. Define $\mathbf{y} = q\text{-Embed}(\mathbf{b}) - \mathbf{Q}_n$. The vectors $\mathbf{x}_i$, for $i \in [m]$, are contained in a $(q-1)n$ dimensional subspace of $\mathbb{R}^{qn}$ with the property that their pairwise inner product is small, while each has a large inner product with some fixed vector $\mathbf{v}$. As in the proof of Theorem 8.10, we can conclude that under the setting $\tau = \frac{q-1}{q} \cdot (1 - \sqrt{1 - \frac{q}{q-1}\delta})$, we can find an α such that the vectors $\{\mathbf{x}_i - \alpha \mathbf{v}\}_i$, have a pairwise non-positive inner product, while their inner product with $\mathbf{v}$ is positive. Applying Part (2) of Lemma 8.7 (our favorite workhorse) we get $m \leq (q-1)n$. ■

The q-ary Elias-Bassalygo bound is now straightforward. We state it for completeness.

Theorem 8.18 (q-ary Elias-Bassalygo bound) *If $\mathcal{C}$ is a family of q-ary codes with rate R and relative distance δ, then*

$$R \leq 1 - H_q\left(\frac{q-1}{q}\left(1 - \sqrt{1 - \frac{q}{q-1}\delta}\right)\right).$$

Bibliographic Notes

The Plotkin bound was shown in [89], and the Johnson bound in [52, 53]. The Elias-Bassalygo bound was discovered independently by P. Elias and L. Bassalygo. Elias seemingly discovered the bound in the 1950s but never published his result — it just got integrated into the folklore of coding theory in the US. The first journal paper to mention Elias's proof seems to be a paper by Shannon, Gallager, and Berlekamp [101] in 1967. In the meanwhile, L.A. Bassalygo discovered the same bound in 1965 [12]. The Johnson bounds are from some intermediate period (between Elias's observation, and Bassalygo's publication). The proofs of the Johnson bound in this notes are not from the original papers, but rather from more recent work. The proofs over the binary alphabet are from Agrell, Vardy, and Zeger [1]. The q-ary version is from [47].

Chapter 9

6.897 Algorithmic Introduction to Coding Theory	October 10, 2001
Lecture 9	
Lecturer: Madhu Sudan	*Scribe: Nitin Thaper*

Today we will talk about:

- MacWilliams identities
- LP bound
- Some perspectives on asymptotics

9.1 MacWilliams Identities

Recall that our theme in the last lecture was to upper bounds on the rate of *any* family of codes given its relative distance. Bounds of this form, that hold for *every* code or family of codes, are termed universal bounds. Our first topic today are bounds of this form for all *linear* codes. While these are not directly formulated as upper bounds on the rate of codes, given their relative distance, these bounds do play an essential role in some of the strongest known bounds on rate. We will get to that part later, but first we introduce and prove the MacWilliams Identities.

9.1.1 Linear codes and their duals

Recall that a linear code C is specified by a $k \times n$ generator matrix $\mathbf{G}$. Alternatively, the code can be specified by a $n \times (n-k)$ parity check matrix $\mathbf{H}$. The matrix $\mathbf{G}$ consists of k linearly independent rows while $\mathbf{H}$ consists of $n-k$ linearly independent columns.

The dual of the linear code C, denoted $C^{\perp}$, is the the code *generated* by $\mathbf{H}^T$, the transpose of the parity check matrix of C. It is obvious that $C^{\perp}$ is also a linear code, with block length n and message length $n-k$. Furthermore every pair of codewords $\mathbf{b} \in C$ and $\mathbf{c} \in C^{\perp}$ satisfy $\langle \mathbf{b}, \mathbf{c} \rangle = 0$. A slightly stronger fact is proven in the proposition below.

Proposition 9.1 *For linear code $C \subseteq \mathbb{F}_q^n$ and vector $\mathbf{x} \in \mathbb{F}_q^n$, the following hold:*

- *If $\mathbf{x} \in C^\perp$, then for every $\mathbf{c} \in C$, $\langle \mathbf{c}, \mathbf{x} \rangle = 0$.*
- *If $\mathbf{x} \notin C^\perp$, then for every $\alpha, \beta \in \mathbb{F}_q$, the sets $\{\mathbf{c} \in C \mid \langle \mathbf{c}, \mathbf{x} \rangle = \alpha\}$ and $\{\mathbf{c} \in C \mid \langle \mathbf{c}, \mathbf{x} \rangle = \beta\}$ have the same cardinality.*

Proof The first part of the proposition follows from the definition of the dual of a code. For the second part note first that without loss of generality, we may assume $\beta = 0$ and $\alpha \neq 0$. Let $S_\alpha = \{\mathbf{c} \in C \mid \langle \mathbf{c}, \mathbf{x} \rangle = \alpha\}$, and $S_0 = \{\mathbf{c} \in C \mid \langle \mathbf{c}, \mathbf{x} \rangle = 0\}$. Note that $\mathbf{0} \in S_0$ and hence the latter is non-empty. We note next that the former is also non-empty. Since $\mathbf{x} \notin C^\perp$ we have that there exists $\mathbf{c} \in C$ such that $\langle \mathbf{c}, \mathbf{x} \rangle = \alpha' \neq 0$. Then we have that $\mathbf{b} = (\alpha')^{-1}\alpha\mathbf{c}$ is an element of C with $\langle \mathbf{b}, \mathbf{x} \rangle = \alpha$, and thus $\mathbf{b} \in S_\alpha$.

For a set $A \subseteq \mathbb{F}_q^n$ and vector $\mathbf{y} \in \mathbb{F}_q^n$, let $\mathbf{y} + A$ denote the set $\{\mathbf{y} + \mathbf{x} \mid \mathbf{x} \in A\}$. Fix $\mathbf{b} \in S_\alpha$. We get $|S_\alpha| = |S_0|$ from the following series of facts:

1. $\mathbf{b} + S_0 \subseteq S_\alpha$: If $\mathbf{a} \in S_0$ then $\langle \mathbf{b} + \mathbf{a}, \mathbf{x} \rangle = \langle \mathbf{b}, \mathbf{x} \rangle + \langle \mathbf{a}, \mathbf{x} \rangle = \alpha + 0 = \alpha$.
2. $|\mathbf{b} + S_0| = |S_0|$: Follows since $\mathbf{b} + \mathbf{a} = \mathbf{b} + \mathbf{a}'$ iff $\mathbf{a} = \mathbf{a}'$.
3. $|S_0| \leq |S_\alpha|$: Follows from Parts (1) and (2) above.
4. $|S_\alpha| \leq |S_0|$: Similar to Parts (1)-(3) above, except that here we work with the set $(-\mathbf{b}) + S_\alpha$.

■

9.1.2 Weight distributions and the weight generating function

Definition 9.2 (Weight distribution & generating function) *Given an $[n, k, d]_q$ code C, and index $i \in \{0, \ldots, n\}$, the ith weight enumerator of C is the number of codewords in C of Hamming weight exactly i. The weight distribution of C is the sequence $\langle A_0, \ldots, A_n \rangle$, where A_i is the ith weight enumerator of C. The weight generating function of C is the formal polynomial $A_C(x) = \sum_{i=0}^n A_i x^i$, where $\langle A_0, \ldots, A_n \rangle$ is the weight distribution of C.*

The MacWilliams Identities relate the weight distribution of a code C with that of the code $C^\perp$. We will do so by studying the *weight generating function*, $A_C(y) = \sum_{i=0} A_i y^i$, of the code C, and relating A_C to $A_{C^\perp}$. As is by now usual, we will prove the identity first for the binary case, and then move to the q-ary case later.

9.1.3 The extended generating function

We will start by defining an elaborate generating function of a code C, called its *extended generating function*. It will be quite evident that this generating function preserves all information about the code C (unlike the weight generating function which does not tell us exactly what the code is). We will then relate this elaborate generating function of the code to that of its dual.

We will introduce the extended generating function gently. We will start by defining the extended generating function of a single bit(!), and then define it for a word in $\{0,1\}^n$. and then define the extended generating function of a code.

Definition 9.3 (Extended generating function of a bit) *The weight generating function of C is the formal polynomial $A(x) = \sum_{i=0}^{n} A_i x^i$. For a bit $b \in \{0,1\}$, the extended generating function $W_b(x,y)$, is a polynomial in two variables x and y defined as:*

$$\begin{aligned} W_b(x,y) &= x \quad \text{if } b = 0 \\ &= y \quad \text{if } b = 1 \end{aligned}$$

Definition 9.4 (Extended generating function of a word) *For a vector $\mathbf{b} = \langle b_1, \ldots, b_n \rangle \in \{0,1\}^n$, the extended generating function $W_{\mathbf{b}}(\mathbf{x},\mathbf{y})$, is a polynomial in $2n$ variables, $\mathbf{x} = \langle x_1, \ldots, x_n \rangle$ and $\mathbf{y} = \langle y_1, \ldots, y_n \rangle$, defined as: $W_{\mathbf{b}}(\mathbf{x},\mathbf{x}) = \prod_{i=1}^{n} W_{b_i}(x_i, y_i)$.*

Finally we define the extended generating function of a code C.

Definition 9.5 (Extended generating function of a code) *For a code $C \subseteq \{0,1\}^n$, the extended generating function $W_C(\mathbf{x},\mathbf{y})$, is a polynomial in $2n$ variables, $\mathbf{x} = \langle x_1, \ldots, x_n \rangle$ and $\mathbf{y} = \langle y_1, \ldots, y_n \rangle$, defined as: $W_C(\mathbf{x},\mathbf{x}) = \sum_{\mathbf{b} \in C} W_{\mathbf{b}}(\mathbf{x},\mathbf{y})$.*

To make sense of the definitions above, it would help to see an example. Consider the code $C = \{000, 101, 110, 011\}$. The extended generating function for this code is:

$$W_C(\mathbf{x},\mathbf{y}) = x_1 x_2 x_3 + y_1 x_2 y_3 + y_1 y_2 x_3 + y_1 y_2 x_3 + x_1 y_2 y_3.$$

It should be immediately clear that the extended generating function carries all the information about a code and does not do so in any especially clever way. The extended generating function is not intended to be a way of compressing information about the code, but rather an elaborate way of representing the code, which will come in useful in studying the combinatorics of codes. At the outset we are not hoping to use them to glean any information about codes in a computationally efficient manner.

Given this elaborate representation of a code, it should not come as a surprise that given the extended generating function of a code C we can derive the extended generating function of its dual. After all. all the information of C is embedded into $W_C(\mathbf{x},\mathbf{y})$, so C can be derived from W_C, $C^\perp$ can be derived from C, and finally $W_{C^\perp}$ can be derived from $C^\perp$. The generalized MacWilliams Identity just gives an explicit form of this rather obvious statement. The strength of the result lies in the fact that the relationship takes an especially simple closed form - one that will turn out to be especially amenable to manipulations later on.

The crux of the identity is some sort of a "Discrete Fourier Transform" (or Hadamard transform or Walsh transform, depending on your loyalties). Instead of focusing on the function $W_C(\mathbf{x},\mathbf{y})$ we will focus on the function $W_C(\mathbf{x}+\mathbf{y},\mathbf{x}-\mathbf{y})$. Before saying what this transform does to the generating function of a code, we will describe what the transform does to the generating function of a single vector $\mathbf{b}$.

Lemma 9.6

$$W_{\mathbf{b}}(\mathbf{x}+\mathbf{y},\mathbf{x}-\mathbf{y}) = \sum_{\mathbf{c} \in \{0,1\}^n} (-1)^{\langle \mathbf{b},\mathbf{c} \rangle} W_{\mathbf{c}}(\mathbf{x},\mathbf{y}).$$

Proof Note by definition of $W_{\mathbf{b}}$ that

$$W_{\mathbf{b}}(\mathbf{x}+\mathbf{y},\mathbf{x}-\mathbf{y}) = \prod_{i=1}^{n} W_{b_i}(x_i+y_i, x_i-y_i) = \prod_{i=1}^{n} \left(x_i + (-1)^{b_i} y_i\right).$$

Expanding the right hand side above we get a sum of 2^n terms of the form $\pm z_1 \cdots z_n$, where $z_i \in \{x_i, y_i\}$ and the sign is odd iff there is an odd number of indices i such that where $z_i = y_i$ and $b_i = 1$. Letting $\mathbf{c} \in \{0,1\}^n$ denote the index of this sum of 2^n terms, and setting $z_i = y_i$ if $c_i = 1$, we see that $z_i = W_{c_i}(x_i, y_i)$ and the sign in front is odd iff $\langle \mathbf{b}, \mathbf{c} \rangle = 1$. Thus we have

$$\prod_{i=1}^{n} \left(x_i + (-1)^{b_i} y_i\right) = \sum_{\mathbf{c} \in \{0,1\}^n} (-1)^{\langle \mathbf{b},\mathbf{c} \rangle} \prod_{i=1}^{n} W_{c_i}(x_i, y_i) = \sum_{\mathbf{c} \in \{0,1\}^n} (-1)^{\langle \mathbf{b},\mathbf{c} \rangle} W_{\mathbf{c}}(\mathbf{x},\mathbf{y}).$$

This yields the lemma. ■

Theorem 9.7 (Generalized MacWilliams Identity [75]) *The extended generating function of a code C and its dual $C^{\perp}$ satisfy the identity:*

$$W_{C^{\perp}}(\mathbf{x},\mathbf{y}) = \frac{1}{|C|} W_C(\mathbf{x}+\mathbf{y},\mathbf{x}-\mathbf{y}).$$

Proof The proof follows easily from Proposition 9.1 and Lemma 9.6. Note that

$$\begin{aligned}
W_C(\mathbf{x}+\mathbf{y},\mathbf{x}-\mathbf{y}) &= \sum_{\mathbf{b} \in C} W_{\mathbf{b}}(\mathbf{x}+\mathbf{y},\mathbf{x}-\mathbf{y}) \\
&= \sum_{\mathbf{b} \in C} \sum_{\mathbf{c} \in \{0,1\}^n} (-1)^{\langle \mathbf{b},\mathbf{c} \rangle} W_{\mathbf{c}}(\mathbf{x},\mathbf{y}) \quad \text{(By Lemma 9.6)} \\
&= \sum_{\mathbf{b} \in C} \sum_{\mathbf{c} \in C^{\perp}} W_{\mathbf{c}}(\mathbf{x},\mathbf{y}) + \sum_{\mathbf{b} \in C} \sum_{\mathbf{c} \notin C^{\perp}} (-1)^{\langle \mathbf{b},\mathbf{c} \rangle} W_{\mathbf{c}}(\mathbf{x},\mathbf{y}) \\
&= |C| W_C(\mathbf{x},\mathbf{y}) + 0,
\end{aligned}$$

where the first part of the last equation is by definition of the extended generating function and the second part applies Proposition 9.1 to every $\mathbf{c} \notin C^{\perp}$. The theorem follows. ■

9.1.4 The MacWilliams Identities

We now return to the goal of studying the weight distribution of a code C. The following, simple proposition shows that we have made some progress already!

Proposition 9.8 *For every linear code C, we have $x^n A_C(y/x) = W_C(x, \ldots, x, y, \ldots, y)$.*

Proof The proposal follows easily by inspecting the right hand side. Under the substitution $x_i = x$ and $y_i = y$, we have

$$\begin{aligned}
W_C(x, \ldots, x, y, \ldots, y) &= \sum_{\mathbf{b} \in C} x^{n-\mathrm{wt}(\mathbf{b})} y^{\mathrm{wt}(\mathbf{b})} \\
&= x^n \sum_{\mathbf{b} \in C} (y/x)^{\mathrm{wt}(\mathbf{b})} \\
&= x^n \sum_{i=0}^{n} \sum_{\mathbf{b} \in C \ \mid \ \mathrm{wt}(\mathbf{b})=i} (y/x)^i \\
&= x^n \sum_{i=0}^{n} A_i (y/x)^i \\
&= x^n A_C(y/x).
\end{aligned}$$

■

The MacWilliams Identity now follows easily:

Theorem 9.9 (MacWilliams Identity [75])

$$A_{C^\perp}(y) = \frac{1}{|C|}(1+y)^n A_C\left(\frac{1-y}{1+y}\right).$$

Proof The proposition we just proved, Proposition 9.8, tells us that $A_{C^\perp}(y) = 1^n W_{C^\perp}(1, \ldots, 1, y, \ldots, y)$. Now, applying the generalized MacWilliams identity (Theorem 9.7), we get

$$W_{C^\perp}(1, \ldots, 1, y, \ldots, y) = \frac{1}{|C|} W_C(1+y, \ldots, 1+y, 1-y, \ldots, 1-y).$$

Finally applying Proposition 9.8 again, we have

$$W_C)(1+y, \ldots, 1+y, 1-y, \ldots, 1-y) = (1+y)^n A_C((1-y)/(1+y)).$$

Putting the above together, we get the theorem. ■

In the appendix to this lecture, we extend the bound above to the q-ary case. We state the q-ary version so that we can discuss the more general result and its implications.

Theorem 9.10 (q-ary MacWilliams Identity) *For a q-ary linear code C of block length n, we have*

$$A_{C^\perp}(y) = \frac{(1+(q-1)y)^n}{|C|} A_C\left(\frac{1-y}{1+(q-1)y}\right).$$

The MacWilliams identity shows that the weight distribution of a linear code can be computed from the weight distribution of its dual! Note, this is totally non-trivial - we can see no obvious reason why specifying the weight distribution of C, should fix the weight distribution of $C^\perp$. In a sense the identity suggests that any two linear codes of a given weight distribution are essentially the same, motivating the following question (the answer to which is not obvious to Madhu).

Question: **True or False:** For any pair of q-ary codes C_1 and C_2 of block length n that have the same weight distribution, there exists a permutation $\pi : [n] \to [n]$ such that $\mathbf{c} = \langle c_1, \ldots, c_n \rangle \in C_1$ iff such that $\mathbf{c}' = \langle c_{\pi(i)}, \ldots, c_{\pi(n)} \rangle \in C_2$.

We now examine the exact form of the relationship obtained between the weight distribution of a code and its dual. As in the case of the relationship between the extended (elaborate?) generating functions of a code and its dual, the exact form of the relationship turns out to be simple and quite useful. In particular, the weight distribution of the dual code is just a linear function of the weight distribution of the primal code, as we note below.

Corollary 9.11 *For every q and n there exists an $(n+1) \times (n+1)$ rational matrix $\mathbf{M}$ such that the following holds: Let $\mathbf{a} = \langle A_0, \ldots, A_n \rangle$ be the weight distribution of a q-ary linear code C of block length n, and let $\mathbf{b} = \langle B_0, \ldots, B_n \rangle$ be the weight distribution of $C^\perp$. Then*

$$\mathbf{b} = \frac{1}{A_0 + \cdots + A_n} \mathbf{aM}.$$

Furthermore, the matrix $\mathbf{M} = \{m_{ij}\}$ is given by $m_{ij} = \sum_{\ell=0}^{i} \binom{n-j}{i-\ell}\binom{j}{\ell}(-1)^\ell (q-1)^{i-\ell}$.

Proof Since B_i is the coefficient of y^i in $A_{C^\perp}(y)$, we need to examine the coefficient of y^i in $\frac{(1+(q-1)y)^n}{|C|} A_C\left(\frac{1-y}{1+(q-1)y}\right)$. Write this quantity as $\frac{1}{|C|}\sum j = 0^n A_j (1+(q-1)y)^{n-j}(1-y)^j$. The coefficient of y^i in this sum is obtained as the coefficient of y^ℓ in the expansion of $(1-y)^j$ times the coefficient of $y^{i-\ell}$ in the expansion of $(1+(q-1)y)^{n-j}$, summed up over $\ell \in [i]$. We thus get:

$$\begin{aligned} B_i &= \frac{1}{|C|}\sum_{j=0}^{n} A_j \sum_{\ell=0}^{i} \binom{n-j}{i-\ell}\binom{j}{\ell}(q-1)^{i-\ell}(-1)^\ell \\ &= \frac{1}{|C|}\sum_{j=0}^{n} A_j m_{ij} \end{aligned}$$

The corollary follows from the fact that $|C| = A_0 + \cdots + A_n$. ■

In the rest of this lecture we will describe (without proof) two consequences of this explicit form of the MacWilliams Identity.

9.2 Weight distribution of MDS codes

Recall the notion of an MDS (Maximum Distance Separable) code. These code are codes that meet the Singleton bound, i.e., have a minimum distance equal to $n - k + 1$, where n is the block length and k is the message length of the code. In this section, we will describe the weight distribution function of linear MDS codes exactly, for every n and k! This ought to be surprising, since this implies that all MDS codes have the same weight distribution and we have no prior reason to believe so! The first step in our proof is a simple but surprising fact.

Proposition 9.12 *If C is a linear MDS code, then so is its dual $C^\perp$.*

Proof The proof is a slight variation of the proof of the Singleton bound. Let C be an $[n, k, n-k+1]$-code. We know that $C^{\perp}$ is an $[n, n-k, d]$-code for some $d \leq k+1$ (using the Singleton bound), and we need to show $d \geq k+1$. Assume otherwise, and suppose there exists a vector $\mathbf{c} \in C^{\perp}$ with $\mathrm{wt}(\mathbf{c}) \leq k$. Without loss of generality assume that the first $i \leq k$ coordinates of $\mathbf{c}$ are non-zero and the remaining coordinates are zero. Then for every codeword $\mathbf{b} \in C$, we have $\sum_{j=1}^{k} b_j c_j = 0$. Project the codewords of C to their first k coordinates. We claim that the projection of C is a subspace of F_q^k of dimension $k-1$ or less (since $\sum_{j=1}^{k} b_j c_j = 0$ for all vectors in the projection). In such a case there exist two vectors $\mathbf{c}_1$ and $\mathbf{c}_2$ in C, whose projection on to the first k coordinates is the same. But then $\Delta(\mathbf{c}_1, \mathbf{c}_2) \leq n-k$ contradicting the hypothesis that C has minimum distance $n-k+1$. ■

Combining the proposition above with MacWilliams Identities, one gets the following surprising result — explicit expressions for the weight distribution for all MDS codes.

Theorem 9.13 *If C is an $[n, k, n-k+1]_q$-MDS code, then the weight distribution $\langle A_0, \ldots, A_n \rangle$ of C is given by $A_i = \binom{n}{i} \sum_{j=0}^{i+k-n} (-1)^j \binom{i}{j} ((q^{i+k-n-j} - 1)$.*

We won't prove the theorem here — but lets see why it should not be too surprising if we put together what we know. Consider the $2n+2$ variables corresponding to the weight distribution of C and $C^{\perp}$. The MacWilliams Identities gives us $n+1$ linear conditions relating these variables once the dimension of the primal code is fixed. Using these conditions in the obvious way, one can see that one can compute the dual weights given the primal distribution. By duality, one can also compute the primal weights given the dual weight distribution. Presumably there are other subsets of $(n+1)$ of the weights that would lead to the rest being specified as well! In the case of MDS codes we know $A_0 = B_0 = 1$ and $A_1 = \cdots = A_{n-k} = B_1 = \cdots = B_k = 0$ - which turns to be $n+2$ variables, that fortunately are of full rank. So the linear system can be solved and yields the theorem above!

Exercise: Determine which subsets of the $2n+2$ variables in the MacWilliams Identities are linearly independent. (Answer not known to Madhu.)

Notice a simple corollary to the above theorem: There are exactly A_i polynomials in $\mathbb{F}_q[x]$ of degree $< k$ that have exactly $n-i$ zeroes in any given subset $S \subseteq \mathbb{F}_q$ of cardinality n.

On the one hand the theorem above is quite impressive in that it gives exact numbers. However since the final expressions are not closed forms it is hard to get a sense of the growth of the expressions. A coarse approximation that works out reasonably is $A_i = \Theta(\binom{n}{i} q^{k+i-n})$ for MDS codes.

9.3 The Linear Programming bound

We now describe the most powerful application of the MacWilliams Identities, namely the Linear Programming (LP) bound on the rate of a code.

Let C be a $[n, ?, d]_q$ code with weight distribution $A_0, A_1, \ldots, A_n$. The number of codewords for the code is $\sum_i A_i$ and hence the rate $R = \log_q(\sum A_i)/n$. As mentioned earlier our goal is to get an asymptotic upper bound on the rate R. This reduces to deriving an upper bound on $\sum A_i$ subject to the restrictions that

$$A_0 = 1, A_1 = 0, A_2 = 0, \ldots, A_{d-1} = 0$$

$$B_0 = 1, B_1, B_2, \ldots, B_n \geq 0$$

As noted earlier, B_i's are essentially linear functions of A_i's. Precisely, $B_i \cdot \left(\sum_{j=0}^{n} A_j\right)$ is given by a linear function of the A_i's.

This motivates the following linear program:

$$\begin{array}{ll} \text{Maximize} & K_n = \sum_{i=0}^{n} A_i \\ \text{subject to} & \\ & \mathbf{aM} \geq \mathbf{0} \\ & A_0 = 1 \\ & A_1 = \cdots = A_{d-1} = 0 \\ & A_d, \ldots, A_n \geq 0 \end{array}$$

where $\mathbf{a} = \langle A_0, \ldots, A_n \rangle$, $\mathbf{M}$ is the matrix described in Corollary 9.11, and the notation $\mathbf{x} \geq \mathbf{0}$, implies every coordinate of $\mathbf{x}$ is non-negative.

The linear program above bounds the number of codewords that any linear code of distance d can have over a q-ary alphabet, and one can hope that linear programming and duality could help anaylze the quantity K_n above. The quantity $R_{\text{LP}} \stackrel{\text{def}}{=} \log_q(K_n)/n$ is called the LP (upper) bound on the rate of a linear code. Somewhat surprisingly, even though the linear program was motivated only for linear codes, the LP bound holds also for non-linear codes, as implied by the following theorem.

Theorem 9.14 *If C is an $(n, k, d)_q$ code with $A_i \stackrel{\text{def}}{=} \frac{1}{|C|} \sum_{\mathbf{c} \in C} |\{\mathbf{b} \in C \mid \Delta(\mathbf{b}, \mathbf{c}) = i\}|$. Let $\mathbf{a} = \langle A_0, \ldots, A_n \rangle$ and let $\mathbf{M}$ be as in Corollary 9.11. Then $\mathbf{aM} \geq \mathbf{0}$.*

Thus the quantity R_{LP} is the LP bound on the rate of *all* codes. The LP upper bound was discovered by Delsarte [27] who showed how to derive several known upper bounds using this framework. Performing a tight asymptotic analysis of the LP bound is non-trivial and took a while after the work of Delsarte. The best asymptotic analysis of the LP bound is due to McEliece, Rodemich, Rumsey and Welch [80] who give two upper bounds on the LP bound. We state their bounds for the binary case:

Theorem 9.15 (MRRW bound [80]) *If $\mathcal{C}$ is a family of binary codes (not necessarily linear) of rate R and relative distance δ then*

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

$$\textit{and} \quad R \leq \min_{u \in [0, 1-2\delta]} \{1 + g(u^2) - g(u^2 + 2\delta u + 2\delta)\} \quad \textit{where} \quad g(x) = H_2\left(\frac{1 - \sqrt{1-x}}{2}\right).$$

Proving these bounds is way out of scope of this lecture(r)! The interested reader is pointed to thesis of Samorodnitsky [94], or the article by Levenshtein [67] for further details.

9.4 Perspectives on Asymptotics

We earlier saw the Gilbert-Varshanov bound, namely $R(\delta, q) \geq 1 - H_q(\delta)$ and saw that codes based on algebraic geometry out-performed this bound. However for the binary alphabet this is the best known lower bound on the rate of a code. Natural questions to ask at this stage are:

1. Is the GV bound tight (or are there codes that do better than the GV bound)?
2. Is the LP bound tight (or do there exist codes that meet the LP bound)?
3. Is the MRRW bound tight (or do there exist codes that meet the MRRW bound)?

Both the GV bound and MRRW bound are known functions and we know that they are not equal. So it is not possible to believe that both bounds are tight. However till recently none of the remaining possibilities even considered pairwize could be ruled out. I.e., it was conceivable that the LP bound and GV bound were both tight, or that the LP bound and MRRW bound were both tight. Recently Samorodnitsky [94] managed to rule out one of these possibilities — he showed that for every δ, the LP bound is at least the arithmetic mean of the GV bound and the MRRW bound. So if the GV bound is tight, then the LP bound can not prove this fact!

Is the GV bound tight for binary codes? Obviously this question can not be answered at the moment. Yet one could ask what the evidence so far suggests. On the one hand no better code has been found, and we could take this as evidence that the GV bound is tight. On the other hand we could ask if there are examples that suggest, without proving, that the GV bound does not appear to be tight. We have already seen one such example — namely the Reed-Solomon codes were doing "better than random" but were using alphabet sizes that were growing with the block length. Turning this intuition into a refutation of the q-ary GV-bound required all the heavy-duty machinery of AG-codes, but it did pay off. Two significant lessons could be learned from this success — (1) we need some example of an anamolous behavior as exhibited by the RS code, and (2) we need an asymptotic direction to exploit in our investigation. Are there other examples of anamolous behaviour? Turns out every example we have seen shows some anamoly.

Hadamard codes If the best we could do with code is to assume that the spheres of radius d around them do not overlap, then a binary code with relative distance $n/2$ should have only two codewords. The Hadamard codes with $\Omega(n)$ codewords are doing significantly better.

Hamming codes & BCH codes If we fix a distance d and consider $n-k$ for $[n,k,d]$-codes then the GV bound requires $n-k \geq d \log n$, while the BCH and Hamming codes only need $n-k \approx (d/2)\log n$. Thus once again, these codes outperform the GV principle.

So the next question ought to be: Is there an asymptotic sense in which one should try to exploit these examples. We describe two approaches below.

Relative Distance close to $\frac{1}{2}$ The MRRW bound, together with the GV bound, implies that the binary code of relative distance $\frac{1}{2}-\epsilon$ with highest possible rate has rate $\Theta(\epsilon^2)$. So in a certain asymptotic sense, the MRRW bound and the GV bound are already tight at this extreme. However the GV bound is non-constructive and constructive results are still lagging behind the GV/MRRW bound in this case. The best known constructive results yield codes of rate $O(\epsilon^3)$ with distance $\frac{1}{2}-\epsilon$. Some constructions that achieve this bound are:

- Reed-Solomon code concatenated with random linear code. (The reader should work this out as an exercise.)
- Algebraic Geometry code concatenated with Hadamard code. Such a construction was given by Katsman, Tsfasman and Vladuts [60] and could be attempted by the reader as an exercise.

- Codes derived from expander graphs due to Alon, Bruck, Naor, Naor, and Roth [4].

Improving these constructions would be a significant step towards achieving the GV bound constructively.

Rate close to 1 At rate close to 1, or as $\delta \to 0$, the GV bound says that $R \geq 1 - \delta \log(1/\delta)$. All bounds upto the Elias bound only show $R \geq 1 - (\delta/2)\log(2/\delta)$. In this regime the gap between the GV bound and the upper bounds seems maximal. This regime may offer one of the best options for "beating the GV bound" if this is possible at all.

Chapter 10

6.897 Algorithmic Introduction to Coding Theory October 22, 2001

Lecture 10

Lecturer: Madhu Sudan *Scribe: Aram Harrow*

Today we move on to the second phase of our course: Algorithms in Coding theory. We will introduce some of the algorithmic tasks associated with coding theory. We'll then move on to a specific algorithm for decoding Reed-Solomon codes.

10.1 Algorithmic Tasks

10.1.1 Algorithmic tasks from Shannon's theory

Shannon's theorem already introduced specified two basic algorithmic tasks, namely *Encoding* and *Decoding*. The final objective is only defined as a combination of the two, given a channel that fixes the error model. Specifically the Shannon algorithmic challenge, restricted to the binary symmetric channel is the following:

> **The Shannon Problem:** Let $D_{p,n}$ be the distribution on n independently chosen random bits where each bit is 1 with probability p and 0 with probability $1-p$. Given $R < 1 - H(p)$, find an *efficiently computable* family of functions $\{E_n\}_n$ and $\{D_n\}_n$ where $E_n : \{0,1\}^{Rn} \to \{0,1\}^n$ and $D_n : \{0,1\}^n \to \{0,1\}^{Rn}$ so as to maximize the error exponent, i.e.,
>
> $$\lim_{n\to\infty} \left\{ -\frac{1}{n} \log \left(\Pr_{m \leftarrow U_{Rn}, \eta \leftarrow D_n} [D_n(E_n(m) + \eta) = m] \right) \right\}.$$

In the above challenge one may plug in any notion of efficiency one feels comfortable with. We will use the notion of "polynomial time computability" for starters. Later, we will switch to more efficient notions such as linear time computability.

Unfortunately this challenge is too complex to handle all at once. The Hamming setup provides a nice modular breakdown of the task. To motivate this, suppose we introduced the Hamming version of the problem, where an *adversary* is allowed to choose the message and introduce errors, up to a specified limit t. Then the above problem transforms to:

The Hamming Problem: Given $R > 0$ find the largest τ and an efficiently computable family of functions $\{E_n\}_n$ and $\{D_n\}_n$ where $E_n : \{0,1\}^{Rn} \to \{0,1\}^n$ and $D_n : \{0,1\}^n \to \{0,1\}^{Rn}$ such that:

$$\forall \eta \in B(\mathbf{0}, \tau n), \quad D_n(E_n(m) + \eta) = m.$$

In this setup one can decouple, at least partially, two steps in the goal: First the image of the encoding function better be a good error-correcting code. Next, this error-correcting code better have a good decoding algorithm. Thus by fixing the intermediate point — namely, the error correcting code, we get two independent problems, whose goals can be decoupled. Still the task of defining the problems precisely is riddled with subtleties. So we will do so slowly.

10.1.2 Encoding

What could be be subtle about the following question? "Given a family of codes $\mathcal{C} = \{C_n\}_n$ with C_n being an $(n, k, d)_q$ code, find an efficient family of algorithms $E_n : \Sigma^k \to \Sigma^n$ so that the image of E_n equals C_n." This definition works for most purposes, and we will be happy with it. However it will be good to recognize that there are issues here that might involve choices.

The main issue, stated in "data structural language", is the distinction between *preprocessing* and *querying*. Preprocessing is the work we do during the design phase of the code. Query processing is the work we do to encode a given message, with all the help that the preprocessing stage may have given.

The goals of preprocessing are often misunderstood. Unless one pins down the family of codes quite succinctly *efficiency* with respect to this task may not make a lot of sense. (An example of a question that does not make sense given what we know is: "How fast can the generator matrix of an algebraic-geometry code be constructed?" We know some specific families for which this question is well-defined — but in general, how do we even represent the algebraic geometry code whose generator matrix is to be constructed.)

The issue that does make sense to focus on, is to ask that the output of the preprocessing stage be succinct. If the code is linear, then a choice for the output is obvious: Output the generator matrix (or almost equivalently, the parity check matrix). But a more satisfactory answer, applicable to all codes and not just the linear ones, is to output a circuit describing the query processing function totally constructively. (Such a circuit would be composed of binary logic gates hooked up as an acyclic digraph with k input wires and n output wires. Computing the output of the circuit, given a fixed input takes time linear in the size of the circuit.)

What are the right goals for the "query processing" phase? The goal is fairly obvious here: We would like the encoding of the message to be performed as efficiently as possible. For linear codes, if we force the preprocessing phase to output a generator matrix, then the query processing is just a matrix-vector product and can be computed in polynomial (specifically $O(n^2)$) time. Later in the course, we will focus on the task of doing this more efficiently — but for now, we will consider this good enough. On the other hand, if the preprocessing phase outputs an encoding circuit, then the size of the output of the first phase *is* the running time of the query processing phase.

We conclude with the following two algorithmic tasks:

Preprocessing Problem: For a fixed family of codes $\mathcal{C}$, given an index i, compute an encoding circuit (or the generator matrix) for the ith code of the family.

A weak requirement is that the above produce an output that has size that is a polynomial in n_i — the block length of the ith code. A stronger requirement is that the above process take time polynomial in n_i. The good news here is that for linear codes the weak requirement can always be fulfilled.

The second algorithmic task, corresponding to the "query phase" is:

Encoding Problem: Given an encoding circuit E (or the generator matrix) for the ith member of a family of codes $\mathcal{C}$, and a message $\mathbf{m}$, compute its encoding.

Of course, in this sense, the encoding problem is trivial to solve in linear time in the size of E.

As mentioned earlier, we won't dwell on the subtleties introduced in this section during this course. But they ought to be kept in mind, when designing and evaluating the "utility" of new codes.

10.1.3 Decoding

Informally, the problem to be formalized is: Given a corrupted version $\mathbf{r} \in \mathbb{F}_q^n$ of an encoding of message $\mathbf{m}$, compute $\mathbf{m}$. However the problem is not well-posed yet, since we haven't quite given a formal definition of $\mathbf{m}$. Another issue is the specification of the encoding function E. When is this specified? And how much time are we given to design the decoding algorithm?

For the first issue: How is $\mathbf{m}$ defined as a function of $\mathbf{r}$, the natural definition based on the Shannon challenge described above, is the *maximum likelihood vector*, i.e., the vector $\mathbf{m}$ for which the probability that the received vector is $\mathbf{r}$ is largest, given the probabilistic channel. This motivates the following problem:

Maximum Likelihood Decoding: Given a channel Channel corrupting strings in $\mathbb{F}_q^n \to \mathbb{F}_q^n$, an encoding function $E : \mathbb{F}_q^k \to \mathbb{F}_q^n$, and a received vector $\mathbf{r} \in \mathbb{F}_q^n$ find $\mathbf{m} \in \mathbb{F}_q^k$ that maximizes $\Pr_{\text{Channel}}[\mathbf{r} = \text{Channel}(E(\mathbf{m}))]$.

When the channel noise model is that of the q-ary symmetric channel, then this question simplifies to the Hamming problem below:

Nearest Codeword Problem: Given an encoding function $E : \mathbb{F}_q^k \to \mathbb{F}_q^n$, and a received vector $\mathbf{r} \in \mathbb{F}_q^n$ find $\mathbf{m} \in \mathbb{F}_q^k$ that minimizes $\Delta(\mathbf{r}, E(\mathbf{m}))$.

The problems above are both quite hard. On the one hand, they don't allow the decoding algorithm to preprocess the code to be decoded. Further, they ask for error-correction possibly well beyond the distance of the code. So it should not be a surprise that versions of these problems become NP-hard. In particular, Berlekamp, McEliece, and van Tilborg [18] showed that the *Nearest Codeword Problem* above is NP-hard, even when the encoding functions are restricted to be linear functions given by their generator matrix. Later, Bruck and Naor [24] showed that one could even fix the family of codes, and thus the decoding algorithm could be allowed to preprocess the code, and the problem remains hard (unless NP has polynomial sized circuits).

Yet neither of these results are sufficiently negative to put a damper on the goal of meeting Shannon's challenge algorithmically. First - they only rule out decoding for some codes (and the codes are not particularly nice). Furthermore, they only rule out decoding these codes when the number of errors is much more than the distance of the code. To get a sense of the positive results in coding theory, we should look at some of the more reasonable decoding problems.

Reasonable decoding problems

The main restrictions that allow for algorithmic results are the following:

- We should only try to decode some fixed, well-known code (where we know the minimum distance etc.).
- We should place limits on the number of errors, and only expect to correct a number of errors in proportion to the distance.

The following three questions are natural questions along this line:

> **Unambiguous decoding:** For a fixed family of codes C, given an index i and a vector $\mathbf{r} \in \mathbb{F}_q^n$ find a codeword $\mathbf{c} \in C_i$, such that $\Delta(\mathbf{c}, \mathbf{r}) < \Delta(C_i)/2$ if such a codeword exists.

Note that the underlying assumption is that $\Delta(C_i)$ is known, or can be computed efficiently. Further, the question is posed so that the answer is unique and somehow reasonable to expect to compute, given the distance of the code. Actually the question remains reasonable for slightly larger values of the error, say up to the minimum distance of the code, though in such cases the answer is no longer unique or unambiguous. Here we can pose two slightly different questions. We call the first of these the "bounded distance decoding" problem, though this term is often used to allude to slightly related questions in the literature (and sometimes this is the *unambiguous decoding problem*).

> **Bounded distance decoding:** For a fixed family of codes C, and error function t, given an index i and a vector $\mathbf{r} \in \mathbb{F}_q^n$ and find *any* codeword $\mathbf{c} \in C_i$, such that $\Delta(\mathbf{c}, \mathbf{r}) \leq t(i)$ if such a codeword exists.

The bounded distance decoding problem is solvable for some codes for some functions $t(i)$ that are noticaebly larger than $\Delta(C_i)/2$. This is done by solving a slightly harder problem called the "list-decoding" problem described below. In principle, we would like to solve this problem for $t(i) = \Delta(C_i)$, but we know of no interesting codes where we can do this.

> **List decoding:** For a fixed family of codes C, and error function t, given an index i and a vector $\mathbf{r} \in \mathbb{F}_q^n$ and find a list of *all* codewords $\mathbf{c} \in C_i$, such that $\Delta(\mathbf{c}, \mathbf{r}) \leq t(i)$.

Both the bounded distance decoding problem and the list decoding problem are really families of problems (even once we fix the code $\mathcal{C}$) parametrized by the error function $t(\cdot)$. The larger this function for which an algorithm works, the better the algorithm.

In the sequel we will give an unambiguous decoding algorithm for Reed Solomon codes. But first we solve a problem that was too trivial to even pose above!

10.2 Erasure-decoding problem

Before we move on to the issue of handling actual *errors*, lets resolve one simple problem — that of decoding from erasures, at least in the case of linear codes. The erasure decoding problem is the following:

> **Erasure-decoding problem:**
> GIVEN: An $k \times n$ generator matrix $\mathbf{G}$ for a linear code C over $\mathbb{F}_q$ and a vector $\mathbf{r} \in (\mathbb{F}_q \cup \{?\})^n$, where ? is a special symbol denoting an erasure.
> FIND: A (or all) codeword $\mathbf{c} \in C$ satisfying $c_i = r_i$ for all $i \in [n]$ such that $r_i \neq ?$.

The algorithm for this part is quite simple. Suppose s symbols in $\mathbf{r}$ are erased. Let $\mathbf{r}'$ be the $(n-s)$-dimensional vector obtained by projecting $\mathbf{r}$ onto the non-erased symbols and let $\mathbf{G}'$ be the matrix obtained by projecting $\mathbf{G}$ onto columns corresponding to non-erasures. Let $\mathbf{m}$ be any solution to $\mathbf{mG}' = \mathbf{r}'$ and let $\mathbf{c} = \mathbf{mG}$. Then $\mathbf{c}$ is a solution to the erasure decoding problem.

When is this solution unique? We can't really look at the linear system and try to argue it can't have multiple solutions. So, we will use the properties of the code. Note that that if the code has minimum distance d, then $d-1$ erasures still allow for recovery of the codeword, information theoretically, since $d-1$ errors are detectable. In our case, this implies that if $s < d$, then the solution is unique.

Even if the solution is not unique, we don't have to give up hope. Since the solution to the erasure-decoding problem is obtained by solving some linear systems, we can actually enumerate all solutions to the erasure decoding problem. The set of solutions take the form $\{\mathbf{c} + \mathbf{bM} | \mathbf{b} \in \mathbb{F}_q^t\}$ for some non-negative t, and some vector $\mathbf{c}$ and some $t \times n$ matrix $\mathbf{M}$. Furthermore the vector $\mathbf{c}$ and matrix $\mathbf{M}$ can be found efficiently. Thus the erasure decoding problem for linear codes can be solved as satisfactorily as one could hope for!

10.3 Unambiguous decoding for Reed-Solomon codes

Let us recall the unambiguous decoding problem for Reed-Solomon codes.

> **Reed-Solomon decoding:**
> GIVEN: n distinct elements $x_1, \ldots, x_n \in \mathbb{F}_q$, $y_1, \ldots, y_n \in \mathbb{F}_q$ and parameters k and $t \leq \frac{n-k}{2}$.
> FIND: A polynomial $p \in \mathbb{F}_q[x]$ of degree less than k, such that $p(x_i) \neq y_i$ for at most t values of $i \in [n]$.

The problem is quite a non-trivial one. At first glance the best we can say about it is that it is in NP — i.e., if such a polynomial exists, then this fact can be verified efficiently, *given p*. However it is not even clear that it lies in co-NP, i.e. given a received vector $\mathbf{y}$, is there a short proof that no polynomial p of degree less than k such that $p(x_i) \neq y_i$ for at most t values of i. Yet a polynomial time solution can be found for this problem. This solution dates back to 1960 when Peterson [87]. came up with an algorithm essentially solving this problem. Strictly speaking, the solution actually applied only to binary BCH codes. The decoding for the Reed-Solomon case is from the extension of Gorenstein and Zierler [42] who generalized BCH codes to the non-binary case and observed that the algorithm generalizes, and also that Reed-Solomon codes are special cases of BCH codes (in this incestuous world).

The Peterson-Gorentstein-Zierler algorithm actually ran in time $O(n^3)$. Later Berlekamp [14] and Massey [79] speeded this algorithm so that it ran in $O(n^2)$ time. Currently the fastest versions run in time $O(n \text{poly} \log n)$ time [57]. We will not give these faster algorithms, but actually give a simple algorithm, running in $O(n^3)$ time, due to Welch and Berlekamp [122, 15]. The actual exposition we give is from [35] (see also [108, Appendix A]).

10.3.1 Error-locating polynomial

The crux of the decoding algorithm is an object called the error-locating polynomial. We define this polynomial below. We warn the reader that, at least from its definition, this polynomial is not necessarily easier to find than the solution polynomial p. However its properties end up helping us.

Definition 10.1 (Error locating polynomial) *Given a vector* $\mathbf{x}$ *and a recieved vector* $\mathbf{r}$ *that is within a distance of* t *from some polynomial* p *of degree less than* k*, a polynomial* $E(x)$ *is called an* error-locating polynomial *if it has degree* t *and satisfies* $E(x_i) = 0$ *if* $p(x_i) \neq y_i$*.*

Note that such a polynomial always does exist: Simply take a set $T \subseteq [n]$ that contains all the error locations and satisfies $|T| = t$ and let $E(x) = \prod_{i \in T}(x - x_i)$.

Next, we define one more polynomial, that is also as hard to find as p, or E. This is the polynomial $N(x) \stackrel{\text{def}}{=} p(x) \cdot E(x)$. While each of these polynomials E, p and N is hard to find, together they are easier to find! Below we list some properties of the triple of polynmials p, E and N.

$$\left.\begin{array}{l} E(x_i) = 0 \text{ if } y_i \neq p(x_i) \text{ and } \deg_x(E) = t \\ N(x) = p(x)E(x) \text{ and } \deg_x(N) < k + t \\ \forall i \in [n] \quad N(x_i) = p(x_i)E(x_i) = y_i E(x_i) \end{array}\right\} \tag{10.1}$$

The last equation above might require some explanation. Note that if $E(x_i) = 0$, then $p(x_i)E(x_i) = y_i E(x_i) = 0$. When $E(x_i) \neq 0$, we know $p(x_i) = y_i$ and so we still have $p(x_i)E(x_i) = y_i E(x_i)$.

Now how do we find any of the above? Turns out, if we just ignore all references to p in Equation (10.1) and just look for E and N satisfying the remaining conditions (and this suffices to pin them down essentially uniquely)! We describe this precisely next.

10.3.2 The Welch-Berlekamp algorithm

We start by describing the major steps in the Welch-Berlekamp algorithm. A priori, it may not be clear as to why this algorithm is either correct, or efficient. We will argue this later.

Welch-Berlekamp Algorithm:

Given: $n, k, t \leq \frac{n-k}{2}$ and n pairs $\{(x_i. y_i)\}_{i=1}^n$ with x_i's distinct.

Step 1: Find polynomials N and E satisfying the following, if they exist:

$$\left.\begin{array}{l} \deg_x(E) = t \\ \deg_x(N) < k + t \\ \forall i \in [n] \quad N(x_i) = y_i E(x_i) \end{array}\right\} \tag{10.2}$$

Step 2: Output $N(x)/E(x)$.

Efficiency

We first note that the algorithm above can be implemented to run in polynomial ($O(n^3)$) time. The second step is obvious, and all we need to argue this for is the first step. However the first step is essentially just solving a linear system. Specifically, we need to find coefficients $\langle N_0, \ldots, N_{k+t-1}\rangle$ and $\langle E_0, \ldots, E_t\rangle$ such that for all i,

$$\sum_{j=0}^{k+t-1} N_j x_i^j = y_i \sum_{j=0}^{t} E_j x_i^j$$

Note that for every i, the constraint above is just a linear constraint in the unknowns. So solving the system above, just amounts to solving a linear system. The only catch is that we need $E_t \neq 0$ and this is not a linear constraint. But notice we can actually force $E_t = 1$ since otherwise we can just divide all coefficients by E_t to get an alternate solution which satisfies the requirements. So, finding a solution to Step 1 just amounts to solving a linear system and hence can be done efficiently.

Correctness

Next we argue the correctness. For this part, assume p is a polynomial of degree $< k$ that agrees with the given set of points at all but t points. In such a case we will show that a pair of polynomials N and E satisfying (10.2) do exist. However we will not be able to claim that there is a unique such pair. Instead we we will show that *any* pair of polynomials satisfying (refl10:eqn:two) have the same ratio (and this ratio equals p).

Claim 10.2 *There exists a pair of polynomials E and N satisfying (10.2) with the property that $N(x)/E(x) = p(x)$.*

Proof We just take E to be an error-locating polynomial for p and the given set of points, and let $N(x) = p(x)E(x)$. Then N and E satisfy all the requirements. ■

We now move to the more interesting claim. Note that it suffices to argue that $N/E = N'/E'$ for any pair of solutions to (10.2) since the claim above can then be used to see that this ratio must be p.

Claim 10.3 *Any solutions (N, E) and (N', E') to (10.2) satisfy*

$$\frac{N(x)}{E(x)} = \frac{N'(x)}{E'(x)}$$

or equivalently $N(x)E'(x) = N'(x)E(x)$.

Proof Note that the degrees of the polynomials $N(x)E'(x)$ and $N'(x)E(x)$ is at most $k-1+2t$. Furthermore, from (10.2) we have, for every $i \in [n]$,

$$y_i E(x_i) = N(x_i) \quad \text{and} \quad N'(x_i) = y_i E'(x_i).$$

Multiplying the two we get:

$$y_i E(x_i) N'(x_i) = y_i E'(x_i) N(x_i).$$

Now we claim that actually

$$E(x_i) N'(x_i) = E'(x_i) N(x_i).$$

This equality is obvious if $y_i \neq 0$, by cancellation. But why is it true is $y_i = 0$? Well, in such a case, we note (from (10.2) again) that $N(x_i) = N'(x_i) = 0$ and so again we have $E(x_i)N'(x_i) = E'(x_i)N(x_i) = 0$. Now if $n > k + 2t - 1$, then we get that the polynomials $N \cdot E'$ and $E \cdot N'$ agree on more points than their degree, and hence they are identical, as desired. ■

We summarize with the following theorem:

Theorem 10.4 *The* **Welch-Berlekamp algorithm** *solves the* **Unambiguous Reed-Solomon decoding problem** *in $O(n^3)$ time.*

As mentioned earlier, this is not the fastest possible algorithm for solving the unambiguous decoding problem. In fact, even the algorithm from [122] is faster and runs in time $O(n^2)$. If we look at the steps of the algorithm given here, one notes that all that the algorithm needs to be able to is solve a "rational function interpolation" problem and a "polynomial division" problem efficiently. Turns out both steps can be solved efficiently, in $O(n \operatorname{poly} \log n)$ time. For the task of polynomial division, such an algorithm dates back to Sieveking [105]. The text by Aho, Hopcroft, and Ullman [2, Chapter 8] is an excellent source for material on this algorithm (and other fast algorithms for polynomial manipulation). The task of rational function interpolation on the other hand reduces to a "GCD-like" computation, which in turn was shown to be efficiently computable in the 1970s, by Schönhage [97]. Madhu's hastily written notes [111] and Michael Rosenblum's more careful writeup [93] fill in the details of the reduction and sketch the algorithm.

Chapter 11

6.897 Algorithmic Introduction to Coding Theory	October 24, 2001
Lecture 11	
Lecturer: Madhu Sudan	*Scribe: Matt Lepinski*

Today we will talk about:

1. Abstraction of the Decoding Algorithm for Reed-Solomon Codes
2. Decoding Concatenated Codes (specifically, the Forney Codes).

11.1 Abstraction of Reed-Solomon Decoding Algorithm

Our first goal today is to give a very abstract view of the Welch-Berlekamp decoding algorithm for Reed-Solomon codes. This abstraction will allow us to see its generality, and thus apply it to other families of error-correcting codes. The algorithm we describe here is from the works of Pellikaan [86], Kötter [61], and Duursma [30].

11.1.1 Reed-Solomon Decoding Review

Recall the Reed-Solomon decoding algorithm from last lecture. Here we have n distinct points $x_1, \ldots, x_n \in \mathbb{F}_q$ given implicitly, and we are explicitly given as input elements $y_1, \ldots, y_n \in \mathbb{F}_q$. The decoding algorithm consists of the following two steps:

1. FIND: Polynomials $a(x)$ and $b(x)$ such that:
   - For all $i \in [n]$, $a(x_i)y_i = b(x_i)$.
   - Degree of a is small (at most t).
   - Degree of b is small (less than $k + t$).
2. OUTPUT: $\frac{b(x)}{a(x)}$

11.1.2 Special Structure of Reed-Solomon Codes

The above algorithm and its proof of correctness used the properties of Reed-Solomon codes in several ways. Below we list the different aspects that seem specific to Reed-Solomon codes.

1. We used the fact that the indices of the codewords (i.e., $x_1, \ldots, x_n$) are field elements.
2. We used the fact that two low degree polynomials cannot agree in very many places, several times (in the construction of the code, as well as the analysis of the decoding algorithm).
3. We used the fact that multiplication of two low degree polynomials is a low degree polynomial.
4. We also used the fact that under the right conditions, the ratio of two polynomials is a polynomial.

Not very many of these above facts were really critical to the proof — they were just the simplest way to get to the end. Some of the above properties (like (2)) are just facts that hold for any error-correcting code. Others, in particular (3), are somewhat special, but can still be abstracted with care.

11.1.3 Multiplication of vectors

One of the critical operations in the decoding algorithm is that of multiplying the error-locator polynomial $a(x)$ with the message polynomial $p(x)$ and considering the evaluations of the product at $x_1, \ldots, x_n$. This is essentially a coordinatewise product of vectors, defined below.

Definition 11.1 *For $\mathbf{u}, \mathbf{v} \in \mathbb{F}_q^n$, their coordinatewise product, denoted $\mathbf{u} \star \mathbf{v}$, is given by*

$$\mathbf{u} \star \mathbf{v} = \langle u_1 v_1, \ldots, u_n v_n \rangle.$$

If the coordinatewise product is a strange operation in linear algebra, then the following product of sets is even stranger, but is critical to the working of the Reed-Solomon decoding algorithm.

Definition 11.2 *For $U, V \subseteq \mathbb{F}_q^n$ their product, denoted $U \star V$ is given by*

$$U \star V = \{\mathbf{u} \star \mathbf{v} | \mathbf{u} \in U, \mathbf{v} \in V\}.$$

Why are these operations interesting? To motivate them, let us look back at the RS decoding algorithm:

- Let U be the set of vectors obtained by evaluations of polynomials of degree at most t.
- Let V be the set of vectors obtained by evaluations of polynomials of degree less than k.
- Then $U \star V$ is the set of evaluations of polynomials of degree less than $k+t$ that factor into a polynomial of degree less than k and a polynomial of degree at most t.
- In particular $U \star V$ is a subset of the set of evaluations of polynomials of degree less than $k+t$.

- To see that this is special, note that U is a vector space of dimension $t+1$ and V is a vector space of dimension k. What we have noticed is that their product is contained in a vector space of dimension $k+t$. This is *very* special. In general if we take two arbitrary vector spaces of dimension k and t, their product would not be contained in any vector space of dimension less than kt — so polynomials end up being very special!

In what follows we will show that this is the *only* speciality of polynomial based codes. We will show that if any code ends up having nice properties with respect to product with some other codes, then it can be decoded.

11.1.4 Error-Locating Pairs

Let C be a $[n, k, ?]_q$ code and suppose, we wish to decode up to e errors with C. The following definition describes a simple combinatorial object, whose existence suffices to give a decoding algorithm for C.

Definition 11.3 *A pair of linear codes (A, B), with $A, B \subseteq \mathbb{F}_q^n$, form an e-*error-correcting pair *for a linear code $C \subseteq \mathbb{F}_q^n$ if they satisfy the following conditions:*

1. $A \star C \subseteq B$.
2. *The dimension of A is sufficiently large: Specifically,* $\dim(A) > e$.
3. *The minimum distance of B is sufficiently large: Specifically,* $\Delta(B) > e$.
4. *The minimum distance of C is sufficiently large:* $\Delta(C) > n - \Delta(A)$.

11.1.5 The Generalized Algorithm

In this algorithm, we assume that we are given generator matrices for linear codes A, B, and C, where $(A, B$ form an e-error-correcting pair for C.

Abstract decoding algorithm

Given: Matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}$ generating codes $A, B, C \subseteq \mathbb{F}_q^n$, with (A, B) forming e-error-correcting pair for C. Received vector $\mathbf{y} \in \mathbb{F}_q^n$.

Step 1: Find $\mathbf{a} \in A$ and $\mathbf{b} \in B$ such that $\mathbf{a} \star \mathbf{y} = \mathbf{b}$ and $(\mathbf{a}, \mathbf{b}) \neq (\mathbf{0}, \mathbf{0})$, if such a pair exists.

Step 2: Compute $\mathbf{z} \in (\mathbb{F}_q \cup \{?\})^n$ as follows: If $a_i = 0$, then $z_i =?$ else $z_i = y_i$.

Step 3: Output the result of performing erasure decoding on $\mathbf{z}$ for code C, if this results in a unique codeword.

As usual we argue efficiency first and correctness later. Efficiency is obvious for Step 2. For Step 3, recall that we observed in the last lecture that erasure-decoding can be done in $O(n^3)$ time for every linear code. So it suffices to argue efficiency of Step 1. We do so by claiming this is a task of finding a solution to a linear system. Note that we are searching for unknowns $a_1, \ldots, a_n, b_1, \ldots, b_n \in \mathbb{F}_q$. The condition that $\mathbf{a} \in A$ places linear constraints on $a_1, \ldots, a_n$. Similarly, the condition $\mathbf{b} \in B$

turns into linear constraints on $b_1, \ldots, b_n$. Also, the constraints $a_i y_i = b_i$ is also linear in a_i and b_i (since y_i is fixed). Lastly, the condition $(\mathbf{a}, \mathbf{b}) \neq (\mathbf{0}, \mathbf{0})$ is just asking for a non-zero solution to this linear system. So the task at hand is that of finding a non-trivial solution to a homogenous linear system. Again this can be done efficiently, in $O(n^3)$ time. It remains to prove correctness of the above algorithm, and we do so next.

11.1.6 Correctness

The proof of correctness goes through the usual steps. We assume below that there exists a codeword $\mathbf{c} \in C$ that is close to (within a Hamming distance of e of) the received vector $\mathbf{y}$. We fix this codeword and use it to argue correctness.

Below we argue that a solution pair $(\mathbf{a}, \mathbf{b})$ to Step 1 does exist. We argue that any solution pair $(\mathbf{a}', \mathbf{b}')$ to Step 1 satisfies $\mathbf{a}' \star \mathbf{c} = \mathbf{b}'$. Next we show that for any pair $\mathbf{a}' \in A$ and $\mathbf{b}' \in B$ there is at most one $\mathbf{c}' \in C$ such that $\mathbf{a}' \star \mathbf{c}' = \mathbf{b}'$. The correctness follows by noticing that the solution $\mathbf{c}'$ output by the algorithm satisfies $\mathbf{a}' \star \mathbf{c}' = \mathbf{b}'$, if $(\mathbf{a}', \mathbf{b}')$ is the solution found by Step 1. Details below.

Claim 11.4 *There exists a pair* $(\mathbf{a}, \mathbf{b})$ *as required in Step 1 exists. Furthermore they satisfy* $\mathbf{a} \star \mathbf{c} = \mathbf{b}$.

Proof By peeking back at the analogous claim in the Reed-Solomon decoder, we realize we want $\mathbf{a}$ to be the "error-locator", i.e., satisfying $a_i = 0$ if $y_i \neq c_i$. Can we find such a vector that is non-zero? Turns out we can, if we know A has sufficiently large dimension. In particular, the constraints $a_i = 0$ give e homogenous linear constraints on $a_1, \ldots, a_n$. Since $\mathbf{A}$ has dimension $e+1$ or larger (Condition (2) of the definition of an error-correcting pair), it contains a non-zero vector that satisfies all these constraints!

Now take $\mathbf{a}$ to be any non-zero vector satisfying $a_i = 0$ if $c_i \neq y_i$. Take $\mathbf{b}$ to be the vector $\mathbf{a} \star \mathbf{c}$. Note that $\mathbf{b} \in B$ since $A \star C \subseteq B$. Furthermore, for every i, we have either $c_i = y_i$ and so $b_i = a_i c_i = a_i y_i$, or we have $a_i = 0$ and hence $b_i = a_i c_i = 0 = a_i y_i$. Furthermore the pair is non-zero since $\mathbf{a} \neq \mathbf{0}$. This concludes the proof of the claim. ■

Claim 11.5 *If* $\mathbf{a}', \mathbf{b}'$ *are any pair of solutions to Step 1, then* $\mathbf{a}' \star \mathbf{c} = \mathbf{b}'$.

Proof Since $\mathbf{a}', \mathbf{b}'$ are outputs of Step 1, they satisfy the condition, $\mathbf{a}' \star \mathbf{y} = \mathbf{b}'$. Let $\mathbf{a}' \star \mathbf{c} = \mathbf{b}^*$. To prove the claim we need to show $\mathbf{b}' = \mathbf{b}^*$. Since $A \star C \subseteq B$, we know that $\mathbf{b}^* \in B$. Therefore, $\mathbf{b}'$ and $\mathbf{b}^*$ are two codewords of B that agree on every coordinate i for which $y_i = c_i$ ($b'_i = a_i y_i$ and $b^*_i = a_i c_i$). But $y_i = c_i$ on at least $n - e$ coordinates, and so $\Delta(\mathbf{b}', \mathbf{b}^*) \leq e$. But $\Delta(B) > e$ (Condition (3) of the definition of an error-correcting pair), implying $\mathbf{b}' = \mathbf{b}^*$ as required. ■

Claim 11.6 *For any* $(\mathbf{a}', \mathbf{b}') \in A \times B - \{(\mathbf{0}, \mathbf{0})\}$ *there exists at most one* $\mathbf{c} \in C$ *such that* $\mathbf{a}' \star \mathbf{c} = \mathbf{b}'$.

Proof Let $\mathbf{c}, \mathbf{c}' \in C$ satisfy $\mathbf{a}' \star \mathbf{c} = \mathbf{b}' = \mathbf{a}' \star \mathbf{c}'$. First, let us note that we actually have $\mathbf{a}' \neq \mathbf{0}$. (If not, then $\mathbf{b}' = \mathbf{a}' \star \mathbf{c}'$ would also be $\mathbf{0}$ and this contradicts the condition that together then are non-zero.)

To prove the claim, we wish to show $\mathbf{c} = \mathbf{c}'$. Since both are codewords of C, it suffices to show that $\Delta(\mathbf{c}, \mathbf{c}') < \Delta(C)$. But note that $c_i = c'_i$ for every i, where $a'_i \neq 0$. Further, since $\mathbf{a}' \neq \mathbf{0}$, we have $a'_i \neq 0$ on at least $\Delta(A)$ coordinates. Thus we have, $\Delta(\mathbf{c}, \mathbf{c}') \leq n - \Delta(A)$. But Condition (4) in the definition of an error-locating pair ensures that $n - \Delta(A) < \Delta(C)$. Thus we get $\Delta(\mathbf{c}, \mathbf{c}') < \Delta(C)$ as desired. ■

We can now formally prove the correctness of the decoding algorithm.

Lemma 11.7 *If (A, B) form an e-error-correcting pair for C and $\mathbf{c} \in C$ and $\mathbf{y} \in \mathbb{F}_q^n$ satisfy $\Delta(\mathbf{c}, \mathbf{y}) \leq e$, then the* **Abstract decoding algorithm** *outputs $\mathbf{c}$ on input $A, B.C$, and $\mathbf{y}$.*

Proof By Claim 11.4, we have that there exists a pair $\mathbf{a}, \mathbf{b}$ satisfying the conditions of Step 1. So some such pair $\mathbf{a}', \mathbf{b}'$ will be found. By Claim 11.5, $\mathbf{c}$ will satisfy $\mathbf{a}' \star \mathbf{c} = \mathbf{b}'$. Since, $\mathbf{a}' \star \mathbf{c} = \mathbf{b}' = \mathbf{a}' \star \mathbf{y}$, we have $c_i = y_i$ whenever $a_i \neq 0$ and thus c_i is a valid solution to Step 3 of the algorithm. To conclude, we need to ensure that $\mathbf{c}$ is the only solution to Step 3 of the algorithm. But this is also clear, since any solution $\mathbf{c}'$ to this step must satisfy $a'_i c'_i = a'_i y_i = b'_i$ for every i, and thus must be a codeword of C satisfying $\mathbf{a}' \star \mathbf{c}' = \mathbf{b}'$ and by Claim 11.6, $\mathbf{c}$ is the unique vector with this property. ■

We conclude with the following theorem:

Theorem 11.8 *Any code C that has an e-error-correcting pair has an efficient ($O(n^3)$ time) algorithm solving the bounded distance decoding problem for up to e errors.*

11.1.7 Applications

Exercise: Verify that the **Welch-Berlekamp** algorithm from the last lecture is an instantiation of the **Abstract decoding algorithm** given today.

We now move on to a more interesting application. Recall the construction of algebraic-geometry codes. (To be more precise, recall that we know very little about them to recall much.)

Algebraic-geometry codes. These codes were constructed by finding n points in $\mathbb{F}_q^m$ and evaluating all polynomials of "order" at most ℓ at all n places. The following properties of order were used in asserting that these gave good codes:

1. There exists an integer g such that for every ℓ, the evaluations of polynomials of order at most ℓ formed a subspace of $\mathbb{F}_q^n$ of dimension at least $\ell - g + 1$.
2. Two distinct polynomials of order at most ℓ can agree on at most ℓ out of the n evaluation points.
3. The product of two polynomials of order ℓ_1 and ℓ_2 has order at most $\ell_1 + \ell_2$.

These properties suffice to prove that codes obtained by the evaluation of polynomials of order at most $n-d+1$ give an $[n, k, d]_q$ code for some $k \geq n-d-g+1$. As we see below, the same properties also give us an $\lfloor \frac{n-g-\ell-1}{2} \rfloor$-error-correcting pair for these codes.

Lemma 11.9 *If C is an algebraic-geometry code obtained by evaluating polynomials of order at most ℓ, then it has an $\lfloor\frac{n-g-\ell-1}{2}\rfloor$-error-correcting pair.*

Proof Let $e \leq \frac{n-g-\ell-1}{2}$. Below all references to the "Conditions" are to the four conditions in the definition of an e-error-correcting pair.

To get an e-error-correcting pair, we need $\dim(A) > e$ (to satisfy Condition 2). We will pick A to be the algebraic-geometry code obtained by evaluations of all polynomials of order at most $e+g$. Since we need $A \star C \subseteq B$, we pick B to be the algebraic-geometry code obtained by all evaluations of polynomials of order at most $e+g+\ell$, and thus satisfy Condition 1. To satisfy Condition 3, we need $\Delta(B) > e$. We know from the properties of algebraic-geometry codes, that B has distance at least $n-e-g-\ell$. From the choice of e, it follows that $n-e-g-\ell > e$. Finally, to get Condition 4, we need to verify that $\Delta(A)+\Delta(C) > n$. Since $\Delta(A) \geq n-e-g$ and $\Delta(C) \geq n-\ell$, this amount to verifying that $2n-\ell-e-g > n$ which is equivalent to verifying that $e < n-\ell-g$. But in fact e is less than half the RHS. Thus (A, B) as chosen above give an e-error-correcting code for C. ■

As a corollary we see that we get a pretty decent decoding algorithm for algebraic-geometry codes, correcting about $(n-\ell-g)/2$ errors. However, it does not decode up to half the minimum distance, since the distance is $n-\ell$ and we are only correcting $(n-\ell-g)/2$ errors. Later in the course we will see a better algorithm.

Before concluding this section, we mention one more case where the abstract decoding algorithm provides an inspiration for a decoding algorithm. This is the case of the Chinese Remainder Codes, described next.

Chinese Remainder Codes. The Chinese Remainder Codes are number-theoretic codes defined as follows:

- Fix primes $p_1, \ldots, p_n$ such that $p_1 < p_2 < \cdots < p_n$
- A *message* is an integer $m \in [0 \ldots K-1]$ where $K = \prod_{i=1}^{k} p_i$.
- The encoding of m is the vector $\langle m(\mathrm{mod} p_1), \ldots, m(\mathrm{mod} p_n)\rangle$.

By the Chinese Remainder Theorem, residues of m modulo *any* k of the n primes, suffices to specify m. Thus specifying its residue modulo n primes, makes the above a redundant encoding. This is not one of our usual algebraic codes — in fact it is not even linear! However, one can apply our usual notions of distance, error-detection and correction to this code. We note the code has distance $n-k+1$ (since specifying the residues modulo k primes, specifies the message). So one can as the question - is it possible to correct $(n-k)/2$ errors. The first decoding algorithm was given by Mandelbaum [76]. Later work of Goldreich, Ron, and Sudan [39] showed how to interpret this algorithm as a relative of the abstract decoding algorithm given here. Turns out both algorithms correct slightly less than $(n-k)/2$ errors in polynomial time — this was fixed later by Guruswami, Sahai, and Sudan [44] using the algorithm of [39] in combination with an algorithm known as the "Generalized Minimum Distance Algorithm" that we will talk about in the next section.

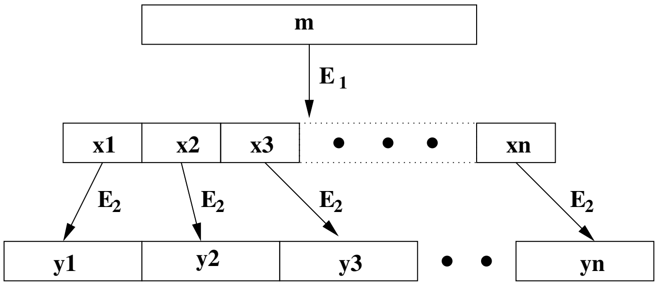


Figure 11.1: Encoding in concatenated codes

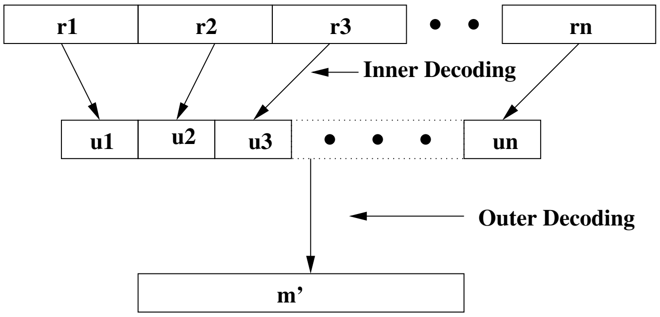


Figure 11.2: Decoding concatenated codes

11.2 Decoding Concatenated Codes

We now move on to an elegant solution for the unambiguous decoding problem for some families of concatenated codes. Let us start by recalling concatenated codes.

Concatenated Codes. Given an $[n, k, d]_Q$ outer code C_1 with encoding function E_1 and an $[n_2, k_2, d_2]_q$ inner code C_2, with encoding function E_2, where $Q = q^{k_2}$, their concatenation, denoted $C_1 \diamond C_2$, is the $[nn_2, kk_2, dd_2]_q$ code obtained as follows: Start with a message $\mathbf{m} \in \mathbb{F}_Q^k$ and encode it using E_1 to get a vector $\mathbf{x} = \langle x_1, \ldots, x_n \rangle$, where $x_i \in \mathbb{F}_Q$. Now, viewing the x_i's as elements of $\mathbb{F}_q^{k_2}$ encode them using E_2 to get $\mathbf{y} = \langle \mathbf{y}_1, \ldots, \mathbf{y}_n \rangle$ where $\mathbf{y}_i \in \mathbb{F}_q^{n_2}$ ad the encoding of $\mathbf{m}$. (See also Figure 11.1.)

Getting some reasonable algorithms for decoding concatenated codes is not so hard, under some reasonable assumptions. Such an algorithm would take a received vector $\mathbf{r} = \langle \mathbf{r}_1, \ldots, \mathbf{r}_n \rangle$, where $\mathbf{r}_i \in \mathbb{F}_q^{n_2}$ and decode them in two steps, inverting the encoding steps. So first it would decode the $\mathbf{r}_i$'s individually to their nearest codewords. For $i \in [n]$, let $E_2(u_i)$ be the codeword of C_2 nearest to $\mathbf{y}_i$, where we view $u_i \in \mathbb{F}_Q$ as an element of $\mathbb{F}_q^{k_2}$. Now we treat $\mathbf{u} = \langle u_1, \ldots, u_n \rangle$ as a corrupted codeword of C_1 and decode it using a decoding algorithm for E_1. (See Figure 11.2.)

First, let us note that for the concatenated codes we have considered so far (Forney codes and Justesen codes), the above is actually efficient. Recall that in these applications the outer code is a well-studied code such as the Reed-Solomon code with efficient decoding algorithms. On the other hand the inner code is not well-understood — the only thing we know about it is that it has good minimum distance properties. So it is not reasonable to expect a sophisticated decoding algorithm for the inner code. But then the inner codes are so small the brute-force decoding only takes polynomial time in the length of the concatenated code, so we don't *need* a sophisticated inner decoding algorithm. So the entire decoding process described above takes time polynomial in the length of the concatenated code, for the typical codes.

However, this doesn't give us an algorithm decoding upto half the minimum distance of the concatenated code! It may only decode $\frac{dd_2}{4}$ errors. We won't prove that it can correct so many errors, or that it can't correct more. But to see a plausibility argument, note that to get a decoding failure, the adversary only has to ensure $d/2$ of the symbols $u_i \neq x_i$ and to get any such decoding error for the inner decoder it may only need to flip $d_2/2$ symbols of the inner alphabet. Thus a total of $dd_2/4$ errors may lead to a decoding failure. Below we will show a clever algorithm which gets around this problem with relatively little extra information about the outer decoding algorithm. This algorithm is the **Generalized Minimum Distance (GMD) Decoding Algorithm**, due to Forney [55] (see also [54]).

Exercise: Prove that the decoding algorithm outlined above does indeed correct at least $(d-1)(d_2-1)/4$ errors.

11.2.1 Decoding errors and erasures

Let us start by looking at the decoding algorithm we have for the outer code. We already seem to be making full use of it — we assume it can correct $(d-1)/2$ errors, and it can't possibly correct more errors unambiguously. Turns out, there is one additional feature of the decoding algorithm for the outer code that comes in quite handy. This is the feature that it can deal with erasures quite naturally and benefit from it.

Proposition 11.10 *Let C be an $[n, n-d+1, d]_q$ Reed-Solomon code. Suppose $\mathbf{r} \in (\mathbb{F}_q \cup \{?\})^n$ is a vector derived from a codeword $\mathbf{c} \in C$ by s erasures and t errors. I.e., $|\{i | r_i = ?\}| = s$ and $|\{i | c_i \neq r_i \text{ and } r_i \neq ?\}| = t$. Then $\mathbf{c}$ can computed efficiently given $\mathbf{r}$, s, t provided $s + 2t < d$.*

Proof The proposition is straightforward, given the observation that the code C', obtained by puncturing C on the coordinates where there are erasures, is an $[n-s, n-d+1, d-s]$ Reed-Solomon code and thus $(d-s-1)/2$ errors can be corrected efficiently. ■

How can we use the option to declare erasures? In the process of decoding the inner code, we ignored some obvious information that was available to us. We not only could find out the encoding of which message word u_i was closest to the ith received block $\mathbf{r}_i$, but we also know how many errors have (potentially) occurred in each block. Somehow we should make use of this information. A natural idea is to to declare blocks with large number of errors to be erasures. This works out roughly correct. We will see that a simple probabilistic interpretation of the distances give a right strategy for declaring erasures.

11.2.2 A randomized decoding algorithm

Let us fix some notation, that we have been introducing as we went along. The message vector is $\mathbf{m}$, its encoding under the outer code is $\mathbf{x} = \langle x_1, \ldots, x_n \rangle$, where $x_i \in \mathbb{F}_Q \cong \mathbb{F}_q^{k_2}$. The encoding of

x_i under the inner code is $\mathbf{y}_i$ and thus the final codeword is $\mathbf{y} = \langle \mathbf{y}_1, \ldots, \mathbf{y}_n \rangle$. The noisy channel corrupts $\mathbf{y}$ and a vector $\mathbf{r} = \langle \mathbf{r}_1, \ldots, \mathbf{r}_n \rangle$ is received. We now describe the decoding algorithm for this code.

Random-Concat-Decoder

Given: $\mathbf{r} = \langle \mathbf{r}_1, \ldots, \mathbf{r}_n \rangle$.

Step 1: For $i \in [n]$, compute u_i that minimizes $\Delta(E_2(u_i), \mathbf{r}_i)$.

Step 2: Set $e'_i = \min\{d_2/2, \Delta(E_2(u_i).\mathbf{r}_i)$.

Step 3: For every $i \in [n]$ repeat the following: With probability $2e'_i/d_2$, set $v_i = ?$, else set $v_i = u_i$.

Step 4: Perform errors and erasures decoding of the vector $\mathbf{v} = \langle v_1, \ldots, v_n \rangle$.

Step 5: If a vector $\mathbf{m}' \in \mathbb{F}_Q^k$ is obtained in Step 4, check to see if $\Delta(E_2 \diamond E_1(\mathbf{m}), \mathbf{r}) \leq (dd_2)/2$ and if so output $\mathbf{m}'$.

The above algorithm clearly works in time that is $O(nn_2Q + T(n))$, where $T(n)$ is the time taken by the errors and erasures decoding algorithm. The main issue is how many errors it corrects. We will claim that it "essentially" solves the unambiguous decoding algorithms for the concatenated code. We do so via a claim that is weak in that it only shows that the expected number of errors and erasures fall within the decoding capability of the outer code.

Lemma 11.11 *Let $e_i = \Delta(\mathbf{r}_i, \mathbf{y}_i)$ and $e = \sum_{i=1}^n e_i$. When $\mathbf{v}$ is picked at random by* **Random-Concat-Decoder***, then we have:*

$$\mathrm{Exp}\left[\#\ \textit{erasures in}\ \mathbf{v}\ + 2 \cdot (\#\ \textit{errors in}\ \mathbf{v})\right] \leq 2e/d_2.$$

Remark: Note that if $e < dd_2/2$, the RHS above is less than d as one would hope for.

Proof Note that is suffices to argue that for every i

$$\Pr\left[\text{erasure in } i\text{th coordinate}\right] 2 \cdot \Pr\left[\text{error in } i\text{th coordinate}\right] \leq 2e_i/d_2, \tag{11.1}$$

and then the lemma will follow by linearity of expectations. We prove this in cases.

Case 1: $u_i = x_i$: In this case, the probability of an error in the ith coordinate is 0 and the probability of an erasure in the ith coordinate is $2e'_i/d_2$. Since $u_i = x_i$, we also have $e_i = e'_i$, and so we find that the LHS of (11.1) is indeed equal to $2e_i/d_2$.

Case 2: $u_i \neq x_i$. In this case, we could have an error in the ith coordinate — this happens with probability $1 - 2e'_i/d_2$, while with probability $2e'_i/d_2$ we get an error. We need to express these quantities as a function of e_i (rather than e'_i) and so we note that $e_i \geq d_2 - e'_i$ (since $\Delta(E_2(x_i), \mathbf{r}_i) \geq \Delta(E_2(x_i), E_2(u_i)) - \Delta(\mathbf{r}_i, E_2(u_i)))$. Now we have that the probability of an error is at most $2e_i/d_2 - 1$ and the probability of an erasure is at most $2 - 2e_i/d_2$. Plugging these into the LHS of (11.1), again we find that the LHS is bounded by $2e_i/d_2$. ■

The above lemma implies that there is positive probability associated with the event that $\mathbf{v}$ has small number of errors and erasures. However we did prove not a high probability result. We won't do so; instead we will just derandomize the algorithm above to get a deterministic algorithm for the unambiguous decoding problem.

11.2.3 Deterministic Decoder for Concatenated Codes

We will develop the deterministic decoder in two (simple) steps. First note that we didn't really need the random choices in Step 3 of **Random-Concat-Decoder** do not need to independent and expectation bound (Lemma 11.11) also holds if these events are completely dependent. So, in particular, the following algorithm would work as well.

Modified-Random-Concat-Decoder

Given: $\mathbf{r} = \langle \mathbf{r}_1, \ldots, \mathbf{r}_n \rangle$.

Step 1: For $i \in [n]$, compute u_i that minimizes $\Delta(E_2(u_i), \mathbf{r}_i)$.

Step 2: Set $e'_i = \min\{d_2/2, \Delta(E_2(u_i).\mathbf{r}_i)$.

Step 3.1: Pick $p \leftarrow [0,1]$ uniformly at random.

Step 3.2: For every $i \in [n]$ if $2e'_i/d_2 > p$, set $v_i = ?$, else set $v_i = u_i$.

Step 4: Perform errors and erasures decoding of the vector $\mathbf{v} = \langle v_1, \ldots, v_n \rangle$.

Step 5: If a vector $\mathbf{m}' \in \mathbb{F}_Q^k$ is obtained in Step 4, check to see if $\Delta(E_2 \diamond E_1(\mathbf{m}), \mathbf{r}) \leq (dd_2)/2$ and if so output $\mathbf{m}'$.

As in the analysis of **Random-Concat-Decoder** we see that the random variable $\mathbf{v}$ as defined in Steps 3.1 and 3.2 above satisfies the condition:

$$\text{Exp}\left[\#\text{ erasures in } \mathbf{v} \ + 2 \cdot (\#\text{ errors in } \mathbf{v})\right] \leq 2e/d_2.$$

In particular, there exists a choice of p in Step 3.1, such that for this choice of p, the vector $\mathbf{v}$ obtained in Step 3.2 satisfies the condition:

$$(\#\text{ erasures in } \mathbf{v}) \ + 2 \cdot (\#\text{ errors in } \mathbf{v}) \leq 2e/d_2.$$

The only interesting choices of p are from the set $S = \{0,1\} \cup \{2e'_i/d_2 | i \in [n]\}$ since for every other choice of p, there is some $p' \in S$ for which the vector $\mathbf{v}$ obtained is identical for p' and p. This gives us the deterministic algorithm below:

Deterministic-Concat-Decoder

Given: $\mathbf{r} = \langle \mathbf{r}_1, \ldots, \mathbf{r}_n \rangle$.

Step 1: For $i \in [n]$, compute u_i that minimizes $\Delta(E_2(u_i), \mathbf{r}_i)$.

Step 2: Set $e'_i = \min\{d_2/2, \Delta(E_2(u_i).\mathbf{r}_i)$.

Step 3.1: Let $S = \{0,1\} \cup \{2e'_i/d_2 | i \in [n]\}$. Repeat Steps 3.2 to 5 for every choice of $p \in S$.

Step 3.2: For every $i \in [n]$ if $2e'_i/d_2 > p$, set $v_i = ?$, else set $v_i = u_i$.

Step 4: Perform errors and erasures decoding of the vector $\mathbf{v} = \langle v_1, \ldots, v_n \rangle$.

Step 5: If a vector $\mathbf{m}' \in \mathbb{F}_Q^k$ is obtained in Step 4, check to see if $\Delta(E_2 \diamond E_1(\mathbf{m}), \mathbf{r}) \leq (dd_2)/2$ and if so output $\mathbf{m}'$.

The running time of this algorithm is $O(nn_2Q + nT(n))$ where $T(n)$ is the running time of the errors and erasures decoder of the outer code. Correctness follows from the arguments developed so far. We summarize the discussion with the following theorem:

Theorem 11.12 *Let C be the concatenation of an $[n, k, d]_Q$ outer code C_1 and an $[n_2, k_2, d_2]_q$ inner codee C_2 with $Q = q^{k_2}$. Suppose C_1 has an "errors and erasures decoding algorithm" running in time $T(n)$ decoding up to s erasures and t errors provided $s + 2t < d$. Then C has an unambiguous decoding algorithm running in time $O(nn_2Q + nT(n))$.*

Again, the running times can be improved with some effort. In particular, Kötter [62] shows how to cut down the run time to just $O(nn_2Q + T(n))$ for some families of concatenated codes.

Chapter 12

6.897 Algorithmic Introduction to Coding Theory October 29, 2001

Lecture 12

Lecturer: Madhu Sudan *Scribe: George Savvides*

Today's topics:

1. Combinatorics (revisited), including two new proofs of the Johnson bound.
2. List decoding for Reed-Solomon codes.

12.1 Combinatorics revisited

Let us recall the notion of an (e, ℓ)-error-correcting code [1]

Definition 12.1 *Given an* $[n, k, d]_q$*-code* C*, we say that* C *is an* (e, ℓ)-error correcting code*, if for* any *received vector* $\mathbf{r} \in \mathbb{F}_q^n$*, there exist at most* ℓ *codewords* $\mathbf{c}_1 \ldots, \mathbf{c}_\ell \in C$ *such that* $\Delta(\mathbf{c}_i, \mathbf{r}) \leq e$.

In other words the code C can correct up to e errors with lists of size ℓ. The natural algorithmic problem arising out of the above definition is whether (and how) we can find all such codewords in polynomial time. That is:

Given: $\mathbf{r} \in \mathbb{F}_q^n$, an arbitrary received vector).

Output: A set (list) of codewords $\{c_1, \ldots, c_\ell\}$ such that every codeword $\mathbf{c}$ that satisfies $\Delta(\mathbf{c}, \mathbf{r}) \leq e$ is included in this set.

We will require that our algorithm's running time be polynomial in the *input* size. Notice that it is possible that if ℓ is superpolynomial, then such an algorithm can not exist! So, a prerequisite

[1] Note that in previous lectures, we used to call this a (t, ℓ)-error-correcting code. We are changing the parameter label for this lecture, since we end up using t for the "opposite" of e, i.e., the number of agreements rather than number of errors.

for the existence of this algorithm is that the code be an (e, ℓ)-error-correcting code for ℓ that is a polynomial in the block length of the code. When do we know that an $[n, k, d]_q$ code is also an (e, ℓ)-error-correcting code for some ℓ that is polynomial in n? The Johnson bound ought to come to mind at this stage. (Though, given the general sentiment against naming bounds by discoverer's names, one can be forgiven for not coming up with the right name. Hopefully the question still does ring a bell.) We recall the Johnson bound below:

Theorem 12.2 (Johnson bound) *Let $\epsilon = \frac{e}{n}$ and $\delta = \frac{d}{n}$. An $[n, k, d]_q$-code is an (e, n)-error-correcting*[2] *code provided that*

$$\epsilon \leq \frac{q-1}{q}\left(1 - \sqrt{1 - \frac{q}{q-1}\delta}\right)$$

Recall that in class we proved the binary case of the theorem, where the condition on ϵ was

$$\epsilon \leq \frac{1}{2}\left(1 - \sqrt{1-2\delta}\right).$$

The Johnson bound applied to Reed-Solomon codes In the case of RS codes, $q \to \infty$, so we can simplify the bound above. In particular, we get that the $[n, k, d]_q$ RS-code is also an $(\epsilon n, n)$-error-correcting code for $\epsilon \leq 1 - \sqrt{1-\delta}$. Using the fact that the rate R of such a code is $1 - \delta$, the above bound says that the "list-of-n decoding radius" of the code is $1 - \sqrt{R}$. To compare this with the list-of-1 decoding radius (or the unambiguous decoding radius), in such case we can only allow $\epsilon \leq \frac{1-R}{2}$. Thus, as we would hope, the list-decoding radius is strictly larger than the unambiguous decoding radius for any Reed-Solomon code of rate less than 1.

Furthermore, to highlight the difference, let us consider an extremal case when we let the rate approach zero:

1. $\lim_{R \to 0} 1 - \sqrt{R} = 1 \quad (\text{so } \epsilon \to 1)$
2. $\lim_{R \to 0} \frac{1-R}{2} = 1/2 \quad (\text{so } \epsilon \to 1/2)$

Thus in Case 2 (unambiguous decoding), a necessary condition for successful decoding is that the fraction of errors be bounded from above by the fraction of correct information. However, list-decoding seems feasible (combinatorially, at least) even when this condition is violated — i.e., when the amount of correct information is overwhelmed by the amount of erroneous information. As we will see over the course of the next couple of lectures, this barrier can be overcome even algorithmically! But first, we will celebrate the Johnson bound (er ... yes, we still don't have a better way to name the bounds) by giving two alternate proofs of the bound.

12.2 Two proofs of the Johnson bound

The elegance and beauty of some of the (numerous) proofs of the Johnson bound are part of the reason why this particular bound is of interest to us. Below we will see two new proofs of the Johnson bound for the case of Reed-Solomon codes. Let us first recall the problem being considered:

[2] yes, n is both the size of the codewords and the size of the list

Given n distinct points $x_1, \ldots, x_n \in \mathbb{F}_q$ and n values $r_1, \ldots .r_n \in \mathbb{F}_q$, bound the number of polynomials p of degree at most k, such that $p(x_i) = r_i$ for at least t values of $i \in [n]$.

Note that we have shifted the focus to the number of agreements rather than the number of errors. This is a better choice, since this is the smaller number and hence it is a more refined number to look at (as opposed to the number of disagreements). Our bounds will be of the form "the number of such polynomials is small, provided $t \geq c\sqrt{kn}$. We now give the proofs.

12.2.1 Using the Inclusion-Exclusion principle

The inclusion-exclusion principle shows how to count the number of elements in the union of ℓ sets, given the sizes of all interesections. While the full formula (not given here) gives the size exactly, portions of the formula are known to give bounds on the size of the union. The union bound is a simple version which says that the size of the union is *at most* the sum of the sizes of the individual sets. Looking at sizes of pairwise intersections gives the lower bound below:

Proposition 12.3 (Weak Inclusion-Exclusion Principle) *For any ℓ subsets $S_1, \ldots, S_\ell$ of some finite universe X, the following inequality holds:*

$$\left|\bigcup_{j=1}^{\ell} S_j\right| \geq \sum_{j=1}^{l} |S_j| - \sum_{1 \leq j_1 < j_2 \leq l} \left|S_{j_1} \bigcap S_{j_2}\right|.$$

Exercise: Prove Proposition 12.3.

Now, we show how to apply the Inclusion-Exclusion Principle to our problem. Suppose $p_1, \ldots, p_\ell$ are degree k polynomials that agree with the given points on at least t places. We define S_j, $j \in [\ell]$ as follows:

$$S_j = \{i \in [n] \mid p_j(x_i) = r_i\}.$$

We note that S_j's satisfy the following properties:

- For every $j \in [n]$, $|S_j| \geq t$: By hypothesis.
- $\left|\bigcup_{j=1}^{\ell} S_j\right| \leq n$: Obvious, since each $S_j \subseteq [n]$.
- For every distinct j_1, j_2, $|S_{j_1} \bigcap S_{j_2}| \leq k$: Since $p_{j_1}(x_i) = p_{j_2}(x_i)$ for every $i \in S_{j_1} \bigcap S_{j_2}$, and two distinct degree k polynomials may agree on at most k distinct elements of $\mathbb{F}_q$.

From the above, and the inclusion-exclusion principle, and the fact that there are $\binom{\ell}{2}$ pairs of intersections to consider, we have:

$$n \geq \ell \cdot t - \binom{\ell}{2} \cdot k. \tag{12.1}$$

Further, by monotonicity, we actually have

$$\forall \ell' \leq \ell \quad n \geq \ell' \cdot t - \binom{\ell'}{2} \cdot k. \tag{12.2}$$

(If there are ℓ polynomials with agreement of t, then are also $\ell' \leq \ell$ polynomials with agreement t.)

The following unremarkable and tedious lemma finishes off the proof:

Lemma 12.4 *Suppose n, k, t are positive integers such that $t > \sqrt{2kn}$. Then $n < \ell' \cdot t - \binom{\ell'}{2} \cdot k$ for $\ell' = \lceil \frac{2n}{t} \rceil$.*

Proof Let $g(x) = \frac{k}{2}x^2 - \left(t + \frac{k}{2}\right)x + n$. Note that $\frac{2n}{t} < \ell' \leq \frac{2n}{t} + 1$. By convexity of $g(\cdot)$ we have $g(\ell') \leq \max\left\{g\left(\frac{2n}{t}\right), g\left(\frac{2n}{t} + 1\right)\right\}$. We argue that both $g\left(\frac{2n}{t} - 1\right)$ and $g\left(\frac{2n}{t}\right)$ are negative. First,

$$\begin{aligned}
g(\frac{2n}{t}) &= \frac{2kn^2}{t^2} - \frac{n(2t+k)}{t} + n \\
&= \frac{n(2kn - t^2 - kt)}{t^2} \\
&< \frac{-nkt}{t^2} \quad \text{(Using } 2nk < t^2\text{)} \\
&\leq 0.
\end{aligned}$$

Next, we have:

$$\begin{aligned}
g(\frac{2n}{t} + 1) &= \frac{k(2n+t)^2}{2t^2} - \frac{(2t+k)(2n+t)}{2t} + n \\
&= \frac{4n^2k + 4nkt + t^2k - 4nt^2 - 2t^3 - 2nkt - kt^2 + 2nt^2}{2t^2} \\
&= \frac{4n^2k + 2nkt - 2nt^2 - 2t^3}{2t^2} \\
&< \frac{t^2(2n+t) - 2nt^2 - 2t^3}{2t^2} \quad \text{(Using } 2nk < t^2\text{)} \\
&= \frac{(2n+t) - 2n - 2t}{2} \\
&= \frac{-t}{2} \\
&< 0.
\end{aligned}$$

■

We conclude with the first bound.

Theorem 12.5 *Given points $\{(x_i, r_i)\}_{i=1}^n$ there are at most $\frac{2n}{t}$ polynomials p of degree at most k that satisfy $|\{i \in [n] \mid p(x_i) = y_i\}| \geq t$ if $t > \sqrt{2kn}$.*

Note that this bound as well as the next one are just weak versions of the bound given in Theorem 12.2. Both apply generally to codes. However we are just specializing the arguments to the case of polynomials, to keep the notation simple. We just repeat them here to give further intuition into what they are trying to prove, and to illustrate the simple proof techniques.

12.2.2 Using graph theory

Our second proof of the "Johnson-like bound" goes via graph theory. We model the problem graph-theoretically as follows:

Given a received vector $\mathbf{r}$ and codewords $\mathbf{c}_1, \ldots, \mathbf{c}_\ell$ we construct a bipartite graph B with vertex set $L \cup R$ and edges being a subset of $L \times R$ as follows:

Left vertices: $L = [n]$, corresponding to the n coordinates of the received vector.

Right vertices: $R = [\ell]$ corresponding to the ℓ codewords $\mathbf{c}_1, \ldots, \mathbf{c}_\ell$.

Edges: There is an edge between $i \in L$ and $j \in R$ iff $r_i = (\mathbf{c}_j)_i$.

The properties of the problem at hand translate to graph theoretic properties as follows:

Distance property: Two codewords can't agree with each other on k indices. In particular this implies that two coordinates can't agree with $\mathbf{r}$ on the same subset of k indices. Thus the graph B does not contain $K_{k,2}$, the complete bipartite graph with k vertices on the left and 2 vertices on the right, as a subgraph.

Agreement property: The agreement between codewords and $\mathbf{r}$ implies that every vertex on the right is adjacent to at least t vertices on the left. For simplicity in the analysis we will throw away some edges in the graph, so as to get right vertices to have degree exactly t. Notice the graph still does not have any $K_{k,2}$ after this deletion.

We will see that these properties will allow us to conclude that the graph doesn't have too many vertices on the right (so ℓ is "small", as desired). Such bounds are called "Zarankiewicz bounds" after a Polish mathematician Kazimierz Zarankiewicz who studied such forbidden subgraph problems.

Lemma 12.6 *If a bipartite graph $B = (L, R, E)$, with $|L| = n$ and $|R| = \ell$, has right degree t and no $K_{k,2}$,*

$$\ell < \frac{n(t-k)}{t^2 - kn} \quad \textit{provided } t^2 > nk.$$

Proof We prove the lemma using a probabilistic argument. Let d_i denote the degree of the ith vertex in L. Note that $\sum_i d_i = \ell \cdot t$.

Let p_i denote the probability, when two distinct elements j_1 and j_2 are picked uniformly at random from R, that both j_i and j_2 are adjacent to i. We have:

$$p_i = \frac{\binom{d_i}{2}}{\binom{\ell}{2}} = \frac{d_i^2 - d_i}{\ell^2 - \ell}.$$

Now consider the expected number of common neighbours that distinct elements j_1 and j_2 have when they are picked uniformly at random from R. This quantity is given by

$$\sum_i p_i = \frac{1}{\ell^2 - \ell} \sum_{i=1}^{n} (d_i^2 - d_i).$$

Since this quantity better be less than k, we get

$$\frac{1}{\ell^2 - \ell} \sum_{i=1}^{n} (d_i^2 - d_i) < k.$$

The non-trivial component in the inequality above is $\sum_{d_i^2}$. Subject to the condition that $\sum_i d_i = \ell t$, this quantity is minimized when all the d_i's are equal This is proves via a standard inequality known as the Cauchy-Schwartz inequality. For any two real vectors $\mathbf{a} = \langle a_1, \dots, a_n \rangle$ and $\mathbf{b} = \langle b_1, \dots, b_n \rangle$, the Cauchy-Schwartz inequality says that $\langle \mathbf{a}, \mathbf{b} \rangle^2 \leq (\|\mathbf{a}\|^2 \cdot \|\mathbf{b}\|^2)$. Applying it to $\mathbf{a} = \langle d_1, \dots, d_n \rangle$ and $\mathbf{b} = \langle 1, \dots, 1 \rangle$, we get $(\sum_{i=1}^n d_i)^2 \leq n \sum_{i=1}^n d_i^2$. Thus the inequality above becomes:

$$\frac{1}{n}\left(\sum_{i=1}^n d_i\right)^2 - \sum_i d_i < k(\ell^2 - \ell).$$

$$\Leftrightarrow \frac{\ell^2 t^2}{n} - \ell t < k(\ell^2 - \ell).$$

$$\Leftrightarrow \ell\left(\frac{t^2}{n} - k\right) < t - k.$$

Thus we have

$$\ell < \frac{n(t-k)}{t^2 - kn} \quad \text{provided } t^2 > nk.$$

■

Theorem 12.7 *Given points $\{(x_i, r_i)\}_{i=1}^n$ there are at most $\frac{n(t-k)}{t^2-kn}$ polynomials p of degree at most k that satisfy $|\{i \in [n] \mid p(x_i) = y_i\}| \geq t$ if $t > \sqrt{kn}$.*

12.3 List decoding for RS codes

We now move on to the algorithmic versions of Theorems 12.5 and 12.7. We state the problem next.

> **List decoding of Reed-Solomon codes:**
> GIVEN: Point pairs $\{(x_i, y_i) \mid i \in [n]\}$ with each pair being distinct and parameters k and t.
> FIND: A list of all polynomials p of degree at most k that satisfy $p(x_i) = y_i$ for at least t values of $i \in [n]$.

Our goal is to get a solution that works provided $t > \sqrt{kn}$ (so as to match Theorem 12.7). Today we will settle for any $t = \Theta(\sqrt{kn})$. The algorithm is actually pretty simple. We motivate it by looking at a toy problem.

12.3.1 A toy problem

Suppose the message polynomial is a polynomial p_1 and the errors in our point pairs actually come from a second polynomial p_2. It is clear that our algorithm should output as answers p_1 and p_2 (assuming both polynomials are represented often enough). How do we come up with an algorithm whose output is "The answer is $p_1(\cdot)$ OR $p_2(\cdot)$"? The Boolean OR in the statement seems to be hard to capture algebraically. We attempt to capture this by introducing *two* variables x and y representing the two coordinates of the pairs that are given to us and noticing that it is easy to come up with an algebraic *relation* explaining all the given points.

The given set of points satisfy:

$$(y_i - p_1(x_i)) \cdot (y_i - p_2(x_i)) = 0.$$

(In other words, multiplication captures the OR of two relations). Indeed if we can find two polynomials p_1 and p_2 such that the above relation is satisfied by all pairs of points, then we could output p_1 and p_2 as our solution, and this would seems to be good enough. (We will prove later that this would suffice, but now we turn to the more important question.) The question is, how do we find such polynomials?

To find p_1 and p_2, we expand the quadratic expression above, and see that there must exist two polynomials $B(x)$ and $C(x)$ such that the following conditions hold:

1. For every $i \in [n]$, $y_i^2 - B(x_i)y_i + C(x_i) = 0$.
2. There exist polynomials p_1 and p_2 such that
   - $B(x) = p_1(x) + p_2(x)$.
   - $C(x) = p_1(x) \cdot p_2(x)$.
3. $\deg(B) \le k$ and $\deg(C) \le 2k$.

As in previous cases (e.g., the Welch-Berlekamp algorithm), we ignore conditions that seem to be hard to enforce and hope the rest suffice to find the solution anyway. In the above set of conditions, (1) and (3) are easy to enforce (we will see so shortly), but condition (2) is not! So we will omit condition (2) and look for B and C satisfying the remaining conditions. This motivates the following algorithm.

Algorithm for toy problem:

Step 1: find B and C such that $\forall\ i \in [n]$:

- $y_i^2 - B(x_i)y_i + C(x_i) = 0$,
- $\deg(B) \le k$, and
- $\deg(C) \le 2k$,

provided such a pair exists.

Step 2: Factor the polynomial $y^2 - B(x)y + C(x)$. If it factors into $(y - f(x)) \cdot (y - g(x))$, output f and g.

As in the case of the Welch-Berlekamp algorithm, we have to argue that this algorithm is *efficient* and *correct* and we go through the usual steps to do so. Btw, this paradigm is not special to coding theory, but rather to algebraic algorithms, where one finds out enough information about the solution to outline an algorithm, and then shows that the information is sufficient to have pinned down the answer separately.

We start by describing the algorithmic complexity of the two steps above:

Step 1: We can approach this problem through linear algebra. We wish to find polynomials B and C, or rather their coefficients so that the constraints of Step 1 are satisfied. The constraints are linear in the coefficients (though not linear in x_i's and y_i's) and so if a solution exists, one can be found efficiently.

Step 2: This specific case is easy to handle based on the standard formula for computing roots of quadratic equations. However, we will appeal to a more general solution. It turns out that the task of factoring multivariate polynomials is solvable very efficiently in general. Three indepent works in the eighties, Grigoriev [43], Kaltofen [58], and Lenstra [65], addressed this problem. At the very least, they show that every polynomial in a constant number of variables can be factored in time polynomial in the degree of the input polynomial. This is sufficient for our purposes. But in case you care, Kaltofen and Trager [59] give much stronger results, essentially giving running times that are polynomial in the degree of the polynomials *and* the number of variables, provided you let them pick the representation of the polynomials. A note on the dependence on the field size. If the field is finite and has characteristic p (i.e., the field is an extension of $\mathbb{F}_p$ for some prime p) then the running time is polynomial in p and the logarithm of the field size. It can also be made purely polynomial in the logarithm of the field size, if we allow the algorithm to be randomized. All of this is inherited from the underlying algorithms for factoring univariate polynomials (see [16, 17, 25] etc.). Over the rationals, the algorithms run deterministically in time polynomial in the input size, again inheriting their performance from the univariate case, solved by Lenstra, Lenstra, and Lovasz [66].

Back to the algorithm at hand, notice that Step 1 requires that a solution exist, and Step 2 has a solution only if a possible factorization exists. Below we argue that the solutions do exist assuming our problem instance is fitting the promises of the toy problem.

12.3.2 Correctness of algorithm for the toy problem

We argue the correctness of the two steps in order.

Claim 12.8 *Under the hypothesis of the toy problem, a solution to Step 1 exists.*

We already argued this in motivating the algorithm, so we won't restate the proof. We move on to the more important claim, which states that any solution to Step 1, has the "right" factorization. This claim is important in that its proof generalizes naturally to the "non-toy case".

Claim 12.9 *If B and C are any solutions to Step 1 and p is any degree k polynomial that agrees with the given point set on $t \geq 2k+1$ points, then $y - p(x) \mid y^2 - B(x)y + C(x)$.*

Remark: Recall that the above claim is a specialization of "Bezout's theorem in the plane" — something we alluded to in our coverage of algebraic-geometry codes. This theorem (family of theorems, really) says that two algebraic curves in the plane can not share too many points without sharing a common component.rather Below we find a proof of this fact in this special case. We will generalize the proof a little more, but not completely, by the end of this lecture.

Proof Before we go into the proof of this particular case, let's see how we may prove that some polynomial of the form $y - p(x)$ divides some other polynomial. The classical way to do this, is to think of the polynomial $Q(x, y) = y^2 - B(x)y + C(x)$ as a polynomial $Q(y)$ only in y with coefficients that happen to be from $\mathbb{F}_q[x]$. To prove that some polynomial $y - \alpha$ divides $Q(y)$, we need to show that $Q(\alpha) = 0$. In our case, this says we should consider the quantity $g(x) = Q(x, p(x))$, which happens to be a polynomial in x and show that it is zero. Now the question has turned into one about showing a polynomial in x is zero. The standard method for this step is to bound the degree of g from above, and show that g has more zeroes than its degree. This is exactly what we do below.

Let $g(x) = Q(x, p(x))$. By inspection, we note that $g(x)$ is a polynomial of degree at most $2k$ in x. Now consider an index i such that $y_i = p(x_i)$. For such an i, we have

$$\begin{aligned} g(x_i) &= Q(x_i, p(x_i)) \quad \text{(By definition of } g\text{)} \\ &= Q(x_i, y_i)) \quad \text{(By choice of } i\text{)} \\ &= 0 \quad \text{(By property of } Q\text{, returned in Step 1)} \end{aligned}$$

Since $y_i = p(x_i)$ on at least $2k+1$ points, we have g has more roots than its degree, and thus $g(x) = 0$. Now, viewing Q as a polynomial in y with coefficients from $\mathbb{F}_q[x]$, we get that $Q(p(x)) = 0$, and so $y - p(x)$ divides $Q(y)$. ■

In our treatment of the "toy problem" above, very little was specific to having the solution come from one of two polynomials. We could easily handle 20 polynomials in the same way. Note however that the answers must come from a *finite* set of polynomials.

General case

We now move to the general case, (i.e., no longer sticking with the toy problem). Surprisingly enough, we won't change the algorithm, and even the analysis will change only mildly!

RS-list-decoder-1:

Given Set of n distinct pairs $\{(x_i, y_i) | i \in [n]\}$, and integers t and k.

Parameters: ℓ and D to be set later.

Step 1: Find a bivariate polynomial $Q(x, y)$ of degree ℓ in y and degree D in x such that:

- $Q(x_i, y_i) = 0 \quad \forall i$.
- $Q \not\equiv 0$.

Step 2: Factor $Q(x, y)$ and output the list of polynomials p of degree at most k provided:

- $y - p(x) \mid Q(x, y)$, and
- $p(x_i) = y_i$ for at least t values of $i \in [n]$.

It is evident that this still works efficiently — Step 1 is still solving a linear system and Step 2 is factoring a multivariate polynomial. What we need to argue is the correctness. The main concern in the correct is that we have no reason to believe a solution should exist for Step 1. It turns out that this is completely trivial — in a technical sense! We will show that really nice polynomials fitting the condition of Step 1 exist without any conditions on the points (only on the degree of Q).

Proposition 12.10 *Given any set $S = \{(x_i, y_i) | i \in [n]\}$, of n pairs from $\mathbb{F}_q \times \mathbb{F}_q$ there is a non-zero polynomial Q with $\deg_x(Q) \leq \frac{n}{\ell}$ and $\deg_y(Q) \leq \ell$, such that $Q(x_i, y_i) = 0$ for every $i \in [n]$.*

Proof Note that the linear system to be solved, to find Q is a *homogenous* linear system (which is why we go to the trouble of avoiding the all zero solution), and it has $(\ell + 1)(\lfloor n/\ell \rfloor + 1) > n$ coefficients, and only n constraints. So the system is "underdetermined" and thus has a non-trivial solution. ■

As a special case, we could ask for polynomials of degree $\sqrt{n}$ in x and y, and Proposition 12.10 asserts that such a polynomial fitting n points always exists. Notice that we are gaining upon univariate fits, by a whole $\sqrt{n}$ factor and this is where the crucial advantage comes from. Of course, the proof makes it totally clear that this Q-polynomial is very unlikely to be uniquely specified at this stage — but this is usually tackled by algebraic means.

We will prove a stronger version of Claim 12.9 below (actually, state a stronger version and let the reader prove it as an exercise).

Lemma 12.11 *If Q is any solution to Step 1 and p is any degree k polynomial that agrees with the given point set on $t > D + \ell \cdot k$ points, then $y - p(x) \mid Q(x, y)$.*

The proof is totally identical to that of Claim 12.9, with the minor change that the degree of the g polynomial (and hence the agreement required) go up to $D\ell$. The reader should verify the claim though!

Putting Proposition 12.10 and Lemma 12.11 together, we get the following theorem.

Theorem 12.12 *The algorithm* **RS-list-decoder-1** *with the setting $D = \sqrt{nk}$ and $\ell = \sqrt{n/k}$ solves the* **RS list decoding problem** *in polynomial time provided $t > 2\sqrt{kn}$.*

Proof By Proposition 12.10, a non-zero polynomial $Q(x, y)$ with $\deg_x(Q) \leq D$ and $\deg_y(Q) \leq D$ does exist. By Lemma 12.11, such a polynomial will have $y - p(x)$ as a factor, if p is any degree k polynomial that has agreement $D + \ell \cdot k = 2\sqrt{nk}$. ∎

This bound is not optimal. Notice that the "Johnson bound" from Theorem 12.7 only requires $t > \sqrt{kn}$ and so we are off by a factor of 2. We can actually get rid of one $\sqrt{2}$ factor in t by a close examination of the proof, and defining and controlling the degree of Q more carefully. We will see this at the beginning of the next lecture. The second $\sqrt{2}$ factor is more important. We will get rid of this in a much more involved algorithm later next lecture.

Bibliographic Notes

The first proof of the Johnson bound (inclusion-exclusion) is a "folklore" result in the CS literature (cf. [31, 40] etc.). The second proof of the Johnson bound is due to Jaikumar Radhakrishnan [90].

The algorithm for list-decoding of Reed-Solomon codes is from Sudan [109]. Most of the key insights, and in particular the solution to the "toy problem", is from the work of Ar, Lipton, Rubinfeld, and Sudan [5]. The main observation in [109] is the "triviality proposition", Proposition 12.10.

Chapter 13

6.897 Algorithmic Introduction to Coding Theory	October 31, 2001
Lecture 13	
Lecturer: Madhu Sudan	*Scribe: Hoeteck Wee*

Today's topics:

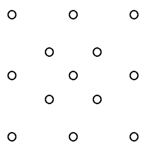
- Review of & improvement to list-decoding from last lecture.
- Mo' better list-decoding.
- Towards generalization of decoding algorithms.

13.1 Review of List-Decoding

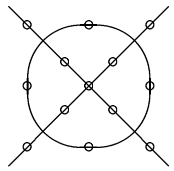
We will start by reviewing the list-decoding algorithm for Reed-Solomon codes from the last lecture, from a very high-level (or hand-wavy, if you prefer) perspective. The goal is to stress the simplicity of the algorithm. Then we will do a careful analysis of the algorithm to see how to get rid of one of the two $\sqrt{2}$ factors that separate the algorithm from the combinatorial bounds.

13.1.1 The algorithm on an example

To recall the idea of the algorithm, it is illustrative to consider an example. We will consider an example consisting of 13 points in the real (Euclidean) plane $(x_i, y_i), 1 \leq i \leq 13$, We will try to find all degree one polynomials passing through at least five points. Note that the numbers are chosen for illustration only. Certainly the ability to find degree one polynomials should not be considered the high-point of the algorithm, since "brute-force" algorithms could do this in quadratic time.



The main notion that distinguishes this approach from the traditional coding-theoretic approach is that instead of pulling out our calculators and interpolating, we are sitting back and contemplating. We try to find a nice algebraic curve that can explain all the 13 points. One such curve is drawn below.



The curve turn out to be $Q(x, y) = x^4 - y^4 - x^2 + y^2 = 0$ (assuming the central point is the origin and that the points are scaled appropriately). The picture above is the plot of all its zeroes. Clearly there are three distinct components to the set of all zeroes. These correspond to the factorization of Q whose factors are $x - y$, $x + y$ and $x^2 + y^2 - 1$. While the algebraic operation of factorization is needed to tell the components, the human eye doesn't need this calculation either - since the picture clearly distinguishes the three factors. In fact a close look at existing factorization algorithms for multivariate polynomials reveals that they are based on the geometric intuition that is evident in the picture above — they factors can be derived by fitting polynomials to zome zero of $Q(x, y)$ and its local naighborhood.

13.1.2 Weighted degree and some minor improvements

To get the best possible performance out of the algorithm above, we need to be more careful with the degree of the polynomial Q that we pick. For example, when we are trying to find a list of all degree 1 polynomials, p, it is better to pick Q of the smallest possible total degree, rather than the smallest individual degrees in x and y. Based on this idea, and keeping in mind that at some point we end up substituting for y a degree k polynomial p in x (and that we have no idea what this polynomial p is going to be), it is best to fit a polynomial Q that minimizes a "weighted" degree function. We define this weighted degree next:

Definition 13.1 *The* (k_1, k_2)-weighted degree *of a monomial* $x^i y^j$ *is the quantity* $i \cdot k_1 + j \cdot k_2$. *The* (k_1, k_2)-weighted degree *of a polynomial* $Q(x, y)$ *is the maximum, over all monomials in* Q *with non-zero coefficients, of their* (k_1, k_2)*-weighted degree.*

We now present the fully optimized algorithm for Reed-Solomon decoding:

RS-list-decoder-2:

Given Set of n distinct pairs $\{(x_i, y_i) | i \in [n]\}$, and integers t and k.

Parameter: D to be set later.

Step 1: Find a bivariate polynomial $Q(x, y)$ of $(1, k)$-weighted degree D such that:

- $Q(x_i, y_i) = 0 \quad \forall i$.
- $Q \not\equiv 0$.

Step 2: Factor $Q(x, y)$ and output the list of polynomials p of degree at most k provided:

- $y - p(x) \mid Q(x, y)$, and
- $p(x_i) = y_i$ for at least t values of $i \in [n]$.

The standard claims adapted for the case of weighted degrees are given below. We skip the proofs, but they are obvious.

Proposition 13.2 *For any set $S = \{(x_i, y_i) | i \in [n]\}$ of n points in $\mathbb{F}_q \times \mathbb{F}_q$, there exists a non-zero polynomial Q of $(1, k)$-weighted degree at most $\sqrt{2nk}$ such that $Q(x_i, y_i) = 0$ for all $i \in [n]$.*

Lemma 13.3 *If a polynomial $Q(x, y)$ of $(1, k)$-weighted degree D and the polynomial $y - p(x)$, with p having degree at most k, have more than D common zeroes, then $y - p(x)$ divides $Q(x, y)$.*

Putting the two together, we get:

Theorem 13.4 ([109]) *The algorithm* **RS-list-decoder-2** *with the setting $D = \sqrt{2nk}$ solves the* **RS list decoding problem** *in polynomial time provided $t > \sqrt{2kn}$.*

We now move towards a better algorithm for list-decoding. But first we try to motivate it. Why should we try to improve this algorithm? One of the main motivations is the goal to get the combinatorial results and algorithmic bounds to match. The algorithm above ends up solving the list-decoding problem exactly when the inclusion-exclusion bound works. What we want next is an algorithm that meets the graph-theoretic bound, i.e., when $t > \sqrt{kn}$.[1].

13.2 An improved list-decoder for RS codes

Before proceeding let us be warned that the "improvement" is only in the number of errors it can correct; and not in the running time. Our running times will actually get worse, but remain polynomially bounded.

[1] the case $t \leq \sqrt{kn}$ is not known

13.2.1 Weighted Decoding Problem

We will actually find an indirect route to our improvement. Rather than solving the standard list-decoding problem, we will explore a weighted list-decoding algorithm.

> **Weighted Reed Solomon List-Decoding Problem:**
> GIVEN: n distinct points $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ with corresponding integer weights $w_1, w_2, \ldots, w_n$ (specified in unary), degree bound k, and an agreement bound W (which replaces the parameter t in the previous problem statement).
> GOAL Find all degree k polynomials p such that $\sum_{i|p(x_i)=y_i} w_i > W$

To motivate this problem, let us note that this is in the spirit of the GMD (generalized minimum distance) decoding problem of Forney [55]. The main differences being that (1) Forney's problem associated a "penalty" with each pair (x_i, y_i) and the goal was to find polynomials that minimized the total penalties, and (2) Forney's problem expected the x_i's to be distinct and we won't. As in the case of the GMD algorithm of Forney, a solution to the weighted RS list-decoding problem has the potential to improve the decoding of concatenated codes and indeed several papers do just this [46, 63, 84].

The main technical result today will be the following:

Lemma 13.5 *There exists a polynomial time algorithm to solve the weighted RS list-decoding problem, provided* $W > \sqrt{2k \sum_{i=1}^{n} \binom{w_i+1}{2}}$

At a quick glance the lemma may not seem surprising. Indeed if we set all the w_i's to 1, then the lemma is identical to Theorem 13.4. To see that the lemma is doing something different, notice that the lemma has a peculiar behaviour with respect to the weights w_i's — it is not *scale invariant.* On the one hand if we take all the weights, w_i's as well as W, and say double them, then the solution set of polynomials does not change. However, the lemma may not guarantee to find the solutions with the smaller weights, but might be able to guarantee to find all solutions with the larger weight. Indeed setting all weights to 2, and setting $W = \sqrt{6kn}$ gives us a list of all polynomials that agree an unweighted set of n distinct pairs in $\sqrt{\frac{3}{2}kn}$ points. Sending w_i's to infinity gives us the result we want for this section, as proven below formally.

Theorem 13.6 ([45]) *There exists a polynomial time to solve the* **RS list decoding problem** *in polynomial time provided* $t > \sqrt{kn}$.

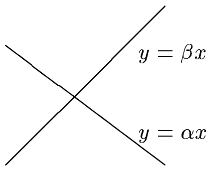
Proof Since t is an integer, we have $t \geq \lfloor \sqrt{kn} \rfloor + 1$. Set $w_1 = \cdots = w_n = w = 2nk$ (i.e., large enough that some later inequalities can be handled) and $W = t \cdot w$ and apply the algorithm from Lemma 13.5. If a polynomial p agrees with the point set on t points, then its weighted agreement is at least tw and Lemma 13.5 guarantees that this polynomial will be included in the output list provided $tw > \sqrt{nkw(w+1)}$, or in other words, provided $t > \sqrt{nk(1 + 1/w)} = \sqrt{nk + \frac{1}{2}}$. It can be easily verified that $\sqrt{nk + \frac{1}{2}} < \lfloor \sqrt{nk} \rfloor + 1 \leq t$ and so the conditions required for applying Lemmm 13.5 are indeed satisfied. ■

Thus to conclude our goal it suffices to build the algorithm that Lemma 13.5 claims to exist.

13.2.2 Algorithm

Our basic problem at this stage is to figure out how to make some algebraic sense of the weights that are given to us. I.e., how to modify the definition of the Q-polynomial so that points with larger weights constraint it more than the points with smaller weight, or to make it clear that Q "respects" points of larger weight more than the points of smaller weight. A natural idea, if we can make any formal sense of it, is to make the polynomial go through points as many times as their weight. So Q should pass through a point with weight 2 twice, while it should pass through a point of weight 10, ten times!

Let us first see an example of what it means to have a curve passing through the point $(0, 0)$ twice.



The equation of this curve is $(y-\alpha x)(y-\beta x) = 0$, or $y^2-(\alpha+\beta)xy+\alpha\beta x^2 = 0$. Intuitively, it is clear what it means for a curve to pass through the origin twice, and that the example above indeed does so. Analytically, this may be explained in terms of the partial derivatives of Q at $(0, 0)$. However, since we are working over finite fields of potentially small characteristic, the partial derivatives may not be informative enough. So we'll describe the algebraic definition. To guess this definition, let us look at two familiar examples. We know what it means for a curve to pass through $(0, 0)$ once - it means that the coefficient of the constant term must be zero. In the example above, where the curve passes through the origin twice, the constant term as well as the monomials of degree one (i.e., x and y) have zero as coefficients. Deducing from these cases, we guess that a polynomial Q passes through $(0, 0)$ at least r times if the monomials of total degree less than r have a zero coefficient.

Now how can we extend the definition to an arbitrary point $(\alpha, \beta) \in \mathbb{F}_q \times \mathbb{F}_q$? The natural way to do this is to shift the system so that (α, β) becomes the origin. Thus we arrive at the following *definition*. (We stress that nothing is being proved so far - we are just guessing a definition; later we will need to prove that this actually works!)

Definition 13.7 *A polynomial $Q(x, y)$ passes through the point (α, β) at least r times if all the coefficients of monomials of total degree less than r of the polynomial $Q_{\alpha,\beta}(x, y) \stackrel{\text{def}}{=} Q(x + \alpha, y + \beta)$ are zero.*

With the above definition in hand, we now describe the optimized algorithm for weighted polynomial fitting:

RS-list-decoder-3:

Given Set of n distinct pairs $\{(x_i, y_i) | i \in [n]\}$, integer weights $w_1, \ldots, w_n$ and an integer degree bound k.

Parameter: D to be set later.

Step 1: Find a bivariate polynomial $Q(x, y)$ of $(1, k)$-weighted degree D such that:

- For every $i \in [n]$, the polynomial Q passes through the point (x_i, y_i) at least w_i times.
- $Q \not\equiv 0$.

Step 2: Factor $Q(x, y)$ and output the list of polynomials p of degree at most k provided:

- $y - p(x) \mid Q(x, y)$, and
- $p(x_i) = y_i$ for at least t values of $i \in [n]$.

To see that the new algorithm still runs in time polynomial in $\sum_{i=1}^n w_i$ and D, we need to show that Step 1 is solvable in so much time. The critical part of this is to show that the conditions being enforced here are still linear and homogenous. Note that the condition being enforced for Q to pass through the origin multiple times *are* linear and homogenous. What we need to establish is that the shifting of a polynomial leads to coefficients that are linear in the coefficients of Q. This is established below.

Proposition 13.8 *The coefficients of the polynomial $Q_{\alpha,\beta}(x, y) \stackrel{\text{def}}{=} Q(x + \alpha, y + \beta)$ are linear in the coefficients of Q.*

Proof We will prove this by giving the coefficients of $Q_{\alpha,\beta}$ explicitly. This is just an ordinary calculation. We get the coefficient of $x^i y^j$ in $Q_{\alpha,\beta}$ equals $\sum_{i' \geq i, j' \geq j} \binom{i'}{i}\binom{j'}{j} \alpha^{i'-i} \beta^{j'-j} q_{i'j'}$ where $Q(x, y) = \sum_{i,j} q_{ij} x^i y^j$. ■

Now we can go about the proof in our usual manner. We start with the observation that a non-zero polynomial does exist, provided we pick D to be large enough.

Proposition 13.9 *For any sequence of distinct points $(x_1, y_1), \ldots, (x_n, y_n) \in \mathbb{F}_q \times \mathbb{F}_q$, and non-negative integral weights $w_1, \ldots, w_n$, there exists a non-zero polynomial Q of $(1, k)$-weighted degree at most $\sqrt{2k \sum_{i=1}^n \binom{w_i+1}{2}}$ such that Q passes through (x_i, y_i) at least w_i times for every $i \in [n]$.*

Proof Each condition of the form Q must pass through (x_i, y_i) at least w_i times, turns into the condition that $\binom{w_i+1}{2}$ coefficients of Q_{x_i,y_i} must be zero. In turn these turn into $\binom{w_i+1}{2}$ homogenous linear conditions on the coefficients of Q (by Proposition 13.8). Thus all in all we have $\sum_{i=1}^n \binom{w_i+1}{2}$ homogenous linear constraints. A simple calculation shows that the number of coefficients of $(1, k)$-weighted degree at most $\sqrt{2k \sum_{i=1}^n \binom{w_i+1}{2}}$ is more than $\sum_{i=1}^n \binom{w_i+1}{2}$ and thus a non-zero solution to the homogenous linear system exists. ■

So far we have proved that multiplicities can be handled. But we haven't related it to the problem at hand - why does making Q pass through a point w_i times, help weight this point w_i times?

We start to prove this below:

Lemma 13.10 *If $Q(x, y)$ passes through a point (α, β) at least w times and p is any polynomial such that $p(\alpha) = \beta$, then $(x - \alpha)^w$ divides the polynomial $g(x) \stackrel{\text{def}}{=} Q(x, p(x))$.*

Proof First by translation, we will turn this into a question about $(0,0)$. Let $Q_{\alpha,\beta}(x,y) = Q(x+\alpha, y+\beta)$. Let $p_{\alpha,\beta}(x) = p(x+\alpha) - \beta$. Since $p(0) = 0$, we have x divides $p_{\alpha,\beta}(x)$. Let $g_{\alpha,\beta}(x) = Q_{\alpha,\beta}(x, p_{\alpha,\beta}(x)) = Q(x+\alpha, p(x+\alpha)) = g(x+\alpha)$. Thus $(x-\alpha)^w$ divides $g(x)$ if and only if x^w divides $g(x+\alpha) = g_{\alpha,\beta}(x)$. In other words we wish to show that the monomials of degree less than w in $g_{\alpha,\beta}$ have zero coefficient. But this is syntactically true. $Q_{\alpha,\beta}$ has no terms of total degree less than w and we are substituting for y a polynomial $p_{\alpha,\beta}(x)$ that is itself a mutliple of x. Thus all terms have degree at least w. ■

The rest of what we want is now straightforward. We will show that if for some degree k polynomial p, the weights of points (x_i, y_i) where $p(x_i) = y_i$ sum to more than the $(1,k)$-weighted degree of Q, then $y - p(x)$ must divide Q.

Lemma 13.11 *Let $Q(x,y)$ be any polynomial of $(1,k)$-weighted degree D that satisfies the conditions of Step 1. Suppose p is a degree k polynomial such that $\sum_{\{i\in[n] \mid p(x_i)=y_i\}} w_i > D$, then $y - p(x)$ divides $Q(x,y)$.*

Proof Let $g(x) = Q(x, p(x))$. Then g has degree at most D. We wish to show that g is identically zero. For a change we won't do this by showing that it has more zeroes than D but by showing that a polynomial of degree greater than D divides it (and this can happen only to the zero polynomial).

Let $S = \{i \in [n] \mid y_i = p(x_i)\}$. Note first that for $i, j \in S$, $x_i \neq x_j$ since otherwise we would have $y_i = p(x_i) = p(x_j) = y_j$ and thus $(x_i, y_i) = (x_j, y_j)$. Next note that for every $i \in S$, we have $(x - x_i)^{w_i}$ divides $g(x)$ (by Lemma 13.10. Putting these together, we get $\prod_{i\in S}(x - x_i)^{w_i}$ divides $g(x)$. (Here we are using the fact that the x_i's are distinct. Really what we should observe first is that the least common multiple of the $(x - x_i)^{w_i}$'s divides g. Then using the distinctness of the x_i's we should claim that the product divides g. But this would require defining least common multiple etc.) Now we are done since the polynomial $\prod_{i\in S}(x - x_i)^{w_i}$ has degree $\sum_{\{i\in[n] \mid p(x_i)=y_i\}} w_i > D$, thus yielding that $g \equiv 0$. ■

We may now have a proof of Lemma 13.5 with the algorithm described explicitly, as summarized below:

Theorem 13.12 ([45]) *The algorithm* **RS-list-decoder-3** *with the setting $D = \sqrt{2k \sum_{i=1}^{n} \binom{w_i+1}{2}}$ solves the* **Weighted RS list decoding problem** *in polynomial time provided $W > \sqrt{2k \sum_{i=1}^{n} \binom{w_i+1}{2}}$.*

Recall that as this concludes the proof of Theorem 13.6 which may be considered the main theorem of this lecture.

13.3 Towards Generalization of Decoding Algorithms

We now move towards an abstraction of the algorithm above. We motivate by considering the decoding problem for a fairly different code — the Chinese Remainder Code — lecture. We already defined this code in a previous lecture. We start by recalling the definition.

13.3.1 Chinese Remainder Codes and Decoding

We start with the classical "Chinese Remainder Theorem".

Theorem 13.13 (Chinese Remainder Theorem (CRT)) *Let $p_1, p_2, \ldots, p_k$ be positive integers that are pairwise relatively prime. Then, the map* CRT *that takes $m \in \mathbb{Z}_K$ to the k-tuple $\langle m(\bmod p_1), \ldots, m(\bmod p_k)\rangle$, where $K = \prod_{i=1}^{k} p_i$, is a bijection. Furthermore, there are polynomial time algorithms to compute* CRT *as well as its inverse.*

In other words, $m \in \mathbb{Z}_K$ is fully specified given its residues modulo $p_1, \ldots .p_k$. The CRT code is based on the observation that specifying $m \in \mathbb{Z}_K$ modulo a large number of relatively prime integers (larger than the minimum required) gives a redundant representation of m. We formalize this next.

Let $p_1 < p_2 < \ldots < p_n$ be primes, and let $K = \prod_{i=1}^{k} p_i$ and $N = \prod_{i=1}^{n} p_i$ where $k < n$. Then, $\langle m(\bmod p_1), \ldots, m(\bmod p_n)\rangle$ is a redundant *encoding* of m. In particular, given any k out of the n possible residues, we can recover m uniquely. It follows that the encoding of any two message must differ in at least $k-1$ positions, so this encoding yields a "$(n, k, n-k+1)$ code". This code motivates the natural (list-)decoding problem.

Problem Given $p_1 < \ldots < p_n$, $k < n$ and a codeword corresponding to the residues $\langle r_1, \ldots, r_n\rangle$.

Task Find a $m \in \{0, 1, \ldots, \prod_{i=1}^{k} p_i - 1\}$ such that $m(\bmod p_i) = r_i$ for many values of i.

> **CRT List-Decoding Problem:**
> Given: $p_1 < \ldots < p_n$, integer K, a vector of residues $\langle r_1, \ldots, r_n\rangle$, and an agreement parameter t.
> Goal Find a list of all $m \in \mathbb{Z}_K$ such that, $m(\bmod p_i) = r_i$ for at least t values of $i \in [n]$.

Our plan to get a decoding algorithm for this problem is by the following sequence of steps: First, we will find a common generalization of the Reed-Solomon and CRT codes; Next, we will "lift" our algorithm for decoding for RS decoding to this common generalization. This won't quite give us an algorithm for the common generalization, but an outline of one. Finally, we will specialize the "algorithm" to the case of CRT codes and then prove its efficiency and correctness. Today we will move towards the first of these steps.

13.3.2 Algebra Review

The Chinese Remainder Codes and the Reed-Solomon Codes may be viewed as examples of a more general class of error-correcting codes based "ideals" in commutative rings. We start by recalling some of the jargon.

- A commutative ring R is an *integral domain* if it does not contain any zero divisors, i.e., there do not exist non-zero elements $a, b \in R$ such that $a \cdot b = 0$.
- An *ideal* I in a ring R is a subset of R closed under addition and general multiplication (by any element in R), i.e., for any $a, b \in I$ and $r \in R$, it is the case that $a + b, ar \in I$.

Ideals in commutative rings are nice objects to work with. First, they capture the "modulo" operation (as formalized in the next definition). The collection of ideals in a ring is interesting to study since they are closed under some natural (and some slightly concocted) operations.

- Let I be an ideal of a ring R. The cosets of the additive subgroup of I^+ of R^+ are the subsets $a+I, a \in R$. The set of cosets R/I forms a group with a ring structure inherited from R. This ring is known as the *quotient ring* of R over the ideal I.

The quotient ring R/I of a ring R over an ideal I forms the basis of a modular reduction. This is the reduction that maps an element $a \in R$ to the coset $a+I$. However, we (especially the computer scientists among us) are used to thinking about modular reductions in slightly different language. First, given an ideal I, we usually pick a set $\Sigma \subseteq R$ of representatives for the quotient ring R/I, i.e., for every $a \in R$, there exists exactly one $b \in \Sigma$ such that $b \in a + I$. Note that the one-to-one correspondence between R/I and Σ gives a ring structure to Σ. Having picked such a set, we use a slightly different notation $a(\text{mod} I)$, or just $a(I)$, to denote the element $b \in \Sigma$ such that $b \in a + I$. If I is the ideal generated by a single element $p \in R$ (i.e., $I = \{pr \mid r \in R\}$), then this gives the familiar $(\text{mod} p)$ operation.

The choice of repreesentatives does not alter the algebra underneath. So we will often leave this choice unspecified. However representations are always important to *computation*. Unless we fix a set Σ and representations of elements in Σ, computational aspects can *not* be clarified. We now go on to other properties of ideals.

Proposition 13.14 *Let I and J be ideals of a ring R. Then their* sum, *denoted $I+J$,* intersection, *denoted $I \cap J$, and product, denoted IJ and defined to be $IJ = \{\sum_i a_i b_i \mid a_i \in I, b_i \in J\}$, are also ideals of R.*

13.3.3 Ideal Error-Correcting Codes

We now describe how the language of ideals can be used to define error-correcting codes.

Definition 13.15 *An* ideal error-correcting code *C is given by an integral domain R, a sequence of n ideals $I_1, \ldots, I_n \subseteq R$ (and associated sets of representatives $\Sigma_1, \ldots, \Sigma_n$), and a* message space *$M \subseteq R$. The canonical encoding function for C maps the message $m \in M$ to the sequence $\langle m(I_1), \ldots, m(I_n)\rangle$.*

Some of the previous choice of notation may become clearer now: Usually $\Sigma_1 = \cdots = \Sigma_n = \Sigma$ and then the code maps M to Σ^n. But is other cases Σ_i's may be different (as in the case of the CRT code). For practical usage Σ_i's better be finite. However the mathematics does not enforce such a restriction. We now see how Reed-Solomon codes and CRT codes are both special cases of ideal error-correcting codes.

1. Chinese Remainder Codes. This corresponds to $R = \mathbb{Z}$, $I_1 = p_1\mathbb{Z}, \ldots, I_n = p_n\mathbb{Z}$, and $M = \{0, 1, \ldots, \prod_{i=1}^k p_i - 1\}$. The representative set Σ_i representing $\mathbb{Z}/p_i\mathbb{Z}$ is the natural one, i.e., $\Sigma_i = \mathbb{Z}_{p_i} = \{0, \ldots, p_i - 1\}$.
2. Reed-Solomon Codes. This corresponds to $R = \mathbb{F}_q[x]$, $I_1 = (x - \alpha_1), \ldots, I_n = (x - \alpha_n)$ where $\alpha_1, \ldots, \alpha_n$ are the n distinct elements of $\mathbb{F}_q$ that we pick to evaluate the polynomial. M is

the set of polynomials in R of degree less than k. The choice of representatives is again the natural one with $\Sigma_1 = \cdots = \Sigma_n = \mathbb{F}_q$, where $\mathbb{F}_q$ stands for the set of degree zero polynomials in R.

In the next lecture we will see how to lift our decoding algorithm(s) and then specialize them to get a decoding algorithm for the CRT code.

Bibliographic notes

The optimized version of the Reed-Solomon decoding algorithm that worked if the agreement parameter $t > \sqrt{2kn}$ is from [109]. A careful analysis of this algorithm for every choice of rate is given in [110]. The improved algorithm working if $t > \sqrt{kn}$ is due to Guruswami and Sudan [45].

Chinese remainder codes have been studied for a long time now. Early papers on this code include those of Asmuth and Bloom [9] and Watson [121]. Even some books have been written on this topic (see Krishna et al. [64] and Soderstrand et al. [107]). References for decoding algorithms for this code include [76, 39, 22, 44]. We will discuss their contributions in further detail in the next lecture. For now, we mention that the formalization of "ideal error-correcting codes" is from [44].

Chapter 14

6.897 Algorithmic Introduction to Coding Theory November 5, 2001

Lecture 14

Lecturer: Madhu Sudan *Scribe: Daniel Preda*

Topics for today:

- Ideal codes
- Abstract list-decoding
- Decoding of CRT codes

14.1 Ideal Error-Correcting Codes

Let us recall the following definition of an ideal error-correcting code from the last lecture.

Definition 14.1 *An* ideal error-correcting code *C is given by an integral domain R, a sequence of n ideals $I_1, \ldots, I_n \subseteq R$ (and associated sets of representatives $\Sigma_1, \ldots, \Sigma_n$), and a* message space *$M \subseteq R$. The canonical encoding function for C maps the message $m \in M$ to the sequence $\langle m(I_1), \ldots, m(I_n)\rangle$.*

We will show to abstract the list-decoding algorithm for Reed-Solomon codes in this framework, and then apply it to the CRT codes. First lets recall how Reed-Solomon and CRT codes fit in this framework.

Reed-Solomon codes

We saw how to express Reed-Solomon codes in this framework:

- The ring $R = \mathbb{F}_q[x]$.

- The ideals $I_1, \dots, I_n$ being given by $I_i = (x - \alpha_i)$, where $(x - \alpha_i)$ represents the ideal of polynomials that are multiples of $x - \alpha_i$, i.e., $(x - \alpha_i) = \{p(x) \cdot (x - \alpha_i) \mid p(x) \in \mathbb{F}_q[x]\}$. The quotient ring $R/I_i \cong \mathbb{F}_q$ where the representatives are the set of polynomials of degree zero.
- The message space $M \subseteq R$ is the set of polynomials of degree less than k.

The ith coordinate of the encoding of a message polynomial m is the unique polynomial $m_i(x)$ of degree zero that is contained in the coset $m(x) + \{p(x) \cdot (x - \alpha_i) \mid \alpha_i \in \mathbb{F}_q\}$, which turns out to be the constant $m(\alpha_i)$. Thus this definition is consistent with our usual definition of Reed-Solomon codes.

CRT Codes

In the case of the CRT codes we have:

- R is *the* integral domain, i.e., the integers.
- $I_1, \dots, I_n$ are the ideals $p_1\mathbb{Z}, \dots, p_n\mathbb{Z}$ where $p_1, \dots, p_n$ are n relatively prime integers, with Σ_i being $\mathbb{Z}_{p_i}$ the set of non-negative integers of value less than p_i.
- The message space $M \subseteq \mathbb{Z}$, is the set $M = \{0, 1, ..., K - 1\}$ (where for convenience we pick $K = \prod_{i=1}^{k} p_i$).

Once again a message $m \in \mathbb{Z}_K$ is mapped to the vector $\langle m \pmod{p_1}, \dots, m \pmod{p_n}\rangle$.

Size and distance of ideal error-correcting codes

While the abstraction so far describes how to construct the ideal error-correcting codes, it doesn't shed light on the minimum distance of the codes constructed this way. Here we give an informal discussion of a notion that is used to prove distance properties (as also specify decoding algorithms). To argue about distance properties, we associate with elements of the ring, a "size". E.g. the size of an integer will simply be its absolute value; the size of a polynomial of degree k over $\mathbb{F}_q$ will be q^{k+1}, etc. In general the size function must satisfy some axioms such as $\text{size}(a + b) \leq \text{size}(a) + \text{size}(b)$; and $\text{size}(ab) = \text{size}(a)\text{size}(b)$ etc. Here we won't go over this concept rigorously — though it can be done [44]. Instead we will keep this property in mind loosely and become precise in the case of Reed-Solomon codes and CRT codes. In the rigorous case, one also associates a size with ideals in R — the size of I being the size of the smallest non-zero element in I. If the size of the ideal also satisfies some size axioms, then minimum distance of the code can be established formally.

14.2 Abstracting the Reed-Solomon decoding algorithm

We start by recalling the (first) list-decoding algorithm for decoding Reed-Solomon codes and slowly translating it so it sounds like a generic decoding algorithm for ideal error correcting codes. The table below is best read row-by-row.

	RS decoder	Abstraction
Given:	$\alpha_1, \ldots, \alpha_n$ $y_1, \ldots, y_n i \in \mathbb{F}_q$	$I_1, \ldots, I_n$ $y_1, \ldots, y_n$, with $y_i \in \Sigma_i$
Find:	All polynomials p of degree less than k such that $p(x_i)$ for many values of $i \in [n]$	All messages $m \in M$ such that $m \pmod{I_i} = y_i$ for many values of i.
Step 1:	Find $Q(x, y) \in \mathbb{F}_q[x, y]$ such that	Since $R = \mathbb{F}_q[x]$, it makes sense to interpret $F_q[x, y]$ as $R[y]$ and thus the sentence on the left translates to "Find $Q(y) \in R[y]$ such that"
	Q non-zero	"Q non-zero"
	$\deg_y(Q)$ small	"$\deg_y(Q)$ small"
	$\deg_x(Q)$ small	Notice that R is not necessarily a ring over some variable; so we can't use $\deg_x$; instead we should ask that the coefficients of Q be small. Thus we get "Coefficients of Q have small size".
	$Q(\alpha_i, y_i) = 0$ for all i	This is a little trickier — but guessing that α_i is somehow related to I_i and y_i remains as it is, we come up with "$Q(y_i) \in I_i$ for all i" — this guess can be verified to be right.
Step 2:	Factor Q and report all polynomials $p(x) \in M$ such that $y - p(x)$ divides Q	The translation seems obvious: "Factor Q and report all messages $m \in M$ such $y - m$ divides Q." The important thing to note is that it is a fortunate coincidence that this is actually well-defined. In particular $R[y]$ is a unique factorization domain (upto multiplications by elements of R). (This need not have been the case if R were not an integral domain, or so we believe.)

We summarize the resulting "high-level algorithm" for decoding ideal codes below. In the explanation, we make a slight syntactic change: Instead of saying $Q(y_i) \in I_i$, we define a new ideal $J_i \subseteq R[y]$ consisting of polynomials Q such that $Q(y_i) \in I_i$. It can be verified that J_i is just the sum of the ideals I_i (viewed as an ideal of $R[y]$) and $(y - y_i)$. While the change is syntactic, it allows us to take a few extra steps in these notes (which we didn't manage in the lecture).

Ideal List-Decoder:

Given: n relatively prime[1], ideals $I_1, \ldots .I_n$ in a ring R, a message space $M \subseteq R$ and n elements

[1] Ideals I and J are relatively prime if $I \cap J = IJ$.

$y_1, \ldots, y_n \in R$.

Unspecified parameters: ℓ and B, whose choices depend on the ring R, its size structure, the message space M, etc.

Step 0: Define ideals $J_1, \ldots, J_n \subseteq R[y]$ to be $J_i = (I_i) + (y - y_i)$.

Step 1: Find non-zero $Q \in R[y]$ such that $\deg(Q) \leq \ell$ and every coefficient of Q is has size at most B such that $Q \in \cap_{i=1}^n J_i$.

Step 2: Factor Q and output all $m \in M$ such that $y - m$ divides Q.

To specify the parameters, and to show how to achieve the steps above effectively, we need to know more about the ring R. Below we give an example of how we can apply the above structure to get a CRT list-decoder. We will still end up doing a lot of work to prove correctness and analyze bounds. But before going on to the case of the CRT codes, lets describe a "minor" twist to the above algorithm. Suppose the ideals $I_1, \ldots, I_n$ had not been relatively prime? What would the right definition have been? The one place where we use the relative primality (of J's but they inherit it from the relative primality of the I's) is in saying the Q should lie in the intersection of the ideals $J_1, \ldots, J_n$. If relative primality doesn't hold, we should probably revert to what seems to be the more correct statement to insist on — i.e., Q should lie in the *product* of $J_1, \ldots, J_n$. Thus we get the following modified algorithm for decoding ideal codes.

Ideal List-Decoder-2:

Given: n ideals $I_1, \ldots .I_n$ in a ring R, a message space $M \subseteq R$ and n elements $y_1, \ldots, y_n \in R$.

Unspecified parameters: ℓ and B, whose choices depend on the ring R, its size structure, the message space M, etc.

Step 0: Define ideals $J_1, \ldots, J_n \subseteq R[y]$ to be $J_i = (I_i) + (y - y_i)$.

Step 1: Find non-zero $Q \in R[y]$ such that $\deg(Q) \leq \ell$ and every coefficient of Q is has size at most B such that $Q \in \prod_{i=1}^n J_i$.

Step 2: Factor Q and output all $m \in M$ such that $y - m$ divides Q.

So what does this "minor change" buy us? Well, it turns out we get the weighted list-decoding algorithm immediately. Since we don't make any restrictions on the I_i's, we can repeat each one w_i times if needed. The change will be in the analysis, in which we need to argue that low-degree polynomials of small degree still exist in the ideal $\prod_{i=1}^n J_i$. We didn't argue this generically even in the simpler case; We certainly won't do it now! But in specific examples it can be done!

We now specialize the simpler algorithm above for the case of CRT codes and analyze it.

14.3 Decoding CRT codes

The algorithm above, applied to the case of CRT codes, becomes the following:

CRT-decoding-1:

Given: n relatively prime integers $\langle p_1 < \cdots < p_n \rangle$, residues $\langle r_1, \ldots, r_n \rangle$ with $r_i \in \mathbb{Z}_{p_i}$, and parameters k and t.

Goal Output all $m \in \mathbb{Z}_K$, where $K = \prod_{i=1}^{k} p_i$, such that $m = r_i \pmod{p_i}$.

Parameters: B, ℓ.

Step 1: Find $Q \in \mathbb{Z}[y]$ such that:

- $Q \not\equiv 0$.
- $\deg(Q) \leq \ell$.
- Coefficients of Q are at most B in absolute value.
- $Q(r_i) = 0 \pmod{p_i}$ for every $i \in [n]$.

Step 2: Factor Q and report all integers m such that $y - m$ divides Q and $m = r_i \pmod{p_i}$ for at least t values of $i \in [n]$.

The conditions $Q(r_i) = 0 \pmod{p_i}$ are not so nice to deal with. So, we modify them to an equivalent form which is easier to deal with: Let $N = \prod_{i=1}^{n} p_i$ and let R be the unique element of $\mathbb{Z}_N$ satisfying $R = r_i \pmod{p_i}$ for every $i \in [n]$ (as guaranteed to exist by the CRT theorem). Note that

$$Q(r_i) = 0 \pmod{p_i}, \quad \forall i \in [n], \qquad \Leftrightarrow \qquad Q(R) = 0 \pmod{N}.$$

Thus our goal in Step 1 is to find a polynomial with small coefficients that is zero modulo N at just one point. We will show below that this problem can be reduced to a "shortest vector problem (SVP)" in integer lattices. We define SVP below.

> **Shortest Vector Problem (SVP):** Approximation version with parameter $\alpha : \mathbb{Z}^+ \to \mathbb{Z}+$
> GIVEN: Integer $d \times n$ matrix $\mathbf{B}$ over the integers, integer bound L, with the promise that there exists a vector $\mathbf{x}' \in \mathbb{Z}^d$ such that $\|\mathbf{x}'\mathbf{B}\| \leq L/\alpha(d)$.
> FIND: A $\mathbf{x} \in \mathbb{Z}^d$ such that $\|\mathbf{x}\mathbf{B}\| \leq L$.

The set of vectors $\mathbf{x}\mathbf{B}$ form an integer "lattice" in d-dimensional space. The task of finding short vectors in such lattices is a classical and well-studied problem. It was long conjectured to be NP-hard to solve exactly, i.e., when $\alpha(\cdot) = 1$. Finally, in a breakthrough result, shown to be true under randomized reductions by Ajtai [3]. Subsequently Micciancio [81] showed that the approximation version with $\alpha(\cdot) = \alpha < \sqrt{2}$ is also NP-hard under randomized reductions. Note that the problem becomes potentially easier as α becomes larger. We are interested in positive results. A seminal result by Lenstra, Lenstra, and Lovasz [66] showed that this problem is tractable for some finite function $\alpha(\cdot)$ (i.e., $\alpha(d) < \infty$ for every d). Their theorem is given below.

Theorem 14.2 ([66]) *The SVP problem with approximation parameter $\alpha(d) = 2^d$ is solvable in polynomial time in the length of the input.*

We now show how to find a polynomial as required in Step 1, using Theorem 14.2.

Lemma 14.3 *Given t-bit integers R and N and a degree bound ℓ and coefficient bound B, a non-zero polynomial Q of degree at most ℓ and coefficients at most B in absolute value satisfying $Q(R) = 0 \pmod{N}$ can be found in polynomial time (in t and ℓ) provided such a polynomial with coefficients at most $B/(\ell 2^\ell)$ exists.*

Proof We will set up a lattice generated by the matrix $\mathbf{B}$ such that short vectors in this lattice correspond to polynomials Q such that $Q(R) = 0 \pmod{N}$. The intuition behind why this should be feasible is the following alternate characterization of a lattice: A lattice in $\mathbb{Z}^d$ is a subset $L \subseteq \mathbb{Z}^d$ that is closed under scalar multiplication and addition, i.e., if $\mathbf{x}, \mathbf{y} \in L$ and $\lambda \in \mathbb{Z}$ then $\lambda\mathbf{x}, \mathbf{x}+\mathbf{y} \in L$. If we consider the set of coefficient vectors $\langle q_0, \ldots, q_\ell$ of polynomials Q such that $Q(R) = 0 \pmod{N}$, then indeed such a set is closed under addition and scalar multiplication. Thus we see that the set of such vectors forms a lattice in $\mathbb{Z}^{\ell+1}$ and it is only a matter of perseverance to find an explicit basis. Below we give an explicit basis, using the fact that any such polynomial can be expressed as $p(x)(x-R) + c \cdot N$, where $p(x)$ has degree at most $\ell - 1$. Consider the $(\ell+1) \times (\ell+1)$ matrix

$$\mathbf{B} = \begin{bmatrix} N & 0 & 0 & 0 & \ldots & 0 \\ -R & 1 & 0 & 0 & \ldots & 0 \\ 0 & -R & 1 & 0 & \ldots & 0 \\ 0 & 0 & -R & 1 & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & -R & 1 \end{bmatrix}.$$

We leave it to the reader to verify that the set of vectors $\langle q_0, \ldots, q_\ell \rangle$ that can be expressed as $\mathbf{xB}$ for $\mathbf{x} \in \mathbb{Z}^{\ell+1}$ is the set of coefficients of all polynomials $Q(y) = \sum_{i=0}^{\ell} q_i y^i$ such that $Q(R) = 0 \pmod{N}$. By Theorem 14.2, a vector of ℓ_2-norm, and thus all coefficients, at most B can be found in polynomial time in this lattice, provided a vector of ℓ_2 norm at most $B/2^{\ell+1}$ exists in this lattice. In turn such a vector does exist provided a vector all of whose coordinates are at most $B/((\ell+1)2^{\ell+1})$ exists. in this lattice. The lemma is thus proven. ■

We now move towards the efficiency of the second step. Once again this follows from the same seminal work of Lenstra, Lenstra, and Lovasz [66] mentioned above, but actually is a completely different result within this work (which uses the algorithm from Theorem 14.2 as a subroutine).

Theorem 14.4 *Given a polynomial $Q \in \mathbb{Z}[y]$ of degree ℓ with t-bit coefficients, Q can be factored over integers (modulo the factorization of the leading term) in time polynomial in ℓ and t.*

As a consequence we have that Step 2 of the algorithm above can also be executed in polynomial time. We only need to show that a good solution exists to Step 1 and then reason about the parameters for which the decoding algorithm works.

In the next lemma we prove that a very short solution to Step 1 exists, and thus Theorem 14.2 can find a moderately short solution.

Lemma 14.5 *A polynomial Q satisfying the conditions of Step 1 with coefficients of absolute value at most $B' \leq N^{1/\ell}$ exists.*

Proof Consider the function $f : \mathbb{Z}_{B'}^{\ell+1} \to \mathbb{Z}_N$ given by $f(q_0, \ldots, q_\ell) = \sum_{i=0}^{\ell} q_i R^i \pmod{N}$. The range of f is finite and has smaller cardinality than the domain (i.e., $N < (B')^{\ell+1}$). Thus there must exist distinct vectors $\mathbf{a}, \mathbf{b} \in \mathbb{Z}_{B'}^{\ell+1}$ such that $f(\mathbf{a}) = f(\mathbf{b})$. We conclude that $f(\mathbf{a} - \mathbf{b}) = 0 \pmod{N}$. Thus we have a non-zero polynomial $Q(y) = \sum_{i=0}^{\ell} (a_i - b_i) y^i$ such that $Q(R) = 0 \pmod{N}$. Since $a_i, b_i \in \mathbb{Z}_{B'}$, we have $|a_i - b_i| \leq B'$. ■

Next, we prove the usual next lemma — that any solution to Step 1 suffices.

Lemma 14.6 *For relatively prime integers $p_1 < \cdots < p_n$, let $N = \prod_{i=1}^n p_i$, and $T = \prod_{i=1}^t p_i$. If $Q(y)$ is any polynomial of degree ℓ with coefficients at most B such that $Q(R) = 0 \pmod{N}$, and $m \le K$ is any integer such that $m = R \pmod{p_i}$ for t values of i and $(\ell+1)BK^\ell < T$, then $y - m$ divides $Q(y)$.*

Proof The lemma turns out to be harder to state than to prove! Let $S = \{i | m = R \pmod{p_i}\}$. Let $T' = \prod_{i \in S} p_i$. Note that $T' \ge T$. By the conditions on Q, we have for every $i \in S$,

$$Q(m)(\bmod p_i) = Q(R)(\bmod p_i) = 0(\bmod p_i).$$

We conclude that $Q(m) = 0(\bmod T')$. But on the other hand we have $|Q(m)| \le \sum_{i=0}^\ell BK^i \le (\ell+1)BK^\ell < T \le T'$. Combining the two conditions, we get $Q(m) = 0$, and thus $y - m$ divides $Q(y)$. ■

Now we seem to have all the ingredients — all we need is to find out what they really mean - the number of parameters flowing around is huge and it is not clear if all the above combine to give any error-correction! To make sense of the above, lets us try using an inappropriate choice of parameters. Let us suppose $p_1 = p_2 = \cdots = p_n = p$. (Ahem! We this is somewhat inconsistent with the hypothesis that p_i's are relatively prime. But this is the best way to get a sense of the idea.) Then $N = p^n$, $T = p^t$, $K = p^k$. Fix some ℓ. We can apply Lemma 14.5 if we let $B' = N^{1/\ell} = p^{n/\ell}$, and then Lemma 14.3 can be applied if $B \approx 2^\ell p^{n/\ell}$. Lemma 14.6 can then be applied if $(\ell+1)BK^\ell < T$, which is well approximated by $p^{n/\ell + k\ell} < p^t$. So with all these outrageous approximations and assumptions, we see we are in a situation similar to the Reed-Solomon decoding case two lectures back: We can decode provided $n/\ell + k\ell < t$. The expression on the LHS is minimized by setting $\ell = \sqrt{n/k}$ and this gives decoding from $2\sqrt{nk}$ agreements. The following theorem is what can be obtained from a rigorous analysis of the above after some minor optimizations in the bounds on the coefficients of the polynomials, and is proven by Goldreich, Ron, and Sudan [39]. The main difference is a $\sqrt{\log p_n / \log p_1}$ factor in the amount of agreements required. This comes up because the p_i's are, unfortunately, not all equal.

Theorem 14.7 ([39]) *There exists a list-decoder for CRT codes that on input an instance with relatively prime integers $p_1, \ldots, p_n$, and information parameter k and agreement parameter t, recovers all messages with agreement t provided $t > \sqrt{2kn \log p_n / \log p_1}$ in polynomial time.*

Using the ideas of multiplicities one can (almost) get rid of the factor $\sqrt{2}$ in the above agreement; and this was done by Boneh [22]. Boneh uses the same multiplicities on each coordinate. Using multiplicities that vary as a function of the p_i's, gives further improvements and one can now decode from $t > \sqrt{(1+\epsilon)kn}$ error in time polynomial in $1/\epsilon$, n and $\log p_n$. This final result was given by Guruswami, Sahai, and Sudan [44].

Bibliographic notes

We never mentioned an unambiguous decoding algorithm for CRT codes, but one certainly exists. The first such algorithm was given by Mandelbaum [76]. Surprisingly this algorithm does not decode up to half the minimum distance, at least not in polynomial time. To get a decoder decoding up to half the minimum distance, one needs to combine this algorithm with a GMD decoder.[2] This was done by [44].

[2]This could be a good, healthy, workout. The Mandelbaum algorithm is what you would get if you set $\ell = 1$ above and picked optimal bounds on the coefficients of the constant and degree one term in the Q-polynomial. Compute the

The first list-decoding algorithm for CRT codes was due to Goldreich et al.[40]. Boneh [22] and Guruswami et al. [44] improved the error-correction bounds subsequently. Guruswami et al. [44] also developed the machinery of ideal codes.

number of errors this algorithm can recover from. Then show how to do a GMD like algorithm + analysis to recover from $(n-k)/2$ errors.

Chapter 15

6.897 Algorithmic Introduction to Coding Theory	September 5, 2001
Lecture 15	
Lecturer: Madhu Sudan	*Scribe: Constantine Caramanis*

15.1 Introduction

This lecture covers the decoding of Reed-Muller Codes. Over the years, these codes have been repeated targets of decoding algorithms, with many successful approaches. In fact, the one of the two papers that led to the current name of these codes really only gives an algorithm for decoding (and doesn't construct codes, per se). Much of this work is carried out in the coding theory literature. It culminates in a decoding algorithm that can decode Reed-Muller codes up to half the minimum distance, for *every* choice of parameters. An elegant description of this algorithm is given in The Handbook of Coding Theory [88] in the chapter on Algebraic Geometry Codes by Høholdt, van Lint, and Pellikaan [50]. We won't cover their algorithm today, but if Madhu gets the energy, he might add a writeup on their algorithm to these lecture notes sometime soon.

Instead today's lecture will focus on list-decoding algorithms for Reed-Muller codes. Unlike the algorithms mentioned above, these do not work for *every* choice of parameters. However if the alphabet size q is sufficiently large, compared to the degree of the polynomials in the message space, these algorithms work very well and decode much more than half the errors.

In addition to the ability to correct many errors, these algorithms are significant in that they played a significant role in many developments in complexity theory over the past two decades. E.g., Lipton [70] showed that the "permanent of a matrix was hard to compute on random matrices". This result was a consequence of a simple unambiguous decoding algorithm for Reed-Muller codes implicit in Beaver and Feigenbaum [13]. The decoding algorithm for Reed-Muller codes also played a role in results showing "IP = PSPACE [72, 99]", "MIP=PSPACE [10]" and the "PCP Theorem [7, 6]". There is also a sequence of results showing progressively weaker complexity-theoretic conditions that would suffice to show "BPP = P" [85, 11, 51]. These results can also be simplified and optimized by using list-decoders for Reed-Muller codes [112].

The algorithm in today's lecture will be derived as a simple algorithm that exploits some randomness properties of "lines" in $\mathbb{F}_q^m$, rather than any serious algebraic properties. The only algebraic elements will be decoding algorithms for Reed-Solomon codes and we'll just adopt them as a black box from previous lecture. We'll describe the algorithm in two steps - first we give an algorithm that decodes from a small number of errors (much less than half the minimum distance). Then we jump to a list-decoder recovering from large number of errors (under the above-mentioned caveat of $q \gg \ell$).

Let us first recall the definition of Reed-Muller codes (first covered in lecture 4 on September 19th, 2001). Recall that these are the generalization of Reed-Solomon codes to multivariate polynomials.

Definition 15.1 (Reed-Muller Codes) *A Reed-Muller code, $\mathrm{RM}_{m,d,q}$, is the code whose codewords are evaluations of m-variate polynomials of total degree at most d, over all elements in $\mathbb{F}_q$.*

The $\mathrm{RM}_{m,d,q}$ code is a linear code with $n = q^m$, $k = \binom{m+d}{d}$, and with relative distance $\left(1 - \frac{d}{q}\right)$, when $d \leq q$. The goal of today's lecture to give list-decoding algorithms that work provided $d \ll q$.

For today's lecture it is convenient to think of this task as a "function reconstruction task" rather than that of the task of reconstructing a vector or string representing the codeword. We will thus think of the received word being given by some function $f : \mathbb{F}_q^m \to \mathbb{F}_q$, and our goal is to output a list of all nearby codewords, where the codewords are also given by functions $p : \mathbb{F}_q^m \to \mathbb{F}_q$. We will also extend the relative Hamming distance to apply to functions. Thus we have the distance between functions f and g to be $\delta(f,g) = \Pr_{\mathbf{x} \in \mathbb{F}_q^m}[f(x) \neq g(x)]$. Note that this is just the standard Hamming distance normalized so as to be between 0 and 1, if f and g are interpreted as strings rather than functions.

Reed-Muller Decoding:
GIVEN: Oracle access to a function $f : \mathbb{F}_q^m \to \mathbb{F}_q$, a degree parameter d and a disagreement parameter δ.
TASK: A (list of all) degree d polynomials $p : \mathbb{F}_q^m \to \mathbb{F}_q$ such that $\delta(f,p) \leq \delta$.

Note that we haven't specified representation of the output polynomial p precisely. The natural representation may be the coefficients of p. However, given that interpolation is a simple task, it would be equally useful to have p represented as a table of values. The latter representation certainly allows one to compute the former representation in time $\mathrm{poly}(q^m)$. However our algorithms will be more natural in the latter representation; and also significantly more efficient. In fact we will produce randomized algorithms that compute $p(\mathbf{a})$ for any vector $\mathbf{a} \in \mathbb{F}_q^m$ in time $\mathrm{poly}(m,d,q)$. When $d \approx m$ this is vastly more efficient that $\mathrm{poly}(q^m)$. The model needs careful fleshing out; but we won't do so right away. Instead we will let it evolve and the spend some time reviewing the final model at the end of these notes.

15.2 Decoding from very low error

We start with describing an extremely simple randomized algorithm that recovers from very little error δ. The amount of error will certainly be small enough to put us in the case of the unambiguous decoding problem. So the polynomial p that is δ-close to f is unique. Our algorithm will try to guess the value $p(\mathbf{a})$, by looking at f on a small random sample of values. The trick is in picking the right "random sample". We will look at f on a "line" in $\mathbb{F}_q^m$. We define the notion of a line next.

15.2.1 Lines in $\mathbb{F}_q^m$

Definition 15.2 *The line $\mathbb{F}_q^m$ through a point* $\mathbf{a}$ *with slope* $\mathbf{b}$ *is the set of points:*

$$\ell_{\mathbf{a},\mathbf{b}} := \{\mathbf{a} + t\mathbf{b} \mid t \in \mathbb{F}_q\}.$$

Note that if $\mathbf{b} = \mathbf{0}$, then the line $\ell_{\mathbf{a},\mathbf{b}}$ consists of a single point, else it consists of exactly q distinct points of $\mathbb{F}_q^m$. While it might be tempting to simply rule out all lines with $\mathbf{b} = \mathbf{0}$ as uninteresting, we won't do so, for reasons to be clarified later. However we will refer to them as *degenerate* lines, just to put them in their place.

In the definition above, we thought of the line as an unordered set of points. However it is also useful to think of them as a "parameterized" set of points. More precisely, a line is a function $\ell_{\mathbf{a},\mathbf{b}} : \mathbb{F}_q \to \mathbb{F}_q^m$, given by $\ell_{\mathbf{a},\mathbf{b}}(t) = \mathbf{a} + t\mathbf{b}$.

The two representations have their advantages. Switching between them is even further beneficial, as we will see towards the end of this lecture. While switching, one should keep in mind that the parameterization specifies the set uniquely; however the set does not specify the parameterization uniquely. In particular the lines $\ell_{\mathbf{a},\mathbf{b}}$ and $\ell_{\mathbf{a}+t_1\mathbf{b},t_2\mathbf{b}}$ are the same for $t_1, t_2 \in \mathbb{F}_q$ if $t_2 \neq 0$. This will be useful to us later.

We start by describing the nice properties of lines. The first nice property of a line is its randomness property. We first define the notion of a random line and a random line through a point $\mathbf{a} \in \mathbb{F}_q^m$.

Definition 15.3 *For a point $\mathbf{a} \in \mathbb{F}_q^m$, a random line through* $\mathbf{a}$ *is the random variable $\ell_{\mathbf{a},\mathbf{b}}$ where* $\mathbf{b}$ *is picked uniformly from $\mathbb{F}_q^m$. A random line in $\mathbb{F}_q^m$ is the random variable $\ell_{\mathbf{a},\mathbf{b}}$ where both* $\mathbf{a}$ *and* $\mathbf{b}$ *are chosen independently and uniformly at random from $\mathbb{F}_q^m$.*

The following proposition explains the niceties of random lines.

Proposition 15.4 *A random line through $\mathbb{F}_q^m$ is a collection of pairwise independent points in $\mathbb{F}_q^m$. I.e., for $t_1 \neq t_2 \in \mathbb{F}_q$, the points $\ell_{\mathbf{a},\mathbf{b}}(t_1)$ and $\ell_{\mathbf{a},\mathbf{b}}(t_2)$ are distributed independently and uniformly from $\mathbb{F}_q^m$. Furthermore, for every* $\mathbf{a}$ *every non-zero point on a random line through $\mathbb{F}_q^m$ is a random point. I.e., for every $t \in \mathbb{F}_q - \{0\}$, the point $\ell_{\mathbf{a},\mathbf{b}}(t)$ is distributed uniformly in $\mathbb{F}_q^m$, when* $\mathbf{b}$ *is distributed uniformly in $\mathbb{F}_q^m$.*

Remark: Note that the proposition above relies on the fact that we allow lines to be degenerate, else it would not be true.

Proof Proof to be added. ■

Next we point out the nice algebraic properties of the line. This notion alludes to the restrictions of functions to lines. Note that when the line is viewed as a function $\ell : \mathbb{F}_q \to \mathbb{F}_q^m$, then it composes naturally with a function $f : \mathbb{F}_q^m \to \mathbb{F}_q$ to give a "univariate function" $f|_\ell : \mathbb{F}_q \to \mathbb{F}_q$, given by $f|_\ell(t) = f(\ell(t))$. The following proposition asserts that restrictions of algebraically nice functions give algebraically nice functions.

Proposition 15.5 *For every degree d polynomial $p : \mathbb{F}_q^m \to \mathbb{F}_q$ and every line $\ell : \mathbb{F}_q \to \mathbb{F}_q^m$, the function $p|_\ell$ is a univariate polynomial of degree at most d.*

Given a line $\ell : \mathbb{F}_q \to \mathbb{F}_q^m$ we say that a point $\mathbf{z} \in \mathbb{F}_q^m$ *lies* on the line ℓ if there exists $t \in \mathbb{F}_q$ such that $\mathbf{z} = \ell(t)$. If the line is not degenerate, then such a t is unique if it exists. If the line is degenerate, then every t works, and we will default to $t = 0$ in case of ambiguities. Specifically, if $\mathbf{z}$ lies on ℓ, then we let $\ell^{-1}(\mathbf{z}) = t$ such that $\mathbf{z} = \ell(t)$, if ℓ is not degenerate, and $\ell^{-1}(\mathbf{z}) = 0$ otherwise.

Univariate functions defined on lines arise naturally in this lecture. Given a univariate polynomial h and a line $\ell : \mathbb{F}_q \to \mathbb{F}_q^m$, we will let $h_{\ell^{-1}}$ denote the partial function from ℓ (viewed as a subset of $\mathbb{F}_q^m$) to $\mathbb{F}_q$ given as follows: $h_{\ell^{-1}}\{\mathbf{z}\} = h(\ell^{-1}(\mathbf{z}))$. If the line ℓ is clear from context we will omit it, and use the notation $h\{\mathbf{z}\}$ to denote $h_{\ell^{-1}}\{\mathbf{z}\}$.

15.2.2 The algorithm

Our first algorithm for decoding Reed-Muller codes is an elementary consequence of the two properties of random lines in $\mathbb{F}_q^m$ mentioned above. (The latter property is actually a property of all lines and hence also a property of random lines.)

As noted earlier it suffices to construct an algorithm that computes $p(\mathbf{a})$, given oracle access to a function $f : \mathbb{F}_q^m \to \mathbb{F}_q$ that is very close to a degree d polynomial $p : \mathbb{F}_q^m \to \mathbb{F}_q$.

In fact it even suffices to give a randomized algorithm that computes this value with probability greater than, say, 2/3, over its internal coin tosses. Repetition followed by a plurality vote suffices to reduce this error probability. Once the error probability reduces to say less than $\frac{1}{3}q^{-m}$, then repeating this trial for every choice of $\mathbf{a}$ produces the correct codeword with probability at least $\frac{2}{3}$.

The basic idea to compute $p(\mathbf{a})$ is to focus on just a random line ℓ through $\mathbf{a}$ and to reconstruct the function $p|_\ell$. By Proposition 15.5 this is a univariate polynomial of degree d; and based on Proposition 15.4 we know that $p|_\ell$ and $f|_\ell$ have good agreement. So $p|_\ell$ can, hopefully, be recovered efficiently and once this is done, all we need to do is output $p_\ell\{\mathbf{a}\}$. Below we describe and analyze the most elementary form of this algorithm. In this form the algorithm only needs $q \geq d+2$ to work. However the amount of error it will correct is quite small. We will worry about that later though.

Simple RM decoder:

Given: Oracle access to $f : \mathbb{F}_q^m \to \mathbb{F}_q$. Point $\mathbf{a} \in \mathbb{F}_q^m$ and parameter d.

Promise: There exists a degree d polynomial $p : \mathbb{F}_q^m \to \mathbb{F}_q$ such that $\delta = \delta(p, f) \leq \frac{1}{3(d+1)}$.

Goal: Output $p(\mathbf{a})$.

Step 1: Pick $\mathbf{b} \leftarrow \mathbb{F}_q^m$ at random and let $\ell = \ell_{\mathbf{a},\mathbf{b}}$.

Step 2: Let $\alpha_1, \ldots, \alpha_{d+1}$ be distinct elements of $\mathbb{F}_q - \{0\}$. For $i \in [d+1]$, let $\beta_i = f(\mathbf{a} + \alpha_i \mathbf{b})$.

Step 3; Interpolate to find a degree d univariate polynomial h such that $h(\alpha_i) = \beta_i$ for every $i \in [d+1]$.

Step 4: Output $h(0)$.

Note that Step 2 above requires $q > d+1$. We will show below that this requirement suffices for correctness, provided the error is small enough.

Lemma 15.6 *Under the assumption $\delta = \delta(f, p) \leq \frac{1}{3(d+1)}$ the algorithm* **Simple RM decoder** *outputs $p(\mathbf{a})$ with probability at least $\frac{2}{3}$.*

Proof We define $d+1$ "bad" events B_i over the random choice of $\mathbf{b}$. We show that if none of these events occurs, then the algorithm correctly outputs $p(\mathbf{a})$. Then we show that the probability that none of these events occurs is at least $1-(d+1)\delta$. The lemma follows once the above is shown.

We define the event B_i to be the case "$\beta_i \neq p|_\ell(\alpha_i)$". Note that if none of these events occur, then we know the value of the function $p|_\ell(\cdot)$ at $d+1$ distinct values in $\mathbb{F}_q$. Further, by Proposition 15.5, $p|_\ell$ is a polynomial of degree at most d. Thus, the polynomial h found in Step 3 *is* the function $p|_\ell$. Thus the value output in Step 4 is $p|_\ell(0) = p(\ell(0)) = p(\mathbf{a})$. It thus suffices to bound the probability of the bad events.

Note that B_i happens if and only if $f(\ell(\alpha_i)) \neq p(\ell(\alpha_i))$. By Proposition 15.4, we have that $\ell(\alpha_i)$ is a random point of $\mathbb{F}_q^m$ and thus the probability that f does not agree with p at this point is exactly $\delta(f,p)$. Thus we have that the probability of B_i is δ. By the union bound, the probability that at least one of the bad events occurs is at most $(d+1)\delta$. The probability that none of them occurs is at most $1-(d+1)\delta$. This concludes the proof. ■

Note that the algorithm is quite efficient — it runs in time $\mathrm{poly}(m,d)$, while the codeword has length $\binom{m+d}{d}$ which could be exponentially larger. Of course, it does not recover all the codeword at once — this is simply impossible in this much time; but it can recover any coordinate of the codeword in such time. When we formally describe the model in which the algorithm works, we will focus on these aspects and then formally describe the result obtained so far. For now, we will satisfy ourselves with just an informal understanding of the result.

15.3 Improving the error-correction capability

We now describe ideas that can be applied to improving the error-correction capability of the simple algorithm described in the previous section. The error-correction capability comes at a price: The requirement on the field size now goes up, and the algorithms get slightly more complicated.

To motivate the basic idea behind the improvement, let us look back to the reason why the error correction capability was so low ($\Theta(1/d)$) in the algorithm of the last section. The reason we lost so much was that we required that $d+1$ queries on a random line through $\mathbf{a}$ should *all* be error-free. To improve the performance, we will make a few more queries, but then allow for the possibility that a few answers are incorrect. The polynomial $p|_\ell$ will then be the polynomial that "usually" agrees with the queried values — this polynomial can be found by a Reed-Solomon decoding step. The number of queries that the algorithm makes can be varied depending on the rate of error we wish to correct, the run time we desire, and the field size. We will set this number, somewhat arbitrarily, to $5(d+1)$ and demonstrate the effect. We thus get the algorithm below:

Improved RM decoder:

Given: Oracle access to $f : \mathbb{F}_q^m \to \mathbb{F}_q$. Point $\mathbf{a} \in \mathbb{F}_q^m$ and parameter d.

Promise: There exists a degree d polynomial $p : \mathbb{F}_q^m \to \mathbb{F}_q$ such that $\delta = \delta(p,f) \leq \frac{2}{15}$.

Goal: Output $p(\mathbf{a})$.

Step 1: Pick $\mathbf{b} \leftarrow \mathbb{F}_q^m$ at random and let $\ell = \ell_{\mathbf{a},\mathbf{b}}$.

Step 2: Let $\alpha_1, \ldots, \alpha_{5(d+1)}$ be distinct elements of $\mathbb{F}_q - \{0\}$. For $i \in [5(d+1)]$, let $\beta_i = f(\mathbf{a} + \alpha_i \mathbf{b})$.

Step 3; Find a degree d univariate polynomial h such that $h(\alpha_i) = \beta_i$ for at least $3(d+1)$ choices of $i \in [5(d+1)]$.

Step 4: Output $h(0)$.

All steps above are efficient. In particular, Step 3 can be executed in poly(d) time by using any unambiguous decoding algorithm for the problem such as the Welch-Berlekamp algorithm described in a previous lecture. We thus get the following proposition.

Proposition 15.7 **Improved RM decoder** *runs in* poly(d, m) *time.*

Lemma 15.8 *Under the assumption* $\delta = \delta(f, p) \leq \frac{2}{15}$ *the algorithm* **Simple RM decoder** *outputs* $p(\mathbf{a})$ *with probability at least* $\frac{2}{3}$.

Proof As in the proof of Lemma 15.6 we define, for every $i \in [5(d+1)]$, B_i to be the event "$\beta_i \neq p|_\ell(\alpha_i)$". Note that for every i the probability of B_i is exactly δ. Now let B be the event that B_i is true for more than $2(d+1)$ choices of i. The probability that B occurs can be upper bounded, using Markov's inequality, by $\delta * (5(d+1))/(2(d+1)) \leq 1/3$. Thus with probability at least 2/3rd B does not occur. Now we note that if B does not occur, then $p|_\ell$ agrees with the points $\{(\alpha_i, \beta_i) \mid i \in [(5(d+1)]\}$ on $3(d+1)$ points and hence is the unique solution in Step 3. Thus in Step 4 we output $p|_\ell(0) = p(\mathbf{a})$ with probability at least 2/3. ■

The ideas in the algorithm above can be pushed further to get an algorithm correcting almost $\frac{1}{2}$ of error though this requires some work (and further restrictions on the ratio of q to d). However, we won't describe the details. Instead we move on to the task of list-decoding.

15.4 A List Decoding Algorithm

Part of the reason why the algorithms from the previous section could not correct too many errors is that they never exploited the ability to list-decode the Reed-Solomon codes (in Step 4). If we did, we would be able to find polynomials with much smaller agreement with f on any given line ℓ. However this doesn't suffice to solve the list-decoding problem for Reed-Muller codes. What should we do with a list of univariate polynomials that agree with f on ℓ? How do we find out which one is the polynomial "p"? In fact, come to think of it, what is p? In previous sections p was uniquely specified as the nearest polynomial (codeword) to the function f (the received vector). Now that we are hoping to perform list-decoding, there is a list of polynomials that could have the desired agreement. It seems that we can no longer use the "reconstruction problem" (compute the value of the nearest polynomial at some fixed point $\mathbf{a} \in \mathbb{F}_q^m$) to solve the decoding problem.

Turns out one can salvage this path after all. The way we'll do this is by focussing on one of the specific polynomials p that is close to f, and by giving a small amount of additional information, called *advice*, that suffices to specify it uniquely. We will then give a reconstruction procedure to computes $p(\mathbf{a})$ given $\mathbf{a}$. By varying the advice, we'll then be able to compute all the polynomials close to f.

So what is the advice that we'll use to specify p? It turns out the value of p at one, randomly chosen point $\mathbf{b} \in \mathbb{F}_q^m$, specifies it uniquely, provided q is large enough relative to δ and d. As usual, when it comes to list-decoding it is more informative to focus on the amount of agreement rather than the amount of disagreement between f and the polynomial p. Define $\tau(f,g) = \Pr_{\mathbf{x} \leftarrow \mathbb{F}_q^m}[f(x) = g(x)]$ to be the *agreement* between f and g. We start with the following simple proposition that proves that the value of a polynomial at a random point specifies it uniquely, given a nearby function f.

Proposition 15.9 *Let $p_1, \ldots, p_n$ be a list of all m-variate degree d polynomials over $\mathbb{F}_q$ satisfying $\tau(f,g) \geq \tau_0$. If $\tau \geq \sqrt{2d/q}$, then $n \leq 2/\tau$ and with probability at least $1 - \binom{n}{2}(\frac{d}{q}) \geq 1 - \frac{2d}{\tau^2 q}$ over the choice of $\mathbf{z} \in \mathbb{F}_q^m$ it is the case that the sequence of elements $\langle p_1(\mathbf{z}), \ldots, p_n(\mathbf{z})\rangle$ are all distinct.*

Proof The first part of the proposition, claiming $n \leq 2/\tau$ is just recalling the Johnson bound from a previous lecture. The second part follows from an application of the union bound to the $\binom{n}{2}$ possible "bad" events B_{ij}, $1 \leq i < j \leq n$, where B_{ij} is the event "$p_i(\mathbf{z}) = p_j(\mathbf{z})$". Note that B_{ij} occurs with probability at most d/q (by the "Schwartz Lemma"). ■

So this motivates the plan of the next algorithm. We will assume that we are given one useful piece of information — namely the value of p at one (randomly chosen) point $\mathbf{z} \in \mathbb{F}_q^m$. Say $p(\mathbf{z}) = \gamma$. Now suppose we wish to reconstruct the value of p at $\mathbf{a} \in \mathbb{F}_q^m$. We will execute the algorithm of the previous sections and pick a line ℓ through $\mathbf{a}$. Suppose we are fortunate enough that $\mathbf{z}$ lies on ℓ. In this case, given a list of polynomials $h_1, \ldots, h_n$ that have non-trivial agreement with f on ℓ, we can determine which one is $p|_\ell$ by considering the values $h_i\{\mathbf{z}\}$. Hopefully they are all different and then the polynomial h_i for which $h_i\{\mathbf{z}\} = \gamma$ is $p|_\ell$. $h_i\{\mathbf{a}\}$ is then the value we are seeking. The only catch is that a random line will no longer work - it is very unlikely to pass through $\mathbf{z}$. We will fix this by deterministically picking the line that passes through $\mathbf{z}$! This plan is implemented below, with some of the steps being clarified further. The algorithm is described for a fixed choice of $\mathbf{z}$ and advice γ. For reasons to be clarified later, we will simply call this a subroutine.

List-decoding subroutine $A_{\mathbf{z},\gamma}$

Given: Oracle access to f and point $\mathbf{a} \in \mathbb{F}_q^m$.

Step 1: Let $\mathbf{b} = \mathbf{z} - \mathbf{a}$ and let $\ell = \ell_{\mathbf{a},\mathbf{b}}$.

Step 2: For $\alpha \in \mathbb{F}_q$, let $\beta_\alpha = f(\mathbf{a} + \alpha\mathbf{b})$.

Step 3; Find all degree d univariate polynomials $h_1, \ldots, h_n$ such that $h_i(\alpha) = \beta_\alpha$ for at least $\frac{\tau}{2}q$ choices of $\alpha \in \mathbb{F}_q$.

Step 4: If there exists a unique index $i \in [n]$ such that $h_i\{\mathbf{z}\} = \gamma$, output $h_i(0)$, else output error.

We start by noticing that all steps above run in polynomial time provided the parameters are favorable. In particular, Step 3 can be executed in polynomial time assuming $\frac{\tau}{2} > \sqrt{d/q}$ using the improved List-decoding algorithm for Reed-Solomon codes from a previous lecture.

Proposition 15.10 *If $\tau > \sqrt{4d/q}$ then the subroutine $A_{\mathbf{z},\gamma}$ runs in time* $\mathrm{poly}(q,m)$.

Exercise: Suggest variations of the algorithm above so it runs in time $\mathrm{poly}(d, \frac{1}{\tau}, \log q)$.

Next we analyze the correctness of the algorithm. We show first that for a random pair $(\mathbf{z}, \mathbf{a})$, the algorithm $A_{\mathbf{z},p(\mathbf{z})}$ is very likely to output $p(\mathbf{a})$. We conclude that there exists a vector $\mathbf{z}$ (in fact most choices would work) such that $A_{\mathbf{z},p(\mathbf{z})}$ computes a function very close to p. This is not what we want — we want an algorithm that always computes p. However the algorithms of the previous section can now be applied to $A_{\mathbf{z},p(\mathbf{z})}$ to get a randomized algorithm that computes p correctly everywhere with high probability. Thus the following lemma will be quite sufficient for our purposes.

Lemma 15.11 *For any $\epsilon > 0$, suppose $q \geq \frac{16(d+1)}{\tau^2\epsilon}$ and p is a polynomial with agreement τ with f. Then, for a random pair $\mathbf{z}, \mathbf{a} \in \mathbb{F}_q^m$, $A_{\mathbf{z},p(\mathbf{z})}$ outputs $p(\mathbf{a})$ with probability at least $1-\epsilon$.*

Proof As in previous proofs, we describe some bad events and then claim that if none of the bad events occur, then the algorithm $A_{z,p(\mathbf{z})}$ outputs $p(\mathbf{a})$. Our first bad event B corresponds the bad events of previous proofs, i.e., to poor agreement between p and f on ℓ. Specifically B is the event that "p and f have less than $\tau/2$ agreement on ℓ". We now describe the second bad event: Let $h_1, \ldots, h_n$ be all univariate polynomials that have $\tau/2$ agreement with $f|_\ell$. Let C be the event that there exists a pair $1 \leq i < j \leq n$ such that $h_i\{\mathbf{z}\} = h_j\{\mathbf{z}\}$. We show below that if neither B nor C occurs, then the algorithm $A_{\mathbf{z},p(\mathbf{z})}$ outputs $p(\mathbf{a})$. Later we give upper bounds on the probabilities of B and C.

Claim 15.12 *If neither of the events B or C occurs, then $A_{\mathbf{z},p(\mathbf{z})}$ outputs $p(\mathbf{a})$ on input $\mathbf{a}$.*

Proof This is relatively straightforward. Since the event B does not occur, we have that the polynomial $p|_\ell$ has at least $\tau/2$ agreement with f on the line ℓ. Thus one of the polynomials $h_1, \ldots, h_n$ computed by $A_{\mathbf{z},p(\mathbf{z})}$ in Step 3 is $p|_\ell$. Say it is the polynomial h_i. Then $h_i\{\mathbf{z}\} = p(\mathbf{z})$. But since the event C did not occur, we know that $h_j\{\mathbf{z}\} \neq h_i\{\mathbf{z}\}$ for any other index j. Thus h_i is the unique polynomial satisfying the condition of Step 4 and thus Step 4 results in the output $h_i(0) = p(\mathbf{a})$. ■

Claim 15.13 *The probability of event B, taken over the choices of $\mathbf{z}$ and $\mathbf{a}$, is at most $\frac{4}{\tau q}$.*

Proof This is a simple application of the Chernoff bound (see first lecture). By Proposition 15.4, the points of the line ℓ are distributed uniformly over $\mathbb{F}_q^m$ and pairwise independent. On any one point, the probability that f agrees with p is τ. The expected number of agreements between f and p on q pairwise independent points is thus τq. The probability that this number deviates from its expectation by half the expectation (a requirement for the event B) is bounded by $\frac{4}{\tau q}$. ■

Claim 15.14 *The probability of event C, taken over the choice of $\mathbf{z}$ and $\mathbf{a}$, is at most $\frac{8d}{\tau^2 q}$, provided $\tau > 2\sqrt{d/q}$.*

Proof The claim would be obvious, following immediately from Proposition 15.10, if $\mathbf{z}$ was chosen to be a random point on line ℓ after the line is fixed. But this is not the case! Or is it?

In the way the algorithm is described, $\mathbf{a}$ and $\mathbf{z}$ are chosen first and then ℓ is defined based on them. However we could pick ℓ at random first, as a *set* and then $\mathbf{a}$ and $\mathbf{z}$ to be two random (distinct, assuming ℓ turns out to be non-degenerate) points on it later. Note that the polynomials $h_1, \ldots, h_n$ are already well-defined once the line is chosen as a *set*, without fixing the parameterization. (They are not fixed in terms of their coefficients; but they are fixed as functions from ℓ to $\mathbb{F}_q$.) Thus the probability, when we pick $\mathbf{z}$ at random on ℓ that $h_i\{\mathbf{z}\} = h_j\{\mathbf{z}\}$ for some distinct pair i, j is at most $\frac{8d}{\tau^2 q}$ (applying Proposition 15.10 in the univariate case with agreement set to $\tau/2$). The claim follows. ■

We are now ready to glue together the proof of the lemma. We will pick q large enough so that the two events above happen with probability at most $\epsilon/2$ each. Thus we get the condition $q \geq \max\{\frac{8}{\tau\epsilon}, \frac{16d}{\tau^2\epsilon}\}$. To make this simpler we set $q \geq \frac{16(d+1)}{\tau^2\epsilon}$. Once we have this condition we find that the probability that B or C occurs is at most ϵ and with the remaining probability $A_{\mathbf{z},p(\mathbf{z})}$ outputs $p(\mathbf{a})$. ■

Now we describe the actual list-decoding algorithm. The algorithm simply picks a collection of random points $\mathbf{z}_1, \ldots, \mathbf{z}_t$'s and enumerates all possible choices of $\gamma \in \mathbb{F}_q$ for $p(\mathbf{z}_i)$. It then applies the algorithm **Improved RM decoder** to the functions $A_{\mathbf{z}_i,\gamma}$. For every degree d polynomial p that has agreement τ with f, one of the algorithms above computes p. Again we summarize this formally below:

RM List-decoder

Given: Oracle access to $f : \mathbb{F}_q^m \to \mathbb{F}_q$. Point $\mathbf{a} \in \mathbb{F}_q^m$, agreement parameter τ and degree parameter d.

Goal: To produce a list of randomized algorithms computing functions $g_1, \ldots, g_n : \mathbb{F}_q^m \to \mathbb{F}_q$ that includes every degree d polynomial $p : \mathbb{F}_q^m \to \mathbb{F}_q$ with τ agreement with f.

Step 1: Compute parameters $\epsilon = \frac{1}{15}$, $t = \log \frac{4}{\tau}$ and $n = tq$.

Step 2: Pick $\mathbf{z}_1, \ldots, \mathbf{z}_t$ independently and uniformly at random from $\mathbb{F}_q^m$.

Step 3: For every $i \in [t]$ and $\gamma \in \mathbb{F}_q$ output the algorithm **Improved RM Decoder** accessing the oracle $A_{\mathbf{z}_i,\gamma}$.

The following lemma analyzes the correctness of the algorithm.

Lemma 15.15 *With probability at least $\frac{1}{2}$, the algorithms constructed by* **RM List-Decoder** *include one for every polynomial p that has agreement τ with f, provided $q \geq \frac{240(d+1)}{\tau^2}$.*

Proof Let $p_1, \ldots, p_m$ be all polynomials of degree at most d that have agreement at least τ with f. Note, by the usual Johnson bounding argument, that $m \leq \frac{2}{\tau}$. Fix $i \in [m]$ and let $p = p_i$. We prove that the event that none of the algorithms constructed by **RM List-decoder** turns out to be a randomized algorithm computing p occurs with probability at most $2^{-t} \leq \frac{\tau}{4}$ (this inequality is the one that leads to the setting of t in the algorithm). By the union bound it follows that all the required polynomials are in the output list with probability at least $\frac{1}{2}$.

Fix $j \in [t]$. Note that the choice of q ensures that Lemma 15.11 can be applied for $\epsilon = \frac{1}{15}$. By Lemma 15.11, $A_{\mathbf{z}_j,p(\mathbf{z}_j}$ computes $p(\mathbf{a}')$ with probability at least $1-\epsilon$ for random input $\mathbf{a}'$. Thus, with probability at least $\frac{1}{2}$ it must be the case that $\mathbf{z}_j$ is such that $A_{\mathbf{z}_j,p(\mathbf{z}_j}$ has agreement at least $1-2\epsilon$ with p. Since $2\epsilon \leq 2/15$, we have that **Improved RM Decoder** computes $p(\mathbf{a})$ correctly

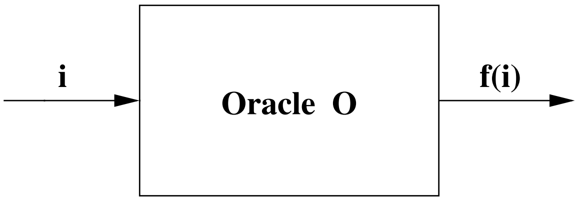


Figure 15.1: Implicit representation of input

on every input $\mathbf{a}$ with probability at least 2/3. Thus with probability at most $\frac{1}{2}$, the algorithm **Improved RM Decoder** with oracle access to $A_{\mathbf{z}_j, p(\mathbf{z}_j)}$ does *not* compute p. The probability that this happens for every $j \in [t]$ is thus at most 2^{-t}. ■

15.5 Formal Model and Theorems

To encapsulate the results of this lecture in the strongest possible way, we ought to formalize the model that is being developed to represent words and decoding algorithms. The main feature we wish to focus on is the *extreme efficiency* of the algorithms in some *implicit* model of the inputs, outputs, decoding and list-decoding. We describe these in order.

In implicit decoding problems, both inputs and outputs will be represented implicitly. In particular, the received vector will be given by an oracle. A vector $\mathbf{v} = \langle v_1, \ldots, v_n \rangle \in \mathbb{F}_q^n$ is specified *implicitly* by an oracle O for a function $f : [n] \to \mathbb{F}_q$. Given an index $i \in [n]$ the oracle responds with the value $f(i) = v_i$. Since the vector $\mathbf{v}$, the oracle O, and the function f represent the same information (though they carry different meanings as computational entities) we will save ourselves some symbols and use the function to represent all three objects. Figure 15.1 gives a pictorial view of implicit inputs.

The implicit representation of outputs requires some care. A first attempt may be to suggest exactly the same definition as used for implicit inputs, i.e., by oracles. But that does not appear to be constructive! What does it mean to say we will simply output a box — we need to be more explicit. How does the box compute its functions. So really we should be representing outputs by algorithms to compute them. But then if the output contains k bits of information, then it is impossible to produce a small algorithm (of length smaller than k) that can give out these k bits of information. Yet, we have seen above that it is possible to decode very efficiently, much more efficiently than the time it would take to write down the output. How did we acheive that? Close examination reveals that our outputs were actually being specified relative to the input. Specifically, we produced an algorithm to compute the output vector that made oracle calls to the input oracle. (See Figure 15.2.) Strictly speaking all we describe is the algorithm A (pictured in Figure 15.3) that, when given oracle access to the input f, computes the output word. These kind of objects are referred to in the computer science literature as probabilistic oracle machines and play a significant role in some of the major developments there.

We now formally describe the implicit decoding problem:

Definition 15.16 (Implicit unambiguous decoding) *For a fixed family of codes $\mathcal{C}$, the implicit decoding problem is:*

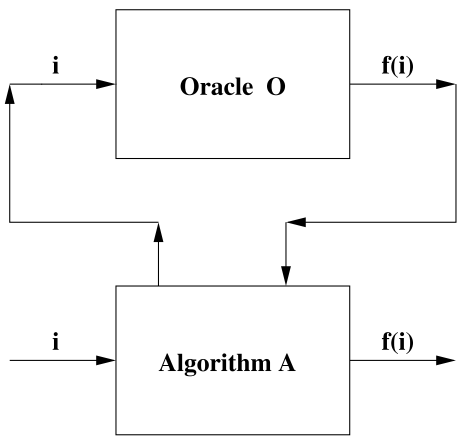


Figure 15.2: Implicit representation of output

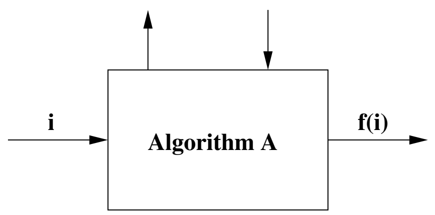


Figure 15.3: The actual implicit output

GIVEN: *Parameters n, q (and any other parameters that are relevant) of the code $C \in \mathcal{C}$, error parameter $e < \frac{\Delta(C)}{2}$, and implicit representation of received vector $\mathbf{f} \in \mathbb{F}_q^n$.*
GOAL: *Output an implicit representation A of codeword $\mathbf{c} \in C$ such that $\Delta(\mathbf{c}, \mathbf{f}) \leq e$, if such a codeword exists. Specifically, A takes as input $i \in [n]$ and outputs c_i, the ith coordinate of $\mathbf{c}$.*

Remarks:

1. The running time of the decoder is defined to be the sum of the the time taken to compute a description of A and the worst-case running time of A.
2. A is allowed to be randomized, in which case it must produce the correct answer (for every coordinate) with probability at least $\frac{2}{3}$ (as is standard for randomized algorithms).
3. If the algorithm runs in time $t(n)$ and uses $r(n)$ random bits on codes of length n, then it can be converted into a randomized algorithm producing the entire codeword in time $O(nt(n)\log n)$ by running it $O(\log n)$ times on each coordinate and taking majority votes and outputting the string so obtained. It can also be converted into a deterministic decoding algorithm running in time $O(n2^{r(n)}t(n))$ by simulating all random strings of length $r(n)$.

The following theorem sums up the unambiguous decoding results for Reed-Muller codes that we obtained in this lecture.

Theorem 15.17 *For a Reed-Muller code $\mathrm{RM}_{m,d,q}$ with $q > 5(d+1)$, the algorithm* **Improved RM decoder** *is a randomized algorithm that corrects upto a fraction $\frac{2}{15}$ error in time $\mathrm{poly}(m, d, \log q)$.*

The list-decoding version can be defined analogously. The list-decoding algorithm should output implicit representations of the ℓ-nearby codewords. We will be slightly more relaxed and allow the algorithm to output $\ell' \geq \ell$ algorithms, with the condition that every nearby codeword be represented (as also other algorithms that don't correspond to codewords). Thus we obtain the following problem:

Definition 15.18 (Implicit list decoding) *For a fixed family of codes $\mathcal{C}$, the implicit decoding problem is:*
GIVEN: *Parameters n, q (and any other parameters that are relevant) of the code $C \in \mathcal{C}$, error parameter $e < \frac{\Delta(C)}{2}$, and implicit representation of received vector $\mathbf{f} \in \mathbb{F}_q^n$.*
GOAL: *Output ℓ' algorithms $A_1, \ldots, A_{\ell'}$ such that for every codeword $\mathbf{c} \in C$ such that $\Delta(\mathbf{c}, \mathbf{f}) \leq e$ there exists an index $i \in [\ell']$ such that A_i with oracle access to $\mathbf{f}$ is an algorithm to compute $\mathbf{c}$.*

In terms of this definition, we can summarize our list-decoding algorithm as follows:

Theorem 15.19 *For a Reed-Muller code $\mathrm{RM}_{m,d,q}$ with $q > \frac{240(d+1)}{\gamma^2}$, for some $\gamma > 0$, the algorithm* **RM List-decoder** *is a randomized algorithm that corrects upto a fraction $1 - \gamma$ error in time $\mathrm{poly}(m, d, q)$.*

15.6 Bibliographic notes

We start with the implicit models discussed in the last section. It is hard to pin down the exact point where these models were first introduced. Often they were implicit in some technical results and then made explicit only by later works. They also have multiple origins within computer science — several works in the late eighties seem to converge on these models independently. One of the first works that used such models is that of Kaltofen and Trager [59], who applied it in the context of algebraic algorithms. This work explicitly focusses on the implicit models (ahem!) and thereby captured the efficiency of several algorithms very elegantly. Around the same time, the implicit input model was introduced into coding theory by Goldreich and Levin [38]. In the language of coding theory, they developed a list-decoding algorithm (the first non-trivial list-decoder!) for the Hadamard code in the implicit input model. They also showed a powerful application of list-decoding to the foundations of cryptography. (We might talk about this result in a later lecture.) Also, at the same time, such models were exploited to check or improve correctness of programs by Blum and Kannan [20] and Blum, Luby, and Rubinfeld [21]. As it turned out, such results could be interpreted as error-correction algorithms for some codes [108, Section 1.3]. Finally, the full-fledged model (implicit input and output, with list-decoding capabilities) were considered in Ar et al. [5] and Arora and Sudan [8], and made explicit in Sudan, Trevisan, and Vadhan [112].

The algorithms have a somewhat simpler history. (As always, notions are harder to trace than algorithms!) The simple algorithm for decoding Reed-Muller codes from Section 15.2 is based on an algorithm of Beaver and Feigenbaum [13], whose ability to decode all polynomials was pointed out by Lipton [70]. The improvement in Section 15.3 is due to Gemmell et al. [34]. Further improvements to this algorithm, decoding arbitrarily close to half the minimum distance for $q \gg d$ were given by Gemmell and Sudan [35]. The list-decoding algorithm in Section 15.4 is due to Sudan, Trevisan, and Vadhan [112]. This algorithm simplifies a previous list-decoding algorithm of Arora and Sudan [8].

Bibliography

[1] Erik Agrell, Alexander Vardy, and Kenneth Zeger. Upper bounds for constant-weight codes. *IEEE Transactions on Information Theory*, 46:2373–2395, 2000.

[2] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Anaylsis of Computer Algorithms*. Addison Wesley Publishing Company, 1974.

[3] Miklos Ajtai. The shortest vector problem is NP-hard for randomized reductions. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pages 10–19, Dallas, Texas, 23–26 May 1998.

[4] Noga Alon, Jehoshua Bruck, Joseph Naor, Moni Naor, and Ronny Roth. Construction of asymptotically good low-rate error-correcting codes through pseudo-random graphs. *IEEE Transactions on Information Theory*, 38:509–516, 1992.

[5] Sigal Ar, Richard Lipton, Ronitt Rubinfeld, and Madhu Sudan. Reconstructing algebraic functions from mixed data. *SIAM Journal on Computing*, 28(2):488–511, 1999.

[6] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM*, 45(3):501–555, May 1998.

[7] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: A new characterization of NP. *Journal of the ACM*, 45(1):70–122, January 1998.

[8] Sanjeev Arora and Madhu Sudan. Improved low-degree testing and its applications. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 485–495, El Paso, Texas, 4-6 May 1997.

[9] C. Asmuth and J. Bloom. A modular approach to key safeguarding. *IEEE Transactions on Information Theory*, 29:208–210, March 1983.

[10] László Babai, Lance Fortnow, and Carsten Lund. Non-deterministic exponential time has two-prover interactive protocols. *Computational Complexity*, 1(1):3–40, 1991.

[11] László Babai, Lance Fortnow, Noam Nisan, and Avi Wigderson. BPP has subexponential time simulations unless EXPTIME has publishable proofs. *Computational Complexity*, 3(4):307–318, 1993.

[12] L.A. Bassalygo. New upper boundes for error-correcting codes. *Problems of Information Transmission*, 1(1):32–35, 1965.

[13] Donald Beaver and Joan Feigenbaum. Hiding instances in multioracle queries. In C. Choffrut and T. Lengauer, editors, *Proceedings of the 7th Annual Symposium on Theoretical Aspects of Computer Science*, pages 37–48, Rouen, France, 22–24 February 1990. Springer.

[14] Elwyn Berlekamp. *Algebraic Coding Theory.* McGraw Hill, New York, 1968.

[15] Elwyn Berlekamp. Bounded distance +1 soft-decision Reed-Solomon decoding. *IEEE Transactions on Information Theory*, 42(3):704–720, 1996.

[16] Elwyn R. Berlekamp. Factoring polynomials over finite fields. *Bell System Technical Journal*, 46:1853–1859, 1967.

[17] Elwyn R. Berlekamp. Factoring polynomials over large finite fields. *Mathematics of Computation*, 24:713–735, 1970.

[18] Elwyn R. Berlekamp, Robert J. McEliece, and Henk C. A. van Tilborg. On the inherent intractability of certain coding problems. *IEEE Transactions on Information Theory*, 24(3):384–386, May 1978.

[19] Richard E. Blahut. *Theory and Practice of Error Control Codes.* Addison-Wesley, Reading, Massachusetts, 1983.

[20] Manuel Blum and Sampath Kannan. Designing programs that check their work. *Journal of the ACM*, 42(1):269–291, January 1995.

[21] Manuel Blum, Michael Luby, and Ronitt Rubinfeld. Self-testing/correcting with applications to numerical problems. *Journal of Computer and System Sciences*, 47(3):549–595, 1993.

[22] Dan Boneh. Finding smooth integers in short intervals using CRT decoding. *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing*, pages 265–272, 2000.

[23] R. C. Bose and D. K. Ray-Chaudhuri. On a class of error correcting binary group codes. *Information and Control*, 3:68–79, 1960.

[24] Jehoshua Bruck and Moni Naor. The hardness of decoding linear codes with preprocessing. *IEEE Transactions on Information Theory*, 36(2), March 1990.

[25] Henri Cohen. *A Course in Computational Algebraic Number Theory.* Graduate Texts in Mathematics **138**, Springer Verlag, Berlin, 1993.

[26] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory.* Wiley Publishing, New York, 1991.

[27] Phillipe Delsarte. An algebraic approach to the association schemes of coding theory. *Philips Research Reports*, Suppl. 10, 1973.

[28] Richard A. DeMillo and Richard J. Lipton. A probabilistic remark on algebraic program testing. *Information Processing Letters*, 7(4):193–195, June 1978.

[29] Ilya Dumer, Daniele Micciancio, and Madhu Sudan. Hardness of approximating the minimum distance of a linear code. *Proceedings of the 40th IEEE Symposium on Foundations of Computer Science*, pages 475–484, 1999.

[30] Iwan M. Duursma. *Decoding Codes from Curves and Cyclic Codes.* PhD thesis, Eindhoven University of Technology, 1993.

[31] Uriel Feige and Carsten Lund. On the hardness of computing the permanent of random matrices. *Computational Complexity*, 6(2):101–132, 1997.

[32] Arnaldo Garcia and Henning Stichtenoth. A tower of Artin-Schreier extensions of function fields attaining the Drinfeld-Vlădut bound. *Inventiones Mathematicae*, 121:211–222, 1995.

[33] Arnaldo Garcia and Henning Stichtenoth. On the asymptotic behavior of some towers of function fields over finite fields. *Journal of Number Theory*, 61(2):248–273, December 1996.

[34] Peter Gemmell, Richard Lipton, Ronitt Rubinfeld, Madhu Sudan, and Avi Wigderson. Self-testing/correcting for polynomials and for approximate functions. In *Proceedings of the Twenty Third Annual ACM Symposium on Theory of Computing*, pages 32–42, New Orleans, Louisiana, 6-8 May 1991.

[35] Peter Gemmell and Madhu Sudan. Highly resilient correctors for multivariate polynomials. *Information Processing Letters*, 43(4):169–174, September 1992.

[36] E. N. Gilbert. A comparison of signalling alphabets. *Bell System Technical Journal*, 31:504–522, May 1952.

[37] M. J. E. Golay. Notes on digital coding. *Proceedings of the IRE*, 37:657, June 1949.

[38] Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In *Proceedings of the Twenty First Annual ACM Symposium on Theory of Computing*, pages 25–32, Seattle, Washington, 15-17 May 1989.

[39] Oded Goldreich, Dana Ron, and Madhu Sudan. Chinese remaindering with errors. *IEEE Transactions on Information Theory*, 46(5):1330–1338, July 2000. Extended version appears as ECCC Technical Report TR98-062 (Revision 4), `http://www.eccc.uni-trier.de/eccc`.

[40] Oded Goldreich, Ronitt Rubinfeld, and Madhu Sudan. Learning polynomials with queries: The highly noisy case. *SIAM Journal on Discrete Mathematics*, 13(4):535–570, November 2000.

[41] V. D. Goppa. Codes associated with divisors. *Problems of Information Transmission*, 13(1):22–26, 1977.

[42] Daniel Gorenstein and Neal Zierler. A class of error-correcting codes in p^m symbols. *Journal of the Society for Industrial and Applied Mathematics*, 9:207–214, June 1961.

[43] Dima Grigoriev. Factorization of polynomials over a finite field and the solutions of systems of algebraic equations. *Translated from Zapiski Nauchnykh Seminarov Lenningradskogo Otdeleniya Matematicheskogo Instituta im. V. A. Steklova AN SSSR*, 137:20–79, 1984.

[44] Venkatesan Guruswami, Amit Sahai, and Madhu Sudan. Soft-decision decoding of Chinese Remainder codes. In *Proceedings of the 41st IEEE Symposium on Foundations of Computer Science*, pages 159–168, Redondo Beach, California, 12-14 November 2000.

[45] Venkatesan Guruswami and Madhu Sudan. Improved decoding of Reed-Solomon and algebraic-geometric codes. *IEEE Transactions on Information Theory*, 45:1757–1767, 1999.

[46] Venkatesan Guruswami and Madhu Sudan. List decoding algorithms for certain concatenated codes. *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing*, pages 181–190, 2000.

[47] Venkatesan Guruswami and Madhu Sudan. Extensions to the Johnson bound. *Manuscript*, February 2001.

[48] Richard W. Hamming. Error Detecting and Error Correcting Codes. *Bell System Technical Journal*, 29:147–160, April 1950.

[49] A. Hocquenghem. Codes correcteurs d'erreurs. *Chiffres (Paris)*, 2:147–156, 1959.

[50] Tom Høholdt, J. H. van Lint, and Ruud Pellikaan. Algebraic geometry codes. *Handbook of Coding Theory, Chapter 10*, 1998.

[51] Russell Impagliazzo and Avi Wigderson. P = BPP if E requires exponential circuits: Derandomizing the XOR Lemma. *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, pages 220–229, May 1997.

[52] Selmer M. Johnson. A new upper bound for error-correcting codes. *IEEE Transactions on Information Theory*, 8:203–207, 1962.

[53] Selmer M. Johnson. Improved asymptotic bounds for error-correcting codes. *IEEE Transactions on Information Theory*, 9:198–205, 1963.

[54] G. David Forney Jr. *Concatenated Codes*. MIT Press, Cambridge, MA, 1966.

[55] G. David Forney Jr. Generalized minimum distance decoding. *IEEE Transactions on Information Theory*, 12(2):125–131, April 1966.

[56] Jørn Justesen. A class of constructive asymptotically good algebraic codes. *IEEE Transactions on Information Theory*, 18:652–656, 1972.

[57] Jørn Justesen. On the complexity of decoding Reed-Solomon codes (corresp.). *IEEE Transactions on Information Theory*, 22(2):237–238, March 1976.

[58] Erich Kaltofen. Polynomial-time reductions from multivariate to bi- and univariate integral polynomial factorization. *SIAM Journal on Computing*, 14(2):469–489, 1985.

[59] Erich Kaltofen and Barry Trager. Computing with polynomials given by black boxes for their evaluations: Greatest common divisors, factorization, separation of numerators and denominators. *Journal of Symbolic Computation*, 9(3):301–320, 1990.

[60] G. L. Katsman, Michael A. Tsfasman, and Serge G. Vlădut. Modular curves and codes with a polynomial construction. *IEEE Transactions on Information Theory*, 30:353–355, 1984.

[61] Ralf Kötter. A unified description of an error locating procedure for linear codes. In *Proceedings of the International Workshop on Algebraic and Combinatorial Coding Theory*, pages 113–117, Voneshta Voda, Bulgaria, 1992.

[62] Ralf Kötter. Fast generalized minimum distance decoding of algebraic geometry and Reed-Solomon codes. *IEEE Transactions on Information Theory*, 42(3):721–737, May 1996.

[63] Ralf Kötter and Alexander Vardy. Algebraic soft-decision decoding of Reed-Solomon codes. *Proceedings of the 38th Annual Allerton Conference on Communication, Control and Computing*, pages 625–635, October 2000.

[64] H. Krishna, B. Krishna, K.-Y. Lin, and J.-D. Sun. *Computational Number Theory and Digital Signal Processing: Fast Algorithms and Error Control Techniques*. CRC Press Inc., Boca Raton, Florida, 1994.

[65] Arjen K. Lenstra. Factoring multivariate polynomials over finite fields. *Journal of Computer and System Sciences*, 30(2):235–248, April 1985.

[66] Arjen K. Lenstra, Hendrik W. Lenstra, and Lászlo Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261:515–534, 1982.

[67] V. I. Levenshtein. *Universal bounds for codes and designs*, pages 499–648. Volume 1 of Pless and Huffman [88], 1998.

[68] Rudolf Lidl and Harald Niedereitter. *Introduction to Finite Fields and Their Applications.* Cambridge University Press, 2nd edition, 1994.

[69] Nati Linial and Alex Samorodnitsky. Linear codes and sum of characters. *Combinatorica*, (To appear).

[70] Richard Lipton. New directions in testing. In *Distributed Computing and Cryptography*, volume 2 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 191–202. AMS, 1991.

[71] László Lovász. On the Shannon capacity of a graph. *IEEE Transactions on Information Theory*, 25:1–7, January 1979.

[72] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *Journal of the ACM*, 39(4):859–868, October 1992.

[73] Saunders MacLane and Garrett Birkhoff. *Algebra.* Chelsea Publishing Company, N.Y., 3rd edition, 1988.

[74] F. J. MacWilliams and Neil J. A. Sloane. *The Theory of Error-Correcting Codes.* Elsevier/North-Holland, Amsterdam, 1981.

[75] Florence Jessie MacWilliams. A theorem on the distribution of weights in a systematic code. *Bell Systems Technical Journal*, 42:79–94, January 1963.

[76] David M. Mandelbaum. On a class of arithmetic codes and a decoding algorithm. *IEEE Transactions on Information Theory*, 21(1):85–88, January 1976.

[77] Y. I. Manin and Serge G. Vlǎdut. Linear codes and modular curves. *J. Soviet. Math.*, 30:2611–2643, 1985.

[78] James L. Massey. *Threshold decoding.* MIT Press, Cambridge, Massachusetts, USA, 1963.

[79] James L. Massey. Shift-register synthesis and BCH decoding. *IEEE Transactions on Information Theory*, 15:122–127, January 1969.

[80] Robert J. McEliece, Eugene R. Rodemich, Howard Rumsey Jr., and Lloyd R. Welch. New upper bounds on the rate of a code via the Delsarte-MacWilliams inequalities. *IEEE Transactions on Information Theory*, 23:157–166, 1977.

[81] Daniele Micciancio. The shortest vector in a lattice is hard to approximate to within some constant. In *Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science*, pages 92–98, Palo Alto, California, 8–11 November 1998.

[82] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms.* Cambridge University Press, 1995.

[83] D. E. Muller. Application of Boolean algebra to switching circuit design and to error detection. *IEEE Transactions on Computers*, 3:6–12, 1954.

[84] Rasmus R. Nielsen. Decoding concatenated codes using Sudan's algorithm. *Manuscript submitted for publication*, May 2000.

[85] Noam Nisan and Avi Wigderson. Hardness vs randomness. *Journal of Computer and System Sciences*, 49(2):149–167, October 1994.

[86] Ruud Pellikaan. On decoding linear codes by error correcting pairs. Preprint, Eindhoven University of Technology, 1988.

[87] W. Wesley Peterson. Encoding and error-correction procedures for Bose-Chaudhuri codes. *IEEE Transactions on Information Theory*, 6:459–470, 1960.

[88] Vera S. Pless and W. Cary Huffman (Eds.). *Handbook of Coding Theory (2 Volumes)*. Elsevier, 1998.

[89] M. Plotkin. Binary codes with specified minimum distance. *IRE Transactions on Information Theory*, 6:445–450, 1960.

[90] Jaikumar Radhakrishnan. Personal communication, January 1997.

[91] Irving S. Reed. A class of multiple-error-correcting codes and the decoding scheme. *IEEE Transactions on Information Theory*, 4:38–49, 1954.

[92] Irving S. Reed and Gustav Solomon. Polynomial codes over certain finite fields. *J. SIAM*, 8:300–304, 1960.

[93] Michael Rosenblum. A fast algorithm for rational function approximations. Available from `http://theory.lcs.mit.edu/~madhu/FT01/notes/rosenblum.ps`, November 1999.

[94] Alex Samorodnitsky. *Applications of Harmonic Analysis in Combinatorics and in Coding Theory*. PhD thesis, Department of Mathematics, Hebrew University, 1998.

[95] Alex Samorodnitsky. On the Kabatyanskii-Levenshtein bound for sphere packing, 2000.

[96] Alex Samorodnitsky. On the optimum of Delsarte's linear program. *Journal of Combinatorial Theory*, (To appear).

[97] Arnold Schönhage. Schnelle berechnung von ketterbruchentwicklungen. *Acta Informatica*, 1:139–144, 1971.

[98] J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM*, 27(4):701–717, October 1980.

[99] Adi Shamir. IP = PSPACE. *Journal of the ACM*, 39(4):869–877, October 1992.

[100] Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 623–656, 1948.

[101] Claude E. Shannon, Robert G. Gallager, and Elwyn R. Berlekamp. Lower bounds to error probability for coding on discrete memoryless channels. *Information and Control*, 10:65–103 (Part I), 522–552 (Part II), 1967.

[102] Victor Shoup. New algorithms for finding irreducible polynomials over finite fields. *Mathematics of Computation*, 54:435–447, 1990.

[103] Kenneth Shum. *A Low-Complexity Construction of Algebraic Geometric Codes Better Than the Gilbert-Varshamov Bound*. PhD thesis, University of Southern California, December 2000.

[104] Kenneth W. Shum, Ilia Aleshnikov, P. Vijay Kumar, Henning Stichtenoth, and Vinay Deolalikar. A low-complexity algorithm for the construction of algebraic geometric codes better than the Gilbert-Varshamov bound. *IEEE Transactions on Information Theory*, 47(6):2225–2241, September 2001.

[105] Malte Sieveking. An algorithm for division of power series. *Computing*, 10:153–156, 1972.

[106] Richard C. Singleton. Maximum distance q-nary codes. *IEEE Transactions on Information Theory*, 10:116–118, April 1964.

[107] M.A. Soderstrand, W.K. Jenkins, G.A. Jullien, and F.J. Taylor. *Residue Number System Arithmetic: Modern Applications in Digital Signal Processing.* IEEE Press, New York, 1986.

[108] Madhu Sudan. *Efficient Checking of Polynomials and Proofs and the Hardness of Approximation Problems.* PhD thesis, University of California at Berkeley, October 1992. Also appears as *Lecture Notes in Computer Science*, vol. 1001, Springer, 1996.

[109] Madhu Sudan. Decoding of Reed-Solomon codes beyond the error-correction bound. *Journal of Complexity*, 13(1):180–193, 1997.

[110] Madhu Sudan. Decoding of Reed-Solomon codes beyond the error-correction diameter. *Proceedings of the 35th Annual Allerton Conference on Communication, Control and Computing*, 1997.

[111] Madhu Sudan. Notes on an efficient solution to the rational function interpolation problem. Available from `http://theory.lcs.mit.edu/~madhu/FT01/notes/rational.ps`, 1999.

[112] Madhu Sudan, Luca Trevisan, and Salil Vadhan. Pseudorandom generators without the XOR lemma. *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, pages 537–546, 1999.

[113] Aimo Tietavainen. On the nonexistence of perfect codes over finite fields. *SIAM Journal of Applied Mathematics*, 24(1):88–96, January 1973.

[114] Michael A. Tsfasman, Serge G. Vlădut, and Thomas Zink. Modular curves, Shimura curves, and codes better than the Varshamov-Gilbert bound. *Math. Nachrichten*, 109:21–28, 1982.

[115] Rüdiger Urbanke. *Modern Coding Theory – SS2001.* EPFL, DSC-LTHC, Available from `http://lthcwww.epfl.ch/content.php?title=coding2001`, May 15 2001.

[116] Jacobus H. van Lint. Nonexistence theorems for perfect error-correcting codes. In G. Birkhoff and M. Hall Jr., editors, *Proceedings of the Symposium on Computers in Algebra and Number Theory, New York, 1970*, pages 89–95. American Mathematical Society, Providence, RI, 1971.

[117] Jacobus H. van Lint. *Introduction to Coding Theory.* Graduate Texts in Mathematics **86**, (Third Edition) Springer-Verlag, Berlin, 1999.

[118] Alexander Vardy. The intractability of computing the minimum distance of a code. *IEEE Transactions on Information Theory*, 43:1757–1766, November 1997.

[119] R. R. Varshamov. Estimate of the number of signals in error correcting codes. *Doklady Akadamii Nauk*, 117:739–741, 1957.

[120] Robert J. Walker. *Algebraic Curves.* Springer-Verlag, 1978.

[121] R. W. Watson and C. W. Hastings. Self-checked computation using residue arithmetic. In *Proceedings of the IEEE*, volume 44, pages 1920–1931, December 1966.

[122] Lloyd R. Welch and Elwyn R. Berlekamp. Error correction of algebraic block codes. *US Patent Number 4,633,470*, December 1986.

[123] Edward J. Weldon, Jr. Justesen's construction — the low-rate case. *IEEE Transactions on Information Theory*, 19:711–713, 1973.

[124] J. M. Wozencraft. Threshold decoding. Personal communication in [78, Section 2.5], 1963.

[125] Richard E. Zippel. Probabilistic algorithms for sparse polynomials. In *EUROSAM '79, Lecture Notes in Computer Science*, volume 72, pages 216–225, 1979.

[126] Victor V. Zyablov. An estimate on the complexity of constructing binary linear cascade codes. *Problems of Information Transmission*, 7(1):3–10, 1971.