Contents

1 Lecture 1

1.1 What are Error-Correcting Codes?

Error-correcting codes provide a judiciously designed, noise-resilient redundant form of data.

The idea of "coding data", is robustness against noise through redundancy and the possibility of error recovery (pinpointing mistakes). Our hopes are:

- It is information-theoretically possible to recover the data
- Minimize redundancy (i.e. overhead)
- The algorithm to encode/recover the original data is efficient

Error-correcting codes have plenty of applications:

- Communication
- Storage (can be thought of communication in time)
- Central tool in discrete math, theoretical CS, information theory
- Quantum computing (heat and noise is present in every computation!)

We have many noise models in thinking about what we want our codes to protect against. We can mix and match to make a problem.

- Erasures, errors, deletions
- Bit level vs symbol/packet level
- Worst-case noise vs probabilistic/stochastic noise (or some hybrid model)
- Many desiderata/decoding models: local decoding/list decoding

The flow of a coding model is as follows.

1. Encoding: info $\to$ codeword
2. Noise: codeword $\to$ noised encoding
3. Decoding (Error Correction): noised encoding $\to$ info

The encoding map is generally injective (you can go from "valid" codeword to info easily).

1.2 One-Bit Erasures

Definition 1.1
A (binary) code is some subset $C \subseteq \{0,1\}^n$. The elements of C are called “codewords”.

Suppose our noise is such that one bit is missing (and the location of the noise is known). So the transformation was:

$$c_1c_2\dots c_n \mapsto c_1c_2\dots c_{i-1}?c_{i+1}\dots c_n$$

we want to find what was in the question mark.

A simple code is repeating the same bit twice:

$$C_{rep} = \{(c_1, \ldots, c_n) \mid c_1 = c_2, c_3 = c_4, \ldots, c_{n-1} = c_n\}$$

Then the question mark is clearly recoverable from the neighbor bit.

A better code that works with this is one with only even parity (called the parity-check code).

$$C_{pc} = \{(c_1, \ldots, c_n) \in \{0,1\}^n \mid c_1 + c_2 + \cdots + c_n \equiv 0 \pmod 2\}$$

Then, the question mark is just the parity of the rest, i.e.

$$? = c_1 + c_2 + \cdots + c_{i-1} + c_{i+1} + \cdots + c_n \bmod 2$$

The second one is better since we can represent more messages, i.e.

$$|C_{pc}| = 2^{n-1} > |C_{rep}| = 2^{\frac{n}{2}}$$

The number of redundant bits in C_{pc} is 1, while there are $n/2$ redundant bits in C_{rep}. Here are the encoding maps:

$$(x_1, \ldots, x_{n-1}) \mapsto (x_1, \ldots, x_{n-1}, \bigoplus_i x_i)$$

$$(x_1, \ldots, x_{n/2}) \mapsto (x_1, x_1, x_2, x_2 \ldots, x_{n/2}, x_{n/2})$$

Fact: C_{pc} is the BEST code to correct 1-bit erasure (it has the least overhead). Proof shell: suppose you had a code with more codewords. This means that there are some codewords that differ by at most 1 bit, but then if you erase that bit the noisy data is ambiguous.

To generalize this setup there are two models. Consider $\alpha < 1$:

1. Hamming model: worst-case, erase any αn out of n bits (zero-error info theory)
2. Shannon model: stochastic, erase each bit independently with probability α

1.3 One-Bit Errors

In this setup, one bit is flipped but we don't know which one.

$$c_1 c_2 \ldots c_n \mapsto c_1 c_2 \ldots c_{i-1} \overline{c_i} c_{i+1} \ldots c_n$$

Now, we can replicate the message 3 times as a code; we can use majority vote to correct one bit. But $\frac{2}{3}$ of the bits are redundant! This means $|C| = 2^{n/3}$. Let's see if we can be smarter.

$$C_{VT} = \{(c_1, c_2, \ldots, c_n) \mid c_1 + 2c_2 + \cdots + nc_n \equiv a \pmod B\}$$

This is a "weighted" parity check. The claim is that if $B > 2n$, then you can correct 1-bit errors. Proof shell: suppose you flip a bit, it will go up or down by at most n and this uniquely identifies which bit got flipped. If B is larger, note that there will be less codewords, so we'd prefer B to be as low as possible.

Since there are some that are bigger than average, there exists an $a \in \{0, \ldots, B-1\}$ such that $|C_{VT}| \geq \frac{2^n}{B} = \frac{2^n}{2n+1}$.

Definition 1.2 (Rate)
An **alphabet** Σ are the characters in the codeword.
The number of redundant bits is $n - \log_{|\Sigma|} |C|$. The **rate** of C is $R(C) = \frac{\log_{|\Sigma|} |C|}{n}$, i.e. the fraction of

non-redundant bits in your code. Note $|C| = |\Sigma|^{Rn}$.

So our code has $\leq \log_2 n + O(1)$ redundant bits, so a rate of nearly 1 asymbtotically.

Theorem 1.1 (Hamming Bound)
If $C \subseteq \{0,1\}^n$ corrects 1-bit flips, then $|C| \leq \frac{2^n}{n+1}$.

Here is a simple proof: consider the hamming ball of distance 1 (i.e. everything produced by 0 or 1 bit flips). These balls cannot overlap (this would imply ambiguous decoding). Thus, by pigeonhole there can only be at most $\frac{2^n}{n+1}$ such balls.

1.4 More than 1-bit flips

To correct two bit errors we can build off our old code:

$$\alpha_1 c_1 + \cdots + \alpha_n c_n \equiv m \pmod{M}$$

where $\pm\alpha_p \pm \alpha_q \neq \pm\alpha_r \pm \alpha_s$ for any 4 tuple. This is sufficient to avoid ambiguity.

We can also add a second check:

$$1^2 c_1 + 2^2 c_2 + \cdots + n^2 c_n \equiv a' \pmod{A}$$

where $A = O(n^2)$. This is a sufficient α because $f(n) = n^2$ is strictly convex. Thus, with the old check together, this makes a code strong enough. Similar to before:

$$|C| \geq \frac{2^n}{A \cdot B} = \Omega\left(\frac{2^n}{n^3}\right)$$

Theorem 1.2 (Generalized Hamming Bound)
If $C \subseteq \{0,1\}^n$ corrects (up to) k-bit flips, then $|C| \leq \frac{2^n}{\sum_{i=0}^{k}\binom{n}{k}} = O\left(\frac{2^n}{n^k}\right)$.

This follows by a similar ball argument, except you consider up to k bit errors.

1.5 Hamming Code

Let's revisit the single-bit flip case. Our "VT Check" was:

$$1c_1 + 2c_2 + \cdots + nc_n \equiv 0 \pmod{B}$$

Let's use linear algebra instead and map ic_i to $\mathbf{v_i}c_i$, thus writing

$$\mathbf{v_1}c_1 + \mathbf{v_2}c_2 + \cdots + \mathbf{v_n}c_n = \mathbf{0}$$

where we take operations mod 2.

To take a (potentially corrupted) $\mathbf{c}$ we can check the following using the parity check matrix on the left.

$$\begin{pmatrix} \mathbf{v_1} & \mathbf{v_2} & \cdots & \mathbf{v_n} \end{pmatrix} \begin{pmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{pmatrix} = \mathbf{0}$$

where m, the height of the matrix, is the total amount of bits to represent all the codewords. $m = \lceil \log_2(n+1) \rceil$. This can still correct a one-bit flip!

One can think of any codeword as a set of column linearly dependencies.

Definition 1.3 (Code notation)
A linear $[n, k, d]_q$ code has block size n, dimension k, minimum distance d and alphabet size q. The dimension k is the dimension of C as a subspace.

If we choose $m = 3$, what we constructed above is the $[7, 4, 3]_2$ Hamming code. And the parity check matrix is:

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

To fix an error:

$$\begin{aligned} Hy &= H(c + e_i) \\ &= Hc + He_i \\ &= 0 + (\text{binary representation of } i) \end{aligned}$$

where the quantity Hy is defined as the "syndrome."

We argue this achieves the Hamming bound. Suppose $n = 2^m - 1$ exactly. Note that $\dim(C_{Ham}) = 2^m - 1 - m$ (because the rank of the parity-check matrix is m) so

$$|C_{Ham}| = 2^{2^m - 1 - m} = \frac{2^n}{2^m} = \frac{2^n}{n+1}$$

Let's revisit our two-bit error correction with our Hamming knowledge. Let's write the equations in matrix vector form.

$$\begin{pmatrix} 1 & 2 & \dots & n \\ 1^2 & 2^2 & \dots & n^2 \end{pmatrix} \begin{pmatrix} c_1 \\ \vdots \\ c_n \end{pmatrix} = \begin{pmatrix} a \\ b \end{pmatrix}$$

To match the setup we made in Hamming, we would want some check like

$$\begin{pmatrix} \mathbf{v_1} & \mathbf{v_2} & \dots & \mathbf{v_n} \\ \mathbf{v_1}^2 & \mathbf{v_2}^2 & \dots & \mathbf{v_n}^2 \end{pmatrix} \begin{pmatrix} c_1 \\ \vdots \\ c_n \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

We want a nice canonical way to square vectors. We can identify the m-bit vectors with a field, meaning we can add and multiply them. Formally:

$$\mathbb{F}_2 m \simeq \mathbb{F}_2^m$$

In particular, this identification are polynomials $\mathbb{F}_2$ of degree up to $m - 1$.

Note 1.1
If you have $f, g \in \mathbb{F}_2[x]$, then $(f(x) + g(x))^2 = f(x)^2 + g(x)^2$. (The cross term has coefficient 2, which is 0 mod 2).

This means squaring the vectors doesn't actually get us anything new that we couldn't do with the linear terms. So, let's instead define C_{BCH_2}, with parity check matrix:

$$H = \begin{pmatrix} \mathbf{v_1} & \mathbf{v_2} & \dots & \mathbf{v_n} \\ \mathbf{v_1}^3 & \mathbf{v_2}^3 & \dots & \mathbf{v_n}^3 \end{pmatrix}$$

Exercise: Show that C_{BCH_2} can correct 2-bit flips, i.e. show that if:

$$\alpha \neq \beta, \gamma \neq \delta$$
$$\alpha + \beta = \gamma + \delta$$
$$\alpha^3 + \beta^3 = \gamma^3 + \delta^3$$

then $\{\alpha, \beta\} = \{\gamma, \delta\}$

2 Lecture 2

2.1 The Basic Definitions

We spam definitions which will help us formalize all the crazy stuff we saw last time.

Definition 2.1 (Distance of a Code)
$C \subseteq \Sigma^n$. Then the **distance** of the code is:

$$d(C) = \min_{c \neq c' \in C} \delta(c, c')$$

This is related to the "packing radius," which is the largest radius τ such that Hamming balls of radius τ around codewords are disjoint. This is exactly $\tau = \left\lfloor \frac{d-1}{2} \right\rfloor$.

The reason distance is so important is that it tells you how robust a code is.

Theorem 2.1 (Distance Property)
Consider a code $C \subseteq \Sigma^n$. The following are equivalent:

- C has minimum distance at least $s + 1$
- C can detect up to s errors
- C can correct up to s erasures

In addition, you can correct up to e errors if and only if the minimum distance is at least $2e + 1$.

This is true pretty intuitive (Hamming balls should not touch) but the proofs are not hard to figure out. What about both erasures and errors together?

Theorem 2.2 (Mixed Distances)
$C \subseteq \Sigma^n$ can correct e errors and s erasures if and only if $2e + s < d(C)$.

However, this distance analysis is a worst-case analysis. However, for typical codewords, you will not go in the direction of exactly another codeword! For stochastic errors, you can correct way more (with high probability).

Even with worst-case analysis, you can "LIST DECODE" many more errors (in fact almost $\approx d(C)$). By "LIST DECODE", we give a list of possible codewords we could've gotten.

2.2 Linear Codes

The Hamming code we saw last time with parity check

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

is an example of a linear code.

Definition 2.2 (Linear Code)
A q-ary linear code is a set $C \subseteq \mathbb{F}_q^n$ is a subspace of $\mathbb{F}_q^n$.

The nice thing about subspaces is you can represent them with little space, despite the fact that the size of C is exponential in the rate, since $|C| = q^{Rn}$. We can represent C by a basis:

$$G = \begin{bmatrix} g_1 & g_2 & \dots & g_k \end{bmatrix} \in \mathbb{F}_q^{n \times k}$$

where $C = \text{range}(G)$ or $C = \{Gx \mid x \in \mathbb{F}_q^k\}$.

Definition 2.3 (Generator Matrix)
Suppose you have the above basis for C, G. Then G is called a generator matrix for C. In turn, $\dim(C) = k$ and the rate is $R = \frac{k}{n}$. Note that G is not unique.

Furthermore, you can find the normal vectors to C, i.e. describe it as the nullspace of some transformation.

Definition 2.4 (Parity Check Matrix)
For a code C, the parity check matrix $H \in \mathbb{F}_q^{(n-k) \times n}$ is a matrix such that for any codeword c, $Hc = 0$. Furthermore, if G is the generator matrix of C, this implies $HG = 0$.

Both of these matrices are defined as full rank.

Hamming distance is translation invariant, and subspace contains the difference between two codewords. So:

Theorem 2.3
For any linear code:

$$d(C) = \min_{c \neq 0} \text{wt}(c)$$

where

$$\text{wt}(c) = |\{i \mid c_i \neq 0\}|$$

We can actually formulate distance in terms of the parity check matrix.

$$\min\{\text{wt}(c) \mid Hc = 0, c \neq 0\}$$

i.e. this is the sparsest linear dependency of any columns of H.

Definition 2.5 (Relative Distance)
The relative distance of a code C of length n is:

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

In general, there are a few trade-offs:

- k vs d trade-off
- R vs δ trade-off

We are going to see this trade-off in a bound.

Theorem 2.4 (Pigeonhole/Singleton bound)
Consider $C \subseteq [q]^n$. Then:

$$|C| \leq q^{n - d(C) + 1}$$

Proof
Consider the map:

$$\text{proj} : C \to [q]^{n-d(C)+1}$$
$$(c_1, c_2, \ldots, c_n) \mapsto (c_1, c_2, c_{n-d(C)+1})$$

proj is a one-to-one function, because if two codewords c, c' differ in the last bits cut off, their distance would be less than $d(C)$. Since it is one-to-one, we have $|C| \leq q^{n-d+1}$.

For any linear code, this means: $q^k \leq q^{n-d+1}$, or $d \leq n - k + 1$.

With linear algebra comes duality, and linear codes are no different.

Definition 2.6 (Dual of a Linear Code)
The dual of C is:

$$C^\perp = \{y \in \mathbb{F}_q^n \mid y \cdot c, \forall c \in C\}$$

where the dot product is done mod q. Note that $C^\perp$ is also a linear code (i.e. a subspace).

Another way to think of $C^\perp$ is the space of all parity checks that codewords of C must satisfy.

Note 2.1
We have

- $C^\perp = \text{range } H^T$
- the parity check matrix of the dual is G^T
- $(C^\perp)^\perp = C$

In a real vector space, $C \cap C^\perp = \{0\}$. However, over finite fields, it's also possible that $C \subseteq C^\perp$ (a self-orthogonal code) or $C = C^\perp$. For example, $C_{Ham}^\perp \subseteq C_{Ham}$.

It turns out, the dual to C_{Ham} is important. It is called the simplex code $C_{Simplex}$. The generator matrix is nice:

$$\begin{bmatrix} 0 & \ldots & 0 & 1 \\ 0 & \ldots & 1 & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & \ldots & 1 & 1 \end{bmatrix}$$

which in general is a $(2^r - 1) \times r$ matrix. The code iswhich has rate:

$$\text{Rate}(C_{simplex}) = \frac{r}{2^r - 1} \to 0, \text{Rate}(C_{Ham}) = 1 - \frac{r}{2^r - 1} \to 1$$

But the distance is much better!

Fix an $x \in \mathbb{F}_2^n$.

$$\mathbb{P}\left[z \in \mathbb{F}_2^n\right][x \cdot z = 1] = \frac{1}{2}$$

So the hamming weight will be at least half n, i.e.

$$d(C_{simplex}) = 2^{r-1}$$

So, we now ask, can we have both R and δ bounded away from zero?

3 Lecture 3

3.1 Asymptotically Good Codes

As we have seen previously, the Hamming code is a $[2^r - 1, 2^r - r - 1, 3]_2$ code. To generalize the code to any length finite field, we want to do the same thing (having the columns of the parity check count up) and remove linear dependencies (we don't want any two columns to be linearly dependent, because that would imply the minimum distance is 2).

We also saw its dual is a $[2^r - 1, r, 2^{r-1}]_2$ code. The first two come from dual properties, the third we proved last time. These are two opposite ends of the spectrum. The Hamming code is optimal for distance 3 (by the Hamming bound). Simplex code is also optimally-sized for its distance.

Theorem 3.1
If $C \subseteq \{0,1\}^n$ is a code of distance $d > \frac{n}{2}$ then

$$|C| \leq \frac{d}{d - \frac{n}{2}}$$

Proof shell: Map a codeword $c = (c_1, \ldots, c_n) \mapsto v_c = \frac{1}{\sqrt{n}}((-1)^{c_1}, \ldots, (-1)^{c_n})$. Then for any two codewords c_1, c_2, we have:

$$v_{c_1} \cdot v_{c_2} = 1 - \frac{2\Delta(c_1, c_2)}{n}$$

If $d > \frac{n}{2}$, then, $v_{c_1} \cdot v_{c_2} < 0$.

Lemma 1 *If $v_1, \ldots, v_m \in \mathbb{R}^n$, $\|v_i\| = 1$, such that $v_i \cdot v_j \leq -\alpha \; \forall 1 \leq i \leq j \leq m$, then $m \leq \frac{1}{\alpha} + 1$.* □

From here you can apply this theorem with the correct α.
Furthermore, the bound for a q-ary code is of distance $d > \left(1 - \frac{1}{q}\right)n$,

$$|C| \leq \frac{d}{d - \left(\frac{q-1}{q}\right)n}$$

As an aside, if you have $v_i \cdot v_j \leq 0$, then $m \leq 2n$ and $v_i \cdot v_j \leq \epsilon$ is exponentially many.

For the simplex code, $|C_{simplex}| = 2^r$ and $d = 2^{r-1}$ and $n = 2^r - 1$, so

$$\frac{2d}{2d - n} = 2^r = |C_{simplex}|$$

So our goal is:

Definition 3.1 (Asymptotically Good Family)
An asymptotically good family of codes is an infinite family of codes over a fixed alphabet $[q]$

$$C_1, C_2, \ldots$$

Where length n_i of C_i goes to ∞ as $i \to \infty$, where

$$R(C_i) \geq R_0, \delta(C_i) \geq \delta_0$$

for some $R_0, \delta_0 > 0$ which are absolute constants.

3.2 Gilbert-Varshamov Bound

How do we know such asymptotically codes exist? By using bounds.

Theorem 3.2 (Gilbert-Varshamov Bound)
For all $q, n, d \leq n$ there exists a code $C \subseteq [q]^n$ with $d(C) \geq d$ and

$$|C| \geq \frac{q^n}{\sum_{i=1}^{d-1} \binom{n}{i}(q-1)^i}$$

Proof
The denominator here is $|B_q(0, d-1)| = \{x \in [q]^n \mid \text{wt}(x) \leq d-1\}$, i.e. the volume of the Hamming ball around 0 of size $d-1$. Greedily pick codewords such that no codeword is not within $d-1$ of any previously chosen ones. Stop when you can't, the balls must cover the space. The size union of the balls is no more than the sum of their sizes.

$$|C| \cdot |B_q(0, d-1)| \geq q^n$$

which implies the result.

In fact you can even modify this argument so that it gives you a linear code (over $\mathbb{F}_q$). Fill in columns of a $m \times n$ parity check matrix "greedily" (such that no $d-1$ columns are linearly dependent). For the last column, the amount of linear combinations are $\sum_{i=0}^{d-2} \binom{n-1}{i}(q-1)^i$ (this basically proves the claim). If this is less than q^m, then we'd never run out vectors. If it's MUCH BIGGER, then a random parity check matrix (RLC - random linear code) works with high probability.

We can also define an RLC via a $n \times k$ generator matrix. This works if:

$$q^k \ll \frac{q^n}{\sum_{i=1}^{d-1} \binom{n}{i}(q-1)^i}$$

Let us think about the asymptotic form, as n is large. For binary, fix relative distance $\delta \in [0, \frac{1}{2}]$. Note that binomial coefficients grow according to:

$$\frac{2^{h(\delta)n}}{\text{poly}(n)} \leq \binom{n}{\delta n} \leq 2^{h(\delta)n}$$

where the binary entropy is:

$$h(x) = x \log \frac{1}{x} + (1-x) \log \frac{1}{1-x}$$

Furthermore:

$$\sum_{i=0}^{\delta n} \binom{n}{i} \leq 2^{h(\delta)n}$$

By the GV bound:

$$\begin{aligned} |C| &\geq \frac{2^n}{\sum_{i=0}^{d-1} \binom{n}{i}} \\ &\geq \frac{2^n}{\sum_{i=0}^{\delta n} 2^{h(\delta)n}} \\ |C| &\geq 2^{(1-h(\delta))n} \end{aligned}$$

Thus,

Theorem 3.3 (Asymptotic Form of GV Bound)
There exists a binary linear code C with $\delta(C) = \delta$ and rate $R(C) \geq R_{GV}(\delta) = 1 - h(\delta)$.

So the rate graph over delta is the upside-down entropy plot (convex instead of concave).

In the q-ary case, we need to change the interval to $\delta \in [0, 1 - \frac{1}{q}]$, and using entropy function

$$h_q(x) = x \log_q(q-1) + x \log_q \frac{1}{x} + x \log_q \frac{1}{1-x}$$

which is the entropy of a random variable which is 0 with probability x and any value 1 to q with uniform other probability. If q is large, we approach the singleton bound.

For binary, no one knows if the asymptotic GV bound can be beaten (even existentially). It is known that for $q \geq 49$ that you can beat the bound. Also note that any algorithm trying to achieve the GV bound is either:

- Exponential in runtime (Greedy)
- Efficient to sample (RLC), but no known way to certify in polynomial time and no known way to decode.

In fact the lack of decoding is a popular hardness assumption in cryptography. This is called "Learning parity with noise."

Let's look at the regimes at the ends of the GV bound. If $\delta \to 0$, then

$$R_{GV}(\delta) = 1 - O\left(\delta \log\left(\frac{1}{\delta}\right)\right)$$

and if $\delta = \frac{1}{2} - \epsilon$, then

$$R_{GV}\left(\frac{1}{2} - \epsilon\right) = \Theta(\epsilon^2)$$

However, constructively, you can get from explicity construction: If $\delta \to 0$, then

$$R_{GV}(\delta) = 1 - O\left(\delta \cdot \text{polylog}\left(\frac{1}{\delta}\right)\right)$$

and if $\delta = \frac{1}{2} - \epsilon$, then

$$R_{GV}\left(\frac{1}{2} - \epsilon\right) = \Omega\left(\epsilon^{2+o(1)}\right)$$

We will achieve the SOTA with expander codes, expander walks, and pseudorandomness.

4 Lecture 4

4.1 More Bounds on Codes

The GV bound told us that there exists a code that achieves the following tradeoff between distance and rate:

$$R_{GV}(\delta) = 1 - h(\delta)$$

The limitations (upper bounds) of codes are the following:

$$R_{Plotkin}(\delta) = 1 - 2\delta$$
$$R_{Hamming}(\delta) = 1 - h(\delta/2)$$

which also has generalizations to q-ary. We will study codes in the blue region:

However, we can improve both Plotkin and Hamming to the EB bound.

Theorem 4.1 (Elias-Bassalygo Bound)
Consider a binary code C. Calling $J(\delta)$ the Johnson radius:

$$J(\delta) = \frac{1 - \sqrt{1 - 2\delta}}{2}$$

then we have:

$$R(\delta) \leq R_{EB}(\delta) = 1 - h(J(\delta))$$

For q-ary, the bound uses:

$$J_q(\delta) = \left(1 - \frac{1}{q}\right)\left(1 - \sqrt{1 - \frac{q\delta}{q-1}}\right)$$

Proof
We are going to take the Hamming bound and improve it, being a bit more careful using a list decoding idea. We previously take balls of $\frac{d}{2} = \delta/2n$. Now we want to expand the radius to some ρn and prove a bound like

$$|C| \cdot 2^{h(\rho)n} \leq 2^n$$

However, we might end up double counting some points (the balls will not be disjoint at higher radii). So, if each point is counted L times we can prove something like:

$$|C| \cdot 2^{h(\rho)n} \leq L \cdot 2^n$$

If this exponent is small, this will give us a good asymptotic bound. We will show that for $\rho = J(\delta)$, one can take $L = O(n)$.
Equivalently, consider an arbitrary point $y \in \{0,1\}^n$ then:

$$|B(y, \rho n) \cap C| \leq O(n)$$

i.e. with any codeword, we can list-decode up to ρn errors, with list size $O(n)$. This tells us list-decoding gives you even more give! By translation, assume $y = 0$. Recall the map:

$$f : c \mapsto v_c = ((-1)^{c_1}, (-1)^{c_2}, \ldots, (-1)^{c_n})$$

Note $f(y) = (1, 1, \ldots, 1)$. Suppose you have a bunch of codewords: $c_1, c_2, \ldots, c_m$ such that:

$$\Delta(c_i, c_j) \geq \delta n, \Delta(0, c_i) \leq \rho n$$

Consider v_i of the codewords that are in the Hamming ball of the correct radius of v_y, number them v_i. Pick a parameter α to ensure

$$\langle v_i - \alpha v_y, v_j - \alpha v_y \rangle \leq 0$$

which would imply $m \leq 2n$ (by the g). This can be rewritten as:

$$\langle v_i, v_j \rangle - 2\alpha\big(\langle v_y, v_i \rangle\big) + \alpha^2 n$$

Note that the first term is at most $n - 2\delta n$, since these codewords differ in at least δn places. Furthermore, the last term is bounded by the ball distance, i.e. this means:

$$\begin{aligned}\langle v_i - \alpha v_y, v_j - \alpha v_y \rangle &\leq n - 2\delta n + \alpha^2 n - 2\alpha(1-2\rho)n \\ &= n(1 - 2\delta + \alpha^2 - 2\alpha + 4\rho\alpha)\end{aligned}$$

Note that for that inner product to be negative,

$$4\rho \leq 2 - \left(\frac{1-2\delta}{\alpha} + \alpha\right)$$

which means that $\alpha = \sqrt{1-\delta}$ works as the best upper bound (by AM-GM).

Note that if $\delta \in (0, \frac{1}{2}]$, then $\frac{1-\sqrt{1-2\delta}}{2} > \frac{\delta}{2}$ so this is better than the Hamming bound. Furthermore, $\forall x \in [0, \frac{1}{2}], h(x) \geq 4x(1-x)$, so this is better than the plotkin bound as well. So this will make the blue region smaller!

It turns out for list decoding purposes,

$$J_L(\delta) = \frac{1 - \sqrt{1 - 2\delta + \frac{2\delta}{L}}}{2}$$

gives you exactly L list entries.

Also note that sample two strings, where you make it a 1 with probability ρ and a 0 otherwise, then their expected distance is $2J(\delta)(1 - J(\delta)) = \delta$. In other words, $J(\delta)$ is the solution to this polynomial.

The best bound for the rate is the MRRW Linear Programming Bound.

First, the original bound was weaker than R_{EB} for $\delta \leq 0.15$.

$$R_{MRRW1}(\delta) = h\left(\frac{1}{2} - \sqrt{\delta(1-\delta)}\right)$$

The better bound is pretty dank

$$R_{MRRW2}(\delta) = \min_{\delta/2 \leq \rho \leq 1/2} (1 - h(\rho) + R(\rho, \delta))$$

where $R(\cdot, \cdot)$ is crazy! Where does such a bound come from?

Consider a graph $G = (\{0,1\}^n, E)$ where $(u, v) \in E \iff \Delta(u, v) < d$. Then a code $C \subseteq \{0,1\}^n$ of distance $d(C) \geq d$ is just a independent set in G! This can be solved with LP, which is where these bounds come from.

4.2 Reed-Solomon Codes

Let's look at another class of codes, BCH.

Now to define BCH, first take a field extension and distinct nonzero elements:

$$\alpha_1, \alpha_2, \ldots, \alpha_n \in \mathbb{F}_{2^m}$$

The BCH code is represented by the parity check matrix, parameterized by $t \ll n$:

$$H = \begin{bmatrix} 1 & 1 & \dots & 1 \\ \alpha_1 & \alpha_2 & \dots & \alpha_n \\ \alpha_1^3 & \alpha_2^3 & \dots & \alpha_n^3 \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_1^{2t-1} & \alpha_2^{2t-1} & \dots & \alpha_n^{2t-1} \end{bmatrix}$$

For very small δ, Binary BCH codes approach the Hamming bound, i.e.

$$d = 2t + 1, k = n - t\left\lceil \log_2 n \right\rceil = O(1)$$

while GV bound only guarantees $k \approx n - 2t \log n$. The proof of the distance is pretty easy, just show any $2t$ columns are linearly independent. Note that Hamming was just BCH with $t = 1$!

Now we will construct Reed-Solomon codes over a field $\mathbb{F}_q$ over some choice of set $S \subseteq \mathbb{F}_q$ where $S = \{a_1, a_2, \dots, a_n\}$. We work with $1 \le k \le n \le q$:

$$RS_{\mathbb{F}_q}(S, k) = \{(f(a_1), f(a_2), \dots, f(a_n)) \mid f \in \mathbb{F}_q[X], \deg(f) < k\}$$

There is now a natural encoding:

$$f(X) = f_0 + f_1 X + \cdots + f_{k-1} X^{k-1}$$

Where we can associate:

$$(f_0, f_1, \dots, f_{k-1}) \mapsto (f(a_1), f(a_2), \dots, f(a_n))$$

Note the important following fact.

Theorem 4.2 (Interpolation)
For some field $\mathbb{F}$, all $a_1, \dots, a_k \in \mathbb{F}$ and distinct $b_1, \dots, b_k \in \mathbb{F}_q$, there is a unique polynomial $f(X) \in \mathbb{F}_q[X]$ of degree less than k such that $f(a_i) = b_i$ for $i = 1, 2, \dots, k$.

This means a lot for our code. What is the distance of the code? We consider some erasures.

$$(f(a_1), f(a_2), \dots, f(a_n)) \to (f(a_1), ?, ?, f(a_4), \dots, f(a_{n-2}), ?, f(a_n))$$

We only need k points to survive to recover the original polynomial. So $s = n - k$ erasures are correctable. Thus, the distance is $d \ge n - k + 1$, which is exactly the Singleton bound, so $d = n - k + 1$ exactly! The generator matrix of this $[n, k, n - k + 1]_{\mathbb{F}}$ code is just the Vandermonde matrix:

$$\begin{bmatrix} 1 & a_1 & a_1^2 & \dots & a_1^{k-1} \\ 1 & a_2 & a_2^2 & \dots & a_2^{k-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & a_n & a_n^2 & \dots & a_n^{k-1} \end{bmatrix}$$

5 Lecture 5

5.1 Reed-Solomon Continued

What is the dual of the Reed-Solomon code? It's a (generalized) Reed-Solomon code. Let's consider a special case.

Theorem 5.1
Consider $RS_{\mathbb{F}_q}(\mathbb{F}_q^*, k)$. We know $\mathbb{F}^*$, the nonzero elements of $\mathbb{F}_q$, is a cyclic group $\mathbb{F}^* = \{1, \alpha, \alpha^2, \ldots, \alpha^{q-2}\}$. The parity checks of this code is:

$$\{(c_0, c_1, \ldots, c_{n-1}) \in \mathbb{F}^n \mid c(\alpha) = c(\alpha^2) = \cdots = c(\alpha^{n-k}) = 0\}$$

where $c(x) = c_0 + c_1 x + \cdots + c_{n-1}x^{n-1}$.

Proof
Follows from the fact that for $\gamma \in \mathbb{F}, \gamma \neq 1$:

$$\sum_{j=0}^{q-2} \gamma^j = 0$$

What does this mean? This means the parity check matrix is:

$$H = \begin{bmatrix} 1 & \alpha & \alpha^2 & \ldots & \alpha^{n-1} \\ 1 & \alpha^2 & \alpha^4 & \ldots & \alpha^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{n-k} & \alpha^{2(n-k)} & \ldots & \alpha^{(n-k)(n-1)} \end{bmatrix}$$

which is also a Vandermonde matrix! This means

$$RS^{\perp}(\mathbb{F}^*, k) = \{(g(1), \alpha g(\alpha), \alpha^2 g(\alpha^2), \ldots, \alpha^{n-1} g(\alpha^{n-1})) \in \mathbb{F}^n \mid deg(g) < n - k\}$$

This is basically an RS code but we have extra multiplications in the polynomial (which actually doesn't change anything!). This is a **generalized Reed-Solomon code**.

Let us return to the BCH code and represent in terms of the generator of $\mathbb{F}_{2^m}$, α.

$$H = \begin{bmatrix} 1 & \alpha & \alpha^2 & \ldots & \alpha^{n-1} \\ 1 & \alpha^3 & \left(\alpha^3\right)^2 & \ldots & \left(\alpha^3\right)^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{2t-1} & \left(\alpha^{2t-1}\right)^2 & \ldots & \left(\alpha^{2t-1}\right)^{n-1} \end{bmatrix}$$

What are the parity checks here, $c(\alpha) = c(\alpha^3) = \cdots = c(\alpha^{2t-1}) = 0$. We can add the even parity checks for free, since in the finite field this is already true. This means the following fact is true:

Theorem 5.2
A BCH code of length $n = 2^m - 1$ and desired distance at least $d = 2t + 1$ is:

$$RS(\mathbb{F}_{2^m}^*, n - d + 1) \cap \mathbb{F}_2^n$$

i.e. they are the subfield subcode of RS (the ones where the codewords are binary rather than 2^m-ary.)

Note the dimension is at least:

$$k \geq n - \frac{d-1}{2} \log_2(n+1)$$

This is desirable because it's a binary code while Reed-Solomon can only function with field elements in a large field. The problem is that the relative distance goes to 0 for large n, i.e.

$$\frac{d}{n} \leq \frac{1}{\log n} \to 0$$

5.2 Binary Codes from R-S Codes

Fix some linear bijection $\phi : \mathbb{F}_{2^m} \to \mathbb{F}_2^m$ (we can do this due to vector space properties). Consider taking some message $f \to (f(a_1), \ldots, f(a_n)) \in \mathbb{F}_{2^m}^n$. Now lets apply ϕ to get:

$$[\phi(f(a_1)) \ldots \phi(f(a_n))] \in \mathbb{F}_2^{nm}$$

This is a linear code.

The block length of this code is $N = nm$. What is its rate? It's still the same $\frac{k}{n}$. Now let's argue the distance. Take two polynomials $f(x), g(x)$ with minimum distance d, they differ at that many points. Suppose $f(a_i) \neq g(a_i)$. How many bits to these differ at? At least 1.

$$\delta \geq \frac{n-k+1}{nm} \approx \frac{1-R}{m} \geq \Omega\left(\frac{1}{\log n}\right)$$

There are some RS codes where no ϕ can save you from this. However, for all RS codes, $\phi_1, \phi_2, \ldots, \phi_n$ can work and can get to GV bound asymptotically. But these are hard to come up with–they have to be randomly sampled.

The solution is to take each of these $\phi(f(a_i))$ and use an "inner code" C_{in} of rate $\frac{1}{2}$ to get a vector in $\mathbb{F}_2^{2m}$. This is called code concatenation or code composition.

Definition 5.1 (Code Composition)
Suppose C_{out} is a code over $\mathbb{F}_{q^m}$ that is a $[n,k]_{\mathbb{F}_{q^m}}$ code and C_{in} is a $[n',m]_{\mathbb{F}_q}$ code. The composing the maps into $C_{out} \diamond C_{in}$ yields a $[nn', km]_{F_q}$ code.

There are d_{out} positions where $c_i \neq c_i'$. At these positions, c_{in} encodes them into vectors at Hamming distance $\geq d_{in}$. This means the distances multiply as well!

$$d(C_{out} \diamond C_{in}) \geq d_{out} \cdot d_{in}$$

Returning to our $\frac{1}{2}$ example. We see this fixes everything.

$$R_{total} = \frac{R}{2}$$

$$\delta \geq (1-R)\delta_{in}$$

If we can have a constant distance, we meet the GV Bound. To do this, we will pick an inner code in exponential time but since the code is of dimension $\log n$, this is OK! Our goal is a $[2m, m, \delta_{in}m]_{\mathbb{F}_2}$ inner code with δ_{in} bounded away from 0 as $m \to \infty$. We can construct this in $2^{O(m)}$ time with $\delta_{in} \approx h^{-1}(\frac{1}{2}) = 0.11$ with the GV bound construction. This is is a $poly(N)$ time construct of an asymptotically good binary code.

$$\delta(R) = (1-2R)h^{-1}\left(\frac{1}{2}\right)$$

To find the best delta we can:

$$\delta_{\text{Zyablov}}(R) = \max_{1 \geq r \geq R} \left(1 - \frac{R}{r}\right) h^{-1}(1-r)$$

This is an explicit polytime construction! Obviously this is not as great as the GV Bound, which needs exponential time to construct, but still pretty good.

If $\delta = \frac{1}{2} - \epsilon$ then Zyablov is ϵ^3, GV is ϵ^2 and a recent paper found an explicit construction of $\epsilon^{2+o(1)}$.

To clean this up consider the inner code that expands to:

$$f(a_1)a_1f(a_1)f(a_2)a_2f(a_2)\dots f(a_n)a_nf(a_n)$$

and then apply ϕ afterwards, i.e. to get

$$\phi(f(a_1))\phi(a_1f(a_1))\phi(f(a_2))\phi(a_2f(a_2))\dots\phi(f(a_n))\phi(a_nf(a_n))$$

this code can also get that bound.

6 Lecture 6

6.1 Justesen Codes and Friends

We said last time that the Justesen encoding:

$$\phi(f(a_1))\phi(a_1 f(a_1))\phi(f(a_2))\phi(a_2 f(a_2))\dots\phi(f(a_n))\phi(a_n f(a_n))$$

also gives our nice Zyablov bound:

$$\delta(R) \geq (1-2R)h^{-1}\left(\frac{1}{2}\right) - o(1)$$

Here, we think of the inner code for each a_i:

$$E(C_\alpha) : x \mapsto (x, \alpha x)$$

which is a map $\mathbb{F}_{2^m} \to \mathbb{F}_{2^m} \times \mathbb{F}_{2^m}$. The reason this works is the following fact.

Theorem 6.1 (Wozencraft Ensemble)
For must (all but $o(1)$ values of $\alpha \in \mathbb{F}^*$), c_α is close to the GV bound.

$$\forall \epsilon > 0, P_\alpha\left[\delta(C_\alpha) > h^{-1}\left(\frac{1}{2}\right) - \epsilon\right] \to 1$$

Proof 6.1
We first claim that for each $z \in \mathbb{F}_2^{2m}$ s.t. $z \neq 0$ is in at most one C_α. For a good C_α, we want the weight of a z to be $\leq \left(h^{-1}(1/2) - \epsilon\right)2m = \rho \cdot 2m$. Then the amount of bad C_α's is at most the amount of these z's, i.e.:

$$|\text{ball}| = 2^{h(\rho)2m} = 2^{(1/2-\epsilon)2m} = 2^{(1-2\epsilon)m}$$

You can consider this Ensemble as a "derandomization" of a random linear code. We only need m random bits (the αs), not $O(m^2)$ for the randomized matrix case.

Note that the rate we get is limited to be at most $\frac{1}{2}$ because of the inner code. To fix this, we can use an inner code that is less damaging to the rate:

$$C_\alpha^{\text{trunc}} : x \mapsto (x, (\alpha x)_{\text{first } s \text{ bits}})$$

This has rate: $\frac{m}{m+s}$, so we can pick s to be whatever we want.

Now our bound is:

$$\delta_{Justesen}(R) = \max_{\max\{R, \frac{1}{2}\} \leq r \leq 1} \left(1 - \frac{R}{r}\right) h^{-1}(1-r)$$

For $R \geq 0.31$, optimizing r is bigger than $\frac{1}{2}$, so the bound applies the same as Zyablov. For lower rates R, you have choose some other code like:

$$C_{\alpha,\beta} : x \mapsto (x, \alpha x, \beta x)$$

and playing this game over and over fits the Zyablov bound.

6.2 Reed-Solomon Decoding

Remember that $RS_{\mathbb{F}}(S, k)$ has distance $n - k + 1$, which hits the Singleton bound. We will call the polynomial it generates $f(x)$

Erasure Decoding. If up to $s = n - k$ erasures, then we can interpolate the unerased positions (there are k points so we can recover the polynomial). Note for any generator matrix of a linear code with dimension k, we can just recover erasures by solving a linear system which is nicely poly-time anyways.

Error Decoding. We know we can correct up to $e = \left\lfloor \frac{d-1}{2} \right\rfloor = \left\lfloor \frac{n-k}{2} \right\rfloor$ errors, we don't know what's right and what's wrong. Perhaps our output looks like (with bolded things being errors).

$$y_1 y_2 \mathbf{y_3} y_4 \mathbf{y_5} \cdots \mathbf{y_{n-1}} y_n$$

This means $f(x)$ is uniquely determined by any (corrupted) output. How can we find $f(x)$ efficiently?

Well, the brute force algorithm is considering each subset of e errors and interpolating the rest. An efficient algorithm is the Welch-Berlekamp algorithm. Let $F = \{i \mid f(a_i) \neq y_i\}$. If we know F, we are done. The idea is we want to work with an algebraic form of F. We call this the error locator polynomial:

$$E(x) = \prod_{i \in F} (x - a_i)$$

Observe that for all a_i, either $y_i - f(a_i) = 0$ or $E(a_i) = 0$. Then we know:

$$(y_i - f(a_i))E(a_i) = 0$$

We define the function:

$$P(x, y) = (y - f(x))E(x) = E(x)y - E(x)f(x)$$

so that:

$$P(a_i, y_i) = 0$$

The idea is that we will interpolate this curve with a polynomial that is the right degree, and this will recover exactly f. Let's define $N(x) = E(x)f(x)$. Then:

$$P(x, y) = E(x)y - N(x)$$

where we insist that $\deg(E) \leq e$ and $\deg(N) \leq e + k - 1$. Now we can state the algorithm.

1. Find a nonzero polynomial $Q(x, y)$ such that:

   (i) $Q(a_i, y_i) = 0, \forall i$

   (ii) $Q(x, y) = E_1(x)y - N_1(x)$ with the appropriate degree restrictions.

   which amounts to solving a linear system.

2. Output $f(x) = \frac{N_1(x)}{E_1(x)}$.

The following claims are necessary for correctness:

1. A nonzero Q sought after in step 1 exists. Take $Q = P$, such a polynomial does exist by explicity construction.

2. Any such Q found in step 1 must satisfy $\frac{N_1(x)}{E_1(x)} = f(x)$. Well, we know that

   $$Q(x, f(x)) = R(x) = f(x)E_1(x) - N_1(x)$$

   and if we show that $R(x) = 0$, then we are done. Note that degree of R is at most $e + k - 1$. But, it is 0 for all $n - e$ of the correct points! By the amount of errors, this is more than the degree. So it must be the 0 polynomial.

So, the Reed-Solomon code of rate R can be efficiently decoded to $\frac{e}{n} = \frac{1-R}{2}$ error fraction. In fact, it can be shown that an RS code can be coded from any combination of e errors and s erasures as long as: $2e + s \leq n - k$.

7 Lecture 7

7.1 Decoding Concatenated Codes

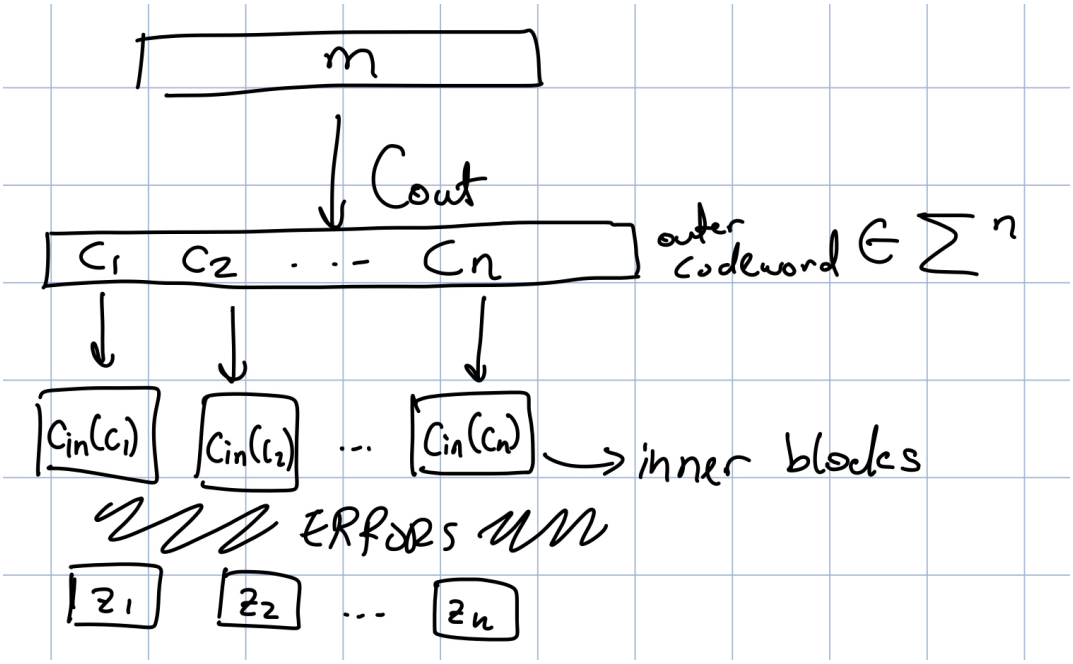
Now how can we decode a concatenated code $C_{out} \diamond Cin$ where the distance of C_{out} is D and that of C_{in} is d. Then $dist(C_{out} \diamond C_{in}) \geq Dd$. Let's try to correct $e < \frac{Dd}{2}$ errors efficiently. This would mean: the error fraction

$$\tau = \frac{\delta_{out}\delta_{in}}{2}$$

which is bounded away from 0 as long as the distances are. A corollary of this is an explicit, asymptotically good binary linear code family that can efficiently correct τ errors with τ bounded away from 0.

$$\tau(R) = \frac{\delta_{Zyablov}(R)}{2}$$

How can we decode? Let's revise what encoding was:



Our assumption is:

$$\sum_{i=1}^{n} \Delta(z_i, C_{in}(c_i)) \leq e$$

The natural approach is to first decode the inner blocks and then decode the outer code.

1. Decode each z_i to $a_i \in \Sigma$ such that $\Delta(z_i, C_{in})$ is minimized (breaking ties arbitrarily). If you have a more efficient encoding scheme, this suffices as well.
2. Decode $\langle a_1, a_2, \ldots, a_n \rangle$ per outer decoder, up to decoding radius $D/2$.

Runtime is efficient as long as the inner code is small, i.e. $\Sigma \leq poly(n)$. Define $N = nn'$.

$$T(\text{outer decoder}) + n \cdot poly(|\Sigma|, n') \leq poly(N)$$

Analysis: If $|\{i \mid a_i \neq c_i\}| < \frac{D}{2}$ the algorithm suceeds. This means that if the algo fails, then we must have $|\{i \mid a_i \neq c_i\}| \geq \frac{D}{2}$ and $a_i \neq c_i$ implies $\Delta(z_i, C_{in}(c_i)) \geq \frac{d}{2}$. Which means that $|\{i \mid a_i \neq c_i\}| \geq \frac{Dd}{4}$. So even this simple algorithm gets within a constant factor of our desired error fraction.

Let's try a different approach. Let's have the inner decode give more guidance to outer decoder as "soft information."

1. Decode each z_i to $a_i \in \Sigma$ such that $\Delta(z_i, C_{in})$ is minimized (breaking ties arbitrarily). Set $w_i = \min\left\{\Delta(z_i, C_{in}(a_i)), \frac{d}{2}\right\}$.
2. For each i, with probability $\frac{2w_i}{d}$, erase a_i. Decode $\langle a_1, a_2, \ldots, a_n\rangle$ per outer decoder.

Theorem 7.1
If $\sum_{i=1}^{n} \Delta(z_i, C_{in}(C_i)) < \frac{Dd}{2}$, then

$$\mathbb{E}\left[2Z_{errors} + Z_{erasures}\right] < D$$

Where the Z is the amount of errors/erasures in the outer code decoding.

Proof
Write some indicators: $Z_{errors} = \sum Z^i_{errors}$ and $Z_{erasures} = \sum Z^i_{erasures}$. If $a_i = c_i$, then $Z^i_{errors} = 0$ and $\mathbb{E}\left[Z^i_{erasures}\right] = \frac{2w_i}{d} = \frac{2e_i}{d}$. In the other case, $\mathbb{E}\left[Z^i_{erasures}\right] = 2w_i/d$ and the other one is the complement.

$$\begin{aligned}\mathbb{E}\left[2Z^i_{errors} + Z^i_{erasures}\right] &= 2 - \frac{2w_i}{d} \\ &\leq 2 - \frac{2(d - e_i)}{d} \\ &\leq \frac{2e_i}{d}\end{aligned}$$

where the second line comes from the fact that $w_i + e_i \geq d$. In both cases,

$$\mathbb{E}\left[2Z^i_{errors} + Z^i_{erasures}\right] \leq \frac{2e_i}{d}$$

which shows the claim, since $e = \sum e_i < \frac{Dd}{2}$.

In reality, this analysis is equivalent to picking a threshold $\theta \in [0, 1]$ uniformly. If $\theta \leq \frac{2w_i}{d}$, we declare an erasure at i. By this probaiblistic method, there must exist some θ^* that makes the algorithm actually work. But, note that there are only changes in the algorithm's threshold when θ goes between the w_i. Trying these $O(d)$ thresholds will get you all possibilies that matter (there are only $O(d)$ w_i's).

This is called the Generalized Minimum Distance (GMD) decoding.

7.2 Concatenation: "Serial" vs. "Parallel"

We can think of the inner codes as local codes C_0. With $r \ll n$ where $r = O(1)$ or $r = O(\log n)$.

What if we do the following instead:

We specify a set $\mathcal{F} \subseteq \binom{[n]}{r}$ (choose a subset of size r under some fixed global ordering) such that for all $S \in \mathcal{F}$, we have $c_S \in C_0$ (where c is projected onto the coordinates in S).

Suppose we want each i to belong to two local checks. Here is one such code, a product code:

$n = r^2$. Then if C_0 is a $[r, s, d]_q$ code, then C_0^2 is $[r^2, s^2, d^2]_q$ code. But, rate and δ both square tho (which means they go down...). The issue with this tensor product representation is I cannot take a local r code and build a code of arbitrary size n.