1 Recap on Concatenated Codes and the Zyablov Tradeoff

Last lecture we examined a technique to construct a binary version of Reed-Solomon codes utilizing a concatenated code construction, and we looked at the Zyablov tradeoff for binary linear codes. For q-ary codes, where q is of the form 2^m, we can exploit the isomorphism $\varphi : \mathbb{F}_{2^m} \to \mathbb{F}_2^m$, in order to turn our blocklength n codewords in $\mathbb{F}_q$ into binary codewords in $\mathbb{F}_2^m$ (for general q-ary codes, each of these smaller blocks is of length $\log_2(q)$). When doing this however, the distance of our new code is worse than our original code:

$$d \geq \frac{n-k+1}{nm} \approx \frac{1-R}{m}$$

where m is $\Omega\left(\frac{1}{\log n}\right)$ and R is the rate of our original code. The solution to this issue was to concatenate our "outer" code with "inner" codes of rate $\frac{1}{2}$, each of blocklength m.

$$f(x)$$
$$\downarrow$$
$$(f(a_1), ..., f(a_n))$$
$$\downarrow$$
$$(\varphi(f(a_1)), ..., \varphi(f(a_n)))$$
$$\downarrow C_{\text{in}}$$
$$(C_{\text{in}}(\varphi(f(a_1))), ..., C_{\text{in}}(\varphi(f(a_n))))$$

By this construction, the distance of our resulting code is at least the product of the distances of our chosen outer code and inner codes and the distance is exactly the product of the dimension of the respective codes:

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

With this gain in distance, we naturally see a drop off in the rate of our new code:

$$\text{Rate} \ = \ \frac{km}{2nm} = \frac{R}{2}$$

Of particular interest is our choice(s) of inner code C_{in}. The issue appears to be circular when we're trying to give explicit descriptions of asymptotically good (i.e. their rate and relative distance are bounded away from zero) binary linear codes (our outer code), and yet require the same (our inner code) when constructing them. So how did we reconcile this? We did so with the Zyablov tradeoff, which gives us, not only the assurance of such good codes existing, but tells us that such good codes can be constructed in poly(n) time. The tradeoff is the following:

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

An alternative way to think about this is that we don't know how to directly find a code that meets the GV Bound. However, we can get a small collection of codes where most of them come close to it.

What was left on the table was how to actually get these explicit descriptions of asymptotically good binary linear codes.

2 Justesen Codes

Justesen [1] proposed a key improvement on the above "naive" construction. Why just choose one kind of inner code to encode each size $\log_q(n)$ block? Why not choose different codes C_α in order to encode each block in the following way:

$$C_\alpha : \mathbb{F}_{2^m} \mapsto \mathbb{F}_{2^m} \times \mathbb{F}_{2^m}$$

given by

$$x \mapsto (x, \alpha x)$$

Justesen codes [1] give us the very explicit construction we desire. Suppose we have our degree $k-1$ polynomial $f(x)$, and evaluation points $S = a_1, .., a_n$, where $S = \mathbb{F}_{2^m}^*$. Then as opposed to simply defining our codewords to be the evaluations of our polynomial at these n points, we define our encoding in the following way:

$$f(x)$$

$$\downarrow$$

$$(f(a_1), a_1 f(a_1), f(a_2), a_2 f(a_2), ..., f(a_n), a_n f(a_n))$$

Using this construction, we get a code of rate $R = \frac{k}{2n}$. With regards to the above construction, we make the following claim with respect to its relative distance:

Claim 6.1. *We get an asymptotically good code construction such that*

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

Having expanded further on Justesen codes, the problem boils down showing that almost all C_α are asymptotically good codes. That is, all but $o(1)$ of such C_α are asymptotically good codes.

2.1 Wozencraft Ensemble

Let us make the following claim

Claim 6.2. *For most (all but $o(1)$) α, C_α approaches the Gilbert-Varshamov Bound. That is, for $\alpha \in \mathbb{F}^* = \mathbb{F}_{2^m}^*$*

$$\mathbf{Pr}\left[\delta(C_\alpha) \geq h^{-1}\left(\frac{1}{2}\right) - \epsilon\right] \to 1$$

as $m \to \infty$ for all $\epsilon > 0$.

This claim tells us that there exists some good ensemble of good codes C_α for choices of $\alpha \in \mathbb{F}^*$ such that if you randomly sample from the ensemble, you can, with high probability find codes that meet the GV bound (this is indicated by the inverse of the binary entropy function). Additionally, the Claim 6.2 leads to Claim 6.1 regarding the asymptotic behavior of the Justesen code

construction's relative distance measure, and that's because if any Reed-Solomon codeword had weight w, then its resulting concatenation by the code ensemble C_α will have a resulting weight of $w2^{m+1}\left(h^{-1}\left(\frac{1}{2}\right) - o(1)\right)$.

The code ensemble C_α is known as the ***Wozencraft Ensemble (WE)***. This WE also has connections to derandomization of random linear codes. Why derandomization? $\rightarrow$

- The number of linear codes at the inner level of our Justesen code construction is 2^m as opposed to 2^{m^2}, and that's by randomly fill out a generator matrix for this code of size $2m \times m$.
- We also desire a small collection of codes, where this collection is of size 2^{m-1}, such that most of these codes are asymptotically good.

So why does this Wozencraft Ensemble give you a small collection of codes that are close to meeting the GV bound? How do we know that $\mathbf{Pr}[\delta(C_\alpha) \geq h^{-}1(1/2) - \epsilon] \rightarrow 1$ as m goes to ∞? We make the following claim:

Claim 6.3. *Each $z \in \mathbb{F}_2^{2m}$, $z \neq 0$ is in at most one C_α.*

That is, for a choice of $z = (z_1, z_2) \in (\mathbb{F}_{2^m})^2$, $z_1 \neq 0$, we have that such a z lives in exactly one code C_α, and that is $\alpha = z_2/z_1$. Otherwise, if $z_1 = 0$ but $z_2 \neq 0$, then it belongs to no C_α:

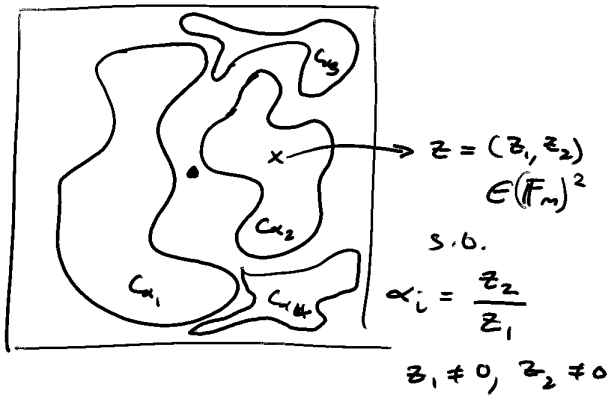

Figure 6.1: When $z_1 \neq 0$, each nonzero field element z lives in exactly one code C_α (no need for $z_2 \neq 0$).

Thus, if we imagine the box above to be all elements of $(\mathbb{F}_{2^m})^2$, then we'd like to pack the above space with codes C_α so that all of them are disjoint from each other $\implies$ that a choice of $z \in \mathbb{F}_{2^m} \times \mathbb{F}_{2^m}$ lives in exactly one code C_α.

In general, however, we'd like to be careful in choosing our α's and don't want to choose codes from the above space willy-nilly, as we are still looking for asymptotically good codes. Thus, there is a region of our space that we forbid choosing α's from:

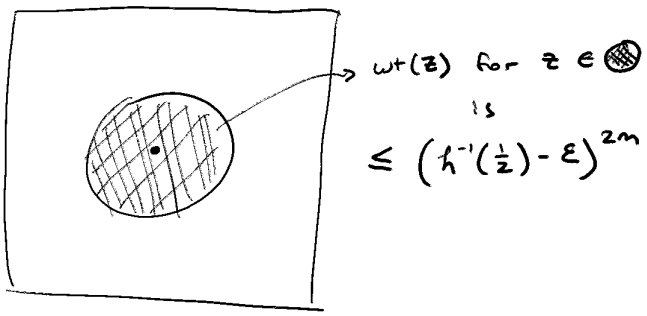

Figure 6.2: "Bad" region from which we'd like to completely avoid choosing any α's from.

Let us quantify the number of α that are "off-limits". Given that the codeword weight for elements in this bad region are at most $(h^{-1}(1/2)-\epsilon)\cdot 2m$ we have that there are roughly $2^{\left(\frac{1}{2}-\epsilon\right)2m} = 2^{(1-2\epsilon)m}$. Thus, all but an exponentially small set will yield good C_α.

2.2 Critiques of Justesen Construction

Here we discuss some of the potential issues with the Justesen construction and their related fixes.

2.2.1 Concatenated Codes with Higher Rates

One of the first potential issues we may have with the Justesen construction as it is defined is that it's rate is limited to less than $1/2$. This is due to the fact that the inner codes we choose have rate at most $1/2$. A reasonable fix for this is to go beyond a rate of $1/2$, and therefore this would require us to define a different ensemble such that randomly sampling from it yields codes of our desired rate. To do so, let us define the following "truncated code" C_α^{trunc}:

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

giving us a rate of

$$R = \frac{m}{m+s}$$

Then we can prove a similar claim for the above family regarding the GV Bound:

Exercise 6.4. *Prove most of C_α^{trunc} meets the GV bound.*

That is, we can prove

$$\mathbf{Pr}\left[\delta(C_\alpha^{\text{trunc}}) > h^{-1}\left(\frac{s}{m+s}\right) - \epsilon\right] \to 1$$

as $m \to \infty$ and for all $\epsilon > 0$. With these constructions, we know we're trying to chase the Zyabolov tradeoff, which tells us the best possible rate-distance tradeoffs we can get for concatenated binary linear codes. The Justesen construction tells us that we can meet this tradeoff as

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

Differentiating the equation above with respect to R would allow us to find the R that maximizes the distance for a chosen rate r. If we have R big enough, then it is prudent enough to select $r \ge \frac{1}{2}$. In fact, more specifically, for $R \ge 0.31$ and optimizing $r \ge \frac{1}{2}$, we get that $\delta_{\text{Justesen}} = \delta_{\text{Zyablov}}$.

2.2.2 Concatenated Codes with Lower Rates

We see that for higher rate concatenated codes, we're able to achieve the Zyablov bound with the Justesen construction. What about for lower rates? In this case, a little more sweat is needed to seek out maximal distances, but we can imagine a similar setup as compared to higher rate codes. We can define another ensemble of codes defined

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

If $x \in \mathbb{F}_{2^m}$, then $C_{\alpha,\beta}(x)$ lives in $(\mathbb{F}_{2^m})^3$, thus giving us a rate $\frac{1}{3}$ code. Much like the Wozencraft ensemble from earlier, we can make similar claims with regards to goodness of codes that are randomly sampled from this ensemble. More concretely, we can say that for most α, β, $C_{\alpha,\beta}$ has a relative distance close to the GV Bound: $\delta \to h^{-1}\left(\frac{2}{3}\right)$. In general then, for lower rate concatenated codes, we simply need to choose an ensemble that satisfies our needs. That is, if we desire a small ensemble of codes with rate r_l that are close to $h^{-1}(1 - r_l)$, we can craft the form of our ensemble according to this desired rate.

2.2.3 Something is Still Wrong - Too Many Codes

Despite having chosen a small enough ensemble to satisfy our choices for the inner code, there are still too many codes at this level, for the outer level of our concatenated code construction to be compatible with. That is to say, the Reed-Solomon code at the outer level isn't long enough! Therefore, as a fix to this, we can take longer *Reed-Solomon-esque codes* while utilizing the same inner codes from before. As opposed to meeting the Singleton bound, these Reed-Solomon-esque codes will approach the Singleton bound ($\delta_{\text{out}} \to 1 - r_{\text{out}}$) Furthermore, since the Reed-Solomon-esque construction is based off of more generalized algebraic geometry codes, we can imagine choosing more general algebraic curves as our function, and choosing points along these curves. Thus instead of having polynomials in one variable, like we we had in the Reed-Solomon construction, we can consider more general polynomials in two or more variables over some algebraic curve.

Remark 6.5 (***Challenge/Open Question***)**.** For all rates (Ex: rate = $\frac{1}{3}$), can you give an ensemble of size 2^k of $[3k, k]_2$ codes such that when randomly sampled from, with high probability you select codes that approach the GV Bound?

Taking a bird's eye view, we started from the Reed-Solomon construction, which gave us optimal codes meeting the Singleton bound. However, we were forced to work with larger finite fields given the condition that $q \geq n$ (where q is the order of the finite field, and n is the blocklength of our code). We not only desired asymptotically good linear codes, but desired that these codes also be binary. Thus, we proposed the Justesen code construction. These concatenated code constructions admitted an optimal rate-distance tradeoff, called the Zyablov tradeoff. Thus, the Justesen construction has given us what we were initially searching for: *an explicit description of asymptotically good binary linear codes that meet the Zyablov Bound.*

3 Reed-Solomon Decoding

Let us reiterate the Reed-Solomon code construction for clarity: An $RS_{\mathbb{F}}(S, k)$ code maps a degree at most $k - 1$ polynomial $f(x)$ to a vector that is the evaluation of $f(x)$ on all points in the set S:

$$f(x) \mapsto (f(a_1), ..., f(a_n)) \text{ for } a_i \in S$$

As we saw before, this $RS_{\mathbb{F}}(S, k)$ code meets the Singleton Bound. Let us now take a look at how to perform erasure and error decoding on Reed-Solomon Codes.

3.1 Erasure Decoding

Suppose we have the following codeword:

$$\boxed{f(a_1)\ f(a_2)\ f(a_3)\ \ldots\ f(a_n)}$$

and we erase some of these positions:

$$\boxed{?}\ f(a_2)\ \boxed{?}\ \boxed{?}\ f(a_5)\ \ldots\ f(a_n)$$

Thus, we can handle at most $n-k$ erasures, as we can simply use polynomial interpolation on the remaining k positions in order to recover our polynomial. Erasing codeword positions is also equivalent to erasing rows of our generator matrix, which in the case of Reed-Solomon codes is an $n \times k$ Vandermonde matrix. Removing at most $n-k$ of the rows of this Vandermonde matrix leaves us with an invertible $k \times k$ submatrix, which we can then use to fully determine our original message. Such a process can be done in near-linear time using a Fast Fourier Transform-like algorithm, which will take $\mathcal{O}(n \log n)$ time.

3.2 Correcting Errors

Now suppose we are dealing with correcting errors on codewords in the Reed-Solomon construction. Unlike erasure errors, the positions of these errors are unknown, so how do we address them? Let's begin by characterizing how many errors our code construction allows us to correct (Assume $f(a_i) = y_i$ under no errors):

$$\boxed{f(a_1)\ f(a_2)\ f(a_3)\ \ldots\ f(a_n)}$$

$$\downarrow\ e \text{ errors}$$

$$\boxed{y_1}\ y_2\ \boxed{y_3}\ \cdots\ \boxed{y_n}$$

Since the number of errors we can tolerate in general is

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

We have that e can be at most

$$\left\lfloor \frac{n-k}{2} \right\rfloor$$

Suppose we're given the promise that $|\{i : f(a_i) \neq y_i\}| \leq e$. Then $f(x)$ can be uniquely determined – the challenge remains in efficiently finding out what our polynomial $f(x)$ is. As we mentioned before, the primary issue is our inability to know for sure where errors have occurred. A naive solution would be to look at all possible positions where errors could occur, namely $\binom{n}{e}$ positions. However, this search would most certainly admit a very inefficient decoding scheme. Thus, we somehow need to exploit the structure of our code in order to find errors.

3.2.1 Welch-Berlekamp Algorithm for Decoding Reed-Solomon Codes

Let us define our error locations to be the set of codeword positions F. That is

$$F = \{i : f(a_i) \neq y_i\}, \quad |F| \leq e = \left\lfloor \frac{n-k}{2} \right\rfloor$$

If we know the identity of F, then we're done, as our problem essentially reduces down to the problem of erasure decoding. Ignoring this trivial case, to address error-correction in general, we must work with an algebraic form of the set F. We construct an *error-locator polynomial* of the following form:

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

Again, it is important to note that we can't know E as this would imply we know the identity of F. Thus, we make the following observations $\forall a_i \in S$ (recall S is the set of evaluation points initially used to define our $RS_{\mathbb{F}}(S, k)$ code):

1. If $a_i \notin F$, then $y_i - f(a_i) = 0$.
2. If $a_i \in F$, then $E(a_i) = 0$.

Following this observation, we deduce the equation $E(a_i)(y_i - f(a_i)) = 0$ for all $a_i \in S$ We use this as a handle to guide the development of the error-decoding algorithm.

Definition 6.6 (Restatement of Above Observation)**.** Define the polynomial

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

- **Note:** We still do not know the identity of F or E
- **Note:** For all i, $P(a_i, y_i) = 0$

To provide some motivation for the above definition, consider the following *noisy-curve fitting view*:

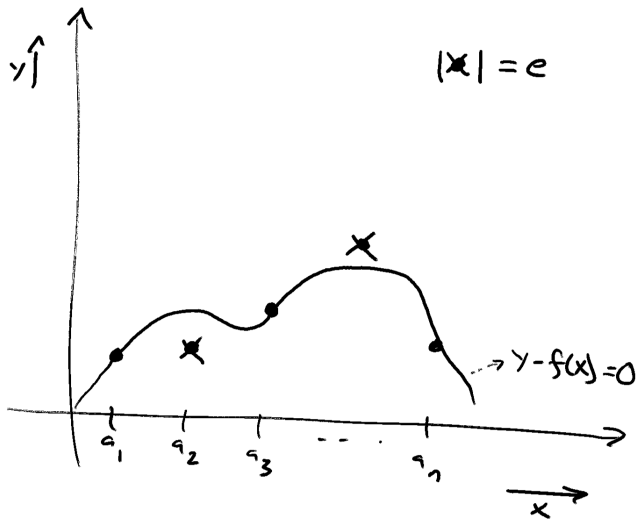

Figure 6.3: Points off of the curve represent erroneous codeword positions. We assume the presence of e erroneous codeword positions.

However, the way we've defined $P(x, y)$ above, this curve should pass through all points including points corresponding to erroneous codeword positions. This gives us the following picture:

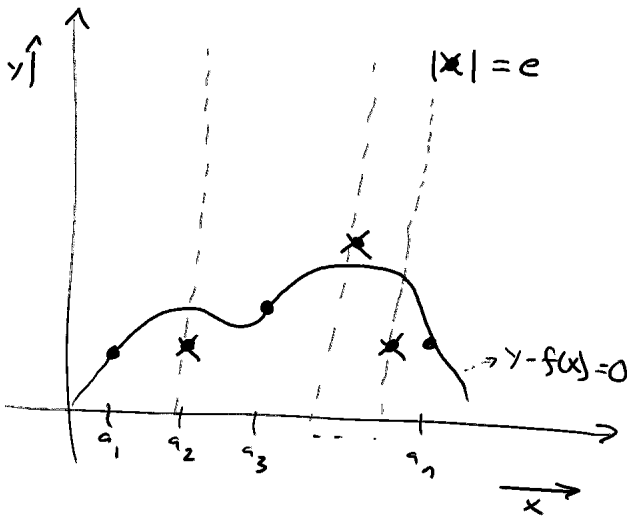

Figure 6.4: To fit all points with the curve $P(x, y)$, the curve through all correct points and the vertical lines are also part of the curve that pass through erroneous points.

To make $P(x, y)$ easier to work with, let's consider use following *"linearization trick"*: We know that $P(x, y)$ is linear in y, thus we can redefine it in the following way:

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

such that the degree of $E(x)y$ is $\leq e$ and the degree of $N(x)$ is $\leq e + k - 1$. With this, we can complete the description of our algorithm, which comes in **2 main steps**.

1. Find a **non-zero** polynomial $Q(x, y)$ such that

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

   (b) $Q(x, y) = E_1(x)y - N_1(x)$ where $\deg(E_1) \leq e$ and $\deg(N_1) \leq e + k - 1$

   This particular step can be done in polynomial time, as we are simply solving a homogeneous linear system to solve for E_1 and N_1. This is equivalent to solving for the coefficients for E_1 and N_1.

2. **Output**

   $$\frac{N_1(x)}{E_1(x)}$$

   as our answer for $f(x)$. We know that dividing polynomials is efficient:

   (a) Quadratic time

   (b) Near-linear time using FFT

Let us now speak to the *correctness of our algorithm.*

Proof. (a) Claim #1: A non-zero $Q(x, y)$ sought after in step 1, exists.

(b) Claim #2: Any such $Q(x, y)$ fond in step 1 must satisfy

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

Proving Claim #1 and Claim #2 gives us the correctness of our algorithm. Therefore,

- *Proof of Claim #1:* Take $Q = P(x, y)$ defined prior such that $Q(x, y) = (y - f(x))E(x)$
- *Proof of Claim #2:* We want to prove that $Q(x, y) = E_1(x)(y - f(x))$ Let $y = f(x)$. Then we'd like to equivalently prove that $E_1(x)f(x) - N_1(x) = 0$. Let us define a function $R(x)$ and prove that $R(x) \equiv 0$.

  Definition 6.7. $R(x) \coloneqq P(x, f(x)) = E_1(x)f(x) - N_1(x)$

  If $f(a_i) = y_i$, then we have that

  $$\begin{aligned} R(a_i) &= f(a_i)E_1(a_i) - N_1(a_i) \\ &= y_i E_1(a_i) - N_1(a_i) \\ &= 0 \end{aligned}$$

  Therefore, $R(x)$ has at least $n - e$ roots. On the other hand, the degree of $R(x)$ is at most $e + k - 1$. Thus if $n - e > e + k - 1$, then we get the identity $R(x) \equiv 0$ (i.e. R is the zero polynomial) and thus finishes the claim. Notice that $n - e > e + k - 1$ is equivalent to $e < \frac{n-k+1}{2}$, which is true by definition of e. Therefore, our chosen $Q(x, y)$ does equal $E_1(x)(y - f(x))$.

Thus, we've proved the following theorem:

Theorem 6.8. *Reed-Solomon codes of rate R can be efficiently decoded to $\frac{1-R}{2}$ error fraction.*

□

References

[1] Justesen, J. Class of constructive asymptotically good algebraic codes. *IEEE Transactions on Information Theory 18*, 5 (1972), 652–656. 2