Lecture 03: Polynomial Based Codes

Error-Correcting Codes (Spring 2016)
Rutgers University
Swastik Kopparty
Scribes: Ross Berkowitz & Amey Bhangale

1 Reed-Solomon Codes

Reed Solomon codes are large alphabet codes. They are not binary, but will eventually be helpful for constructing binary codes too.

1.1 Construction

- Fix $\mathbb{F}_q$ a finite field with q elements (if q is prime, then $\mathbb{F}_q = \mathbb{Z}/q\mathbb{Z}$).
- Fix $S \subset \mathbb{F}_q$, and let $n := |S|$. Assume that $S = \{\alpha_1, \alpha_2, \ldots, \alpha_n\}$.
- k will be a parameter.

The Reed-Solomon code will be $\mathcal{C} \subset \mathbb{F}_q^n$ given by

$$\mathcal{C} := \left\{ \big(f(\alpha_1),\ f(\alpha_2), \ldots, f(\alpha_n)\big) \mid f(x) \in \mathbb{F}_q[x], \quad \deg(f) < k \right\}$$

1.2 Properties

1. $|\mathcal{C}| \leq q^k$, the number of polynomials of degree less than k (there are q^k ways to set coefficients of a degree $k-1$ polynomial).
2. $\mathcal{C}$ is $\mathbb{F}_q$-linear and $\dim(\mathcal{C}) \leq k$.
3. If $d(\mathcal{C}) = d$ then any two polynomials of degree $< k$ agree on at most d coordinates. Furthermore because $\mathcal{C}$ is linear

$$d(\mathcal{C}) = \min_{\substack{C \in \mathcal{C} \\ C \neq 0}} wt(C) = \min_{\substack{C \in \mathcal{C} \\ C \neq 0}} |\{\alpha \in S \mid f(\alpha) \neq 0\}| = n - \max_{\substack{C \in \mathcal{C} \\ C \neq 0}} |\{\alpha \in S \mid f(\alpha) = 0\}| \geq n - k + 1$$

The last inequality above holds because the number of roots of a nonzero degree $< k$ polynomial is at most $k-1$. Also note that number 3 above implies that if $k \leq n$ then $|C| = q^k$.

To summarize,

- Alphabet size q

- Length n
- Dimension k
- Distance $n-k+1$

With the the parameters q, n and k, where q is a prime power and $k \leq n \leq q$. This is a big restriction on n, as over $\mathbb{F}_2$ we only get codes of length 2. Reed-Solomon codes have limited length over any fixed alphabet size. This looks useless as it gives codes with growing length and growing alphabet size. But once you open your mind to codes where the alphabet size keeps growing too, that is a huge step in the understanding of coding.

A Reed-Solomon code has rate

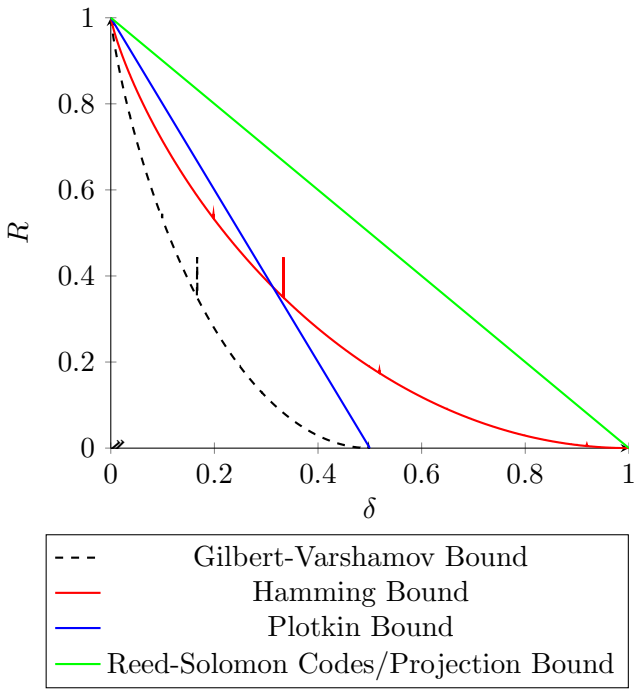
$$R = \frac{\log_{|\Sigma|} |\mathcal{C}|}{n} = \frac{\log_q(q^k)}{n} = \frac{k}{n} = \frac{\dim(\mathcal{C})}{n}$$

and relative distance

$$\delta = \frac{n-k}{n} = 1 - \frac{k}{n} + \frac{1}{n}$$

Asymptotically this gives us $\delta = 1 - R$. We are above the Plotkin bound. This is possible beacause what we proved applied only to binary alphabet codes. Ternary codes would, for example, have a linear bound of $R \leq 1 - \frac{3\delta}{2} + o(1)$. It would be a good exercise to prove the following generalization:

Fact 1 (Plotkin Bound)**.** *If $\mathcal{C}$ is a code with alphabet size $|\Sigma| = q$. Then $R \leq 1 - \frac{q}{q-1}\delta + o_n(1)$.*



1.3 Projection Bound

Reed-Solomon codes are best possible in light of the following bound.

Theorem 2. *Let $\mathcal{C}$ be a code of any alphabet size. Then $\mathcal{C}$ has rate $R \leq 1 - \delta + o(1)$.*

First a quick sketch of the proof. Erase or delete any δ fraction of a codeword. Each codeword should still be recoverable from the remaining $(1-\delta)n$ remaining bits. Therefore all shortened codewords are distinct, and therefore contain at most $(1-\delta)n$ bits of information.

Proof. Delete last $\delta n - 1$ coordinates of all codewords in $\mathcal{C}$. All remaining codewords are distinct (as otherwise two identified codewords would have had distance $\delta n - 1 < \delta n$). Further all remaining shortened codewords lie in $\Sigma^{(1-\delta)n+1}$. Therefore $|\mathcal{C}| \leq |\Sigma|^{(1-\delta)n+1}$. So we directly have that the rate is $R \leq 1 - \delta + \frac{1}{n}$. □

Being open to new definitions is a really useful. Often if we are interested in object A, it is useful to instead weaken our definitions and to search for A', then use our knowledge of A' to help us find A. So too will these growing alphabet codes help us with fixed alphabet codes.

Also it is hard to imagine finding the binary codes we will construct without having passed through finite fields. So we should be thankful to the old mathematicians!

2 Multivariate Polynomial Codes

Our plan is to do the following with the rest of class:

1. Multivariate Codes
2. Decoding Reed-Solomon Codes

2.1 Basic Multivariate Polynomial Code

We will use the following notation in this section:

- m - number of variables
- D - Degree
- $S \subset \mathbb{F}_q^n$

We will define our code to be

$$\mathcal{C} := \{(f(\alpha)_{\alpha \in S} \mid f(x_1, \ldots, x_m) \in \mathbb{F}_q[x_1, \ldots, x_m], \ \deg(f) \leq D\}$$

This is the most general multivariate polynomial. Its distance, among other properties, depends heavily on the choice of S, unlike the Reed-Solomon code. Today we will focus on the case where S is a product set $S = T^m \subset \mathbb{F}_q^n$. In the future though, we will also consider S being the set of points of an algebraic curve. These will be the so called "Algebraic Geometry Codes" and they have many amazing properties.

We will prove a lemma that will show us that any two multivariate polynomials have distance related to their degree.

2.2 Schwarz-Zippel Lemma

The attribution of the Schwarz-Zippel Lemma is long and complicated. We will call it the product set zero or SZ lemma.

Lemma 3 (Schwarz-Zippel Lemma)**.** *Let $f(x_1, \ldots, x_m) \in \mathbb{F}_q[x_1, \ldots, x_m]$ nonzero with $\deg(f) \leq D$. Then*

$$\Pr_{b \in T^m}[f(b) = 0] \leq \frac{D}{|T|}$$

That is, the number of $b \in T^m$ such that $f(b) = 0$ is at most $D|T|^{m-1}$.

Note this generalizes the usual univariate version, which states that a nonzero polynomial of degree D has at most D roots.

Proof. We proceed by induction on m. Let $f(x_1, \ldots, x_m) = \sum_{i=0}^{u} a_i(x_1, \ldots, x_{m-1}) x_m^i$. Where the a_i is an $(m-1) - variate$ polynomial, and we require that a_u is nonzero. We could have replaced u with D as the highest power possible, but picking u maximal such that $a_u \neq 0$ is the main trick of the proof.

Example 4. *If we have $f(x, y) = 1 + xy + x^3y^2 + xy^200$ then we can write $f = 1 + \left(y^200 + y\right) x + 0x^2 + y^2x^3 + 0x^4 + \ldots + 0x^{201}$. But the largest nonzero coefficient is $a_3(y) = y^2$. So we would set $u = 3$ and only write $f = \sum_{i=0}^{3} a_i(y)x^i$.*

Back to the proof. Because we know that f has total degree D, it follows that $\deg(a_i(x_1, \ldots, x_{m-1})) \leq D - i$. Now fix $(b_1, \ldots, b_{m-1}) \in T^{m-1}$ and look at the univariate polynomial

$$f(b_1, \ldots, b_{m-1}, x_m) = \sum_{i=0}^{t} a_i(b_1, \ldots, b_{m-1}) x_m^i$$

If $a_t(b_1, \ldots, b_{m-1}) \neq 0$ then the above polynomial has at most u roots. Therefore we would find that

$$|\{b_m \mid f(b_1, \ldots, b_m) = 0\}| \leq u$$

.

Further we know that

$$\begin{aligned} |\{(b_1, \ldots, b_m) \in T^m \mid f(b_1, \ldots, b_m) = 0\}| \leq \quad & \left|\left\{(b_1, \ldots, b_{m-1} \in T^{m-1} \mid a_u(b_1, \ldots, b_{m-1}) = 0\right\}\right| \cdot |T| \\ & + \left|\left\{(b_1, \ldots, b_{m-1}) \in T^{m-1} \mid a_u(b_1, \ldots, b_{m-1}) \neq 0\right\}\right| \cdot u \end{aligned}$$

Let $V = \{(b_1, \ldots, b_{m-1}) \mid a_u(b_1, \ldots, b_{m-1}) \neq 0\}$. Then the number of roots of f is by the above inequality at most $|V|u + |V^c||T|$. But by the inductive hypothesis we have that

$$|V^c| \leq \deg(a_u)|T|^{m-2} \leq |D-u|T^{m-2}$$

.

So combining it all we get that

$$|\{\text{roots of } f\}| \leq (D-u)|T|^{m-1} + |T|^{m-1}u = D|T|^{m-1}$$

□

This bound can be tight for multiple reasons, and it is not easy to know when it is tight.

2.3 Examples of Polynomial Codes Over Product Sets

What parameters did this give us for multvariate codes over product sets?

- Alphabet size q
- Length $t^m = |T|^m$
- Relative distance $\delta = 1 - \frac{D}{t}$
- Dimension = number of monomials of degree $\leq D$, which is $\binom{D+m}{m}$.
- Subect to the restrictions $D \leq t$ and $t \leq q$.

We give two instantiations.

Example 5. *First the bivariate constant relative distance regime. We fix $\delta \in (0,1)$ a constant. Let $m = 2$, $t = q$, $D = (1-\delta)t$. Then we get parameters*

- *Alphabet size q*
- *Length q^2*
- *Relative distance δ*
- *Dimension $\binom{D+m}{m} \cong \frac{D^2}{2} \cong \frac{(1-\delta)^2}{2}q^2$*
- *Rate $\frac{(1-\delta)^2}{2}$*

Compare this to the Reed-Solomon code with alphabet size q, distance δ and Rate $(1-\delta)$. The bivariate code has a much longer block length for the same alphabet size, but it has a worse rate-distance tradeoff. Nevertheless, it is still a constant rate constant distance code.

Example 6. *Next we do the case when we work over the binary alphabet ($q = 2$), take t=2 (so our product set is all of $\mathbb{F}_2^m$), and linear polynomials (degree $D = 1$). We get the parameters*

- *Alphabet size* 2
- *Length* 2^m
- *Relative distance* $\frac{1}{2}$
- *Dimension* $m + 1$
- *Rate* $\frac{m+1}{2^m}$
- $|C| = 2^{m+1} = 2n$

This is tight for the Plotkin bound!

3 Decoding RS codes

In the previous section, we saw encoding of RS codes. In this section, we will see how to decode the RS codes efficiently up to half the minimum distance. We first recall the setup again.

Setup : Let

$$S = \{\alpha_1, \alpha_2, \cdots, \alpha_n\} \subseteq \mathbb{F}_q$$

be a subset of $\mathbb{F}_q$ which are the points of evaluations and D be the message length. Thus, the set of codewords are:

$$\mathsf{RS} = \{(f(\alpha_1), f(\alpha_2), \cdots, f(\alpha_n)) : f(X) \in \mathbb{F}_q[X], \mathtt{degree}(f) \leq D\}$$

We know that the minimum distance of this code is $n - D$. The decoding problem can be phrased as follows:

Given $r : S \to \mathbb{F}_q$ (" received codeword"), such that there exists $f(X) \in \mathbb{F}_q[X]$ with

(A) $\mathtt{degree}(f) \leq D$

(B) $\Delta(f_{|S}, r) < \frac{n-D}{2}$

where $\Delta(f_{|S}, r)$ denotes the disagreement of r and f restricted to the set S. Find the unique such f efficiently.

Remark: If '$< (n - D)/2$' from condition (B) is replaced with '$= 0$'. Then the problem can be solved by Lagrange interpolation. In fact, in this case we need only $D + 1$ evaluations.

One possible try is - Take any $D + 1$ points from S and interpolate a polynomial from these $D + 1$ points agreeing on r. The problem with this approach is that if r is not a low degree polynomial (of degree at most D), which is the case most of the time when there is an error in the received codeword, then we get different polynomial from interpolation for different set of $D + 1$ points.

Let us start with some *thought experiment*. Let U be a set of error locations i.e.

$$U = \{x \in S \text{ s.t } f(x) \neq r(x)\}$$

Of course, we cannot get hands our hands on the set U as we do not know f and the error locations. Define a polynomial $E(x)$ as follows:

$$E(x) = \prod_{\alpha \in U} (x - \alpha)$$

In words, $E(x)$ is a polynomial which vanishes on all the error locations. Now comes an important observation - "*zero forgives (errors)*". More specifically, we have the following identity no matter what the error locations are:

$$E(\alpha) \cdot r(\alpha) = E(\alpha) \cdot f(\alpha). \tag{1}$$

for all $\alpha \in S$. Why? Because whenever f and r differ at some $\alpha \in S$, we have $E(\alpha) = 0$ and hence in this case the identity holds. For non-error locations, the identity trivially holds.

Observation 7. *$E(x)$ has degree $= |U| < \frac{n-D}{2}$ and hence $E(x) \cdot f(x)$ has degree $< D + \frac{n-D}{2} = \frac{n+D}{2}$*

Let $E(x) \cdot f(x) =: N(x)$. Then, from Equation (1), if we have somehow magically get hands on $N(x)$ and $E(x)$ then we would be done as $f(x) = N(x)/E(x)$. The beauty of the following decoding algorithm is that although it is difficult to get exact polynomials $N(x)$ and $E(x)$ as intended in our thought experiment, we will get two polynomials $N'(x)$ and $E'(x)$ such that $N'(x)/E'(x)$ will always be equal to $f(x)$.

Now we are ready to describe the algorithm formally.

Algorithm: [Berlekamp-Welch]

Input: The received word $r : S \to \mathbb{F}_q$.
Output : The codeword at a distance at most $(n - D)/2 - 1$ (if it exists).

1. Find polynomials $E(x) \neq 0$ (of degree $< \frac{n-D}{2}$) and $N(x)$ (of degree $< \frac{n+D}{2}$) such that

$$E(\alpha) \cdot r(\alpha) = N(\alpha) \qquad \forall \alpha \in S \tag{2}$$

2. If the $E(x)$ does not divide $N(x)$ then output : given r does not satisfy property (A) or (B).

   Otherwise, output the polynomial $F(x) = \frac{N(x)}{E(x)}$.

Figure 1: Decoding Reed-Solomon codes

Remark: As noted earlier, the polynomials $N(x)$ and $E(x)$ found in first step may not be what were there in our thought experiment. But $N(x)/E(x)$ will always be $f(x)$ that we are after.

Theorem 8. *(Correctness of Berlekamp-Welch) If there exists a polynomial $f(x)$ of degree at most D such that $\Delta(f_{|S}, r) < \frac{n-D}{2}$ then the algorithm correctly outputs the polynomial f i.e it correctly decodes the received word to the nearest codeword.*

Proof. We first assume that f and r satisfy the given conditions (A) and (B). We know that Step (1) has at least one solution; take $E(x) = \prod_{\alpha \in U}(x - \alpha)$ and $N(x) = E(x) \cdot f(x)$. We want to show that for *any* pair $E(x)$ and $N(x)$ satisfying equation 2,

$$\frac{N(x)}{E(x)} = f(x).$$

Towards showing this, take the output pair $E(x), N(x)$. Look at the polynomial $h(x) := N(x) - E(x) \cdot f(x)$ where $f(x)$ is the required polynomial satisfying (A) and (B).

If given f and r satisfy the given conditions (A) and (B) then we have the following observations :

(a) $\texttt{degree}(h) < \frac{n+D}{2}$.

(b) For any $\alpha \in S$, if $r(\alpha) = f(\alpha)$ then $h(\alpha) = 0$.

Here, (a) follows from the fact that $N(x)$ has degree strictly less than $(n + D)/2$ and $E(x)$ has degree strictly less than $(n - D)/2$. Also, (b) follows from the fact that $N(\alpha) = E(\alpha) \cdot r(\alpha)$ for all $\alpha \in S$ and the definition of h.

The second observation implies that h has strictly greater than $n - \frac{n-D}{2} = \frac{n+D}{2}$ roots as f and r disagree on strictly less than $(n - D)/2$ points from S . Thus, degree of h is strictly less than the number of roots of h. We know that any non-zero polynomial of degree d has at most d roots. Hence, h must be an identically zero polynomial. Thus, $h(x) = 0$ and hence

$$N(x) = E(x) \cdot f(x) \implies f(x) = \frac{N(x)}{E(x)}.$$

□

Efficiency : In Step 1 of the algorithm, (2) represents a set of n equalities. This can be formulated as solving set of linear equations in the variables coefficients of $E(x)$ and $N(x)$. Thus, there are n unknowns and n linear equations over a field of size q. This can be solved in time $\texttt{poly}(n, q)$. Step 2 is polynomial division which can also be solved in $\texttt{poly}(n, q)$ time. Thus, the overall running time of the decoding algorithm is $\texttt{poly}(n, q)$.

4 Concatenation Codes

RS codes are good in many aspects which we saw earlier. It meets the *distance-rate* trade-off. One of the drawback of RS codes is that if one wants to get arbitrarily large distance, one has to blow up the alphabet size. In particular, the relative distance of RS codes is $1 - D/n$ where n is the

evaluation set size. As the set size is at most the base field size, the field size must be large to get a distance close to 1. Ideally, we would like to get binary (in general, codes over smaller alphabet size) with good distance and rate property. We can achieve this to some extent by the operation called *concatenation* defined as follows:

Definition 9. *Let $C \subseteq \Sigma^N$. Choose some one-to-one map $\phi : \Sigma \to \{0,1\}^n$. For the codeword $(c_1, c_2, \ldots, c_N)$ associate the codeword $(\phi(c_1), \phi(c_2), \ldots, \phi(c_N))$. Denote this new code as $C \circ \phi$. i.e*

$$C \circ \phi = \{(\phi(c_1), \phi(c_2), \ldots, \phi(c_N)) \mid (c_1, c_2, \ldots, c_N) \in C\}.$$

C is called the outer code and ϕ is called the inner code in such concatenation.

Here are few properties of the concatenated code $C \circ \phi$ which follows easily from the definition:

1. $|C \circ \phi| = |C|$.
2. $\texttt{dist}(C \circ \phi) \geq \texttt{dist}(C) \cdot \texttt{dist}(\phi)$.

Note that by choosing ϕ carefully, in some setting of C, one can make $\texttt{dist}(C \circ \phi)$ much larger than $\texttt{dist}(C) \cdot \texttt{dist}(\phi)$. More specifically, if two codewords differ at some location, then the encoding of the two symbols at this location can be more than $\texttt{dist}(\phi)$ apart. If this is true for every pair of codeword and for every location where the differ then we can get a better lower bound on the distance of the concatenated code. But, in general, without knowing the specifics of encoding, $\texttt{dist}(C) \cdot \texttt{dist}(\phi)$ is the best guarantee we can have on the distance of the concatenated code.

4.1 Why concatenation?

Alphabet size also plays a crucial role in the efficiency in encoding and decoding. Thus, smaller the alphabet size better the code is in many aspects. We saw RS codes have good distance property, but the distance parameter depends on the alphabet size. In particular, if we want relative distance to be $1 - \epsilon$ then one must take the codes over alphabet of size at least $\Omega(1/\epsilon)$.

This is where concatenation is useful. If we take a good outer code with larger alphabet size (which we know how to construct) and take a good inner code over small alphabet then concatenation gives us a code over smaller alphabet with not so bad distance property. But there is a catch - we are interested in getting good codes over smaller alphabet size but if we want to achieve it using concatenation then we also need a good inner code over the same alphabet size. This argument seems circular but the important thing to note is that if the alphabet size of the outer code is some small constant (or slow growing function of n - the length of code we are interested in) then one can just find the best code of small length over smaller alphabet by brute-forcing. As the length of this inner code is small, this step can be done *efficiently*.

4.2 One particular instantiation

In this section, we will study one particular instantiation of concatenation codes w.r.t an inner and outer code that we covered already. We will not care about the efficiency right now. In the next lecture, we will see how to construct concatenated codes efficiently with good *distance* and *rate*.

C: = Reed-Solomon code over $\Sigma := \mathbb{F}_{2^n}$ generated by degree D polynomials with evaluations over the complete field $\mathbb{F}_{2^n}$.

ϕ : = trivial one to one mapping from Σ to $\{0,1\}^n$.

We list down the properties of the Reed-Solomon codes:

- Alphabet size is $N = 2^n$.
- length of code = 2^n
- distance = D
- Number of codewords = N^{N-D+1}.

Thus, this implies that the concatenated code $C \circ \phi$ is $C \circ \phi \subseteq \{0,1\}^{Nn}$ with distance at least D. Hence, we get a binary code of length nN with distance D and $|C \circ \phi| = \frac{2^{nN}}{N^{D-1}}$.

When $D = O(1)$, a constant, then from the previous lecture (see lecture 2 notes) we know that *Gilbert-Varshamov* bound (greedy/random construction) shows a code over $\{0,1\}^{nN}$ with distance D and of size $2^{nN}/(nN)^{D-1}$. Thus, this concatenation code beats the *Gilbert-Varshamov* bound for constant distance!

In the next lecture, we will see how to construct good codes over binary alphabet efficiently using concatenation.