Introduction to Coding Theory **Técnico–ULisboa, 2026/2027**

Part 6: Reed–Solomon codes

Lecturer: João Ribeiro

Recommended reading: Essential Coding Theory Chapter 5 and Section 12.1.

1 Introduction

So far in this course we have been mostly focused on understanding the limits of various coding tasks (unique and list decoding against adversarial errors, coding against random errors). Outside the extreme high-rate/low-distance) regime we have yet to see constructions of codes with efficient (i.e., running in time polynomial in the block length) decoding procedures.

In these notes we will cover arguably the most important code construction in coding theory: *Reed–Solomon codes*. Their linearity and amazing error-correction guarantees (at the expense of a somewhat large alphabet) make Reed–Solomon codes an indispensable tool in coding theory. Beyond that, Reed–Solomon codes have also found surprising applications beyond error-correction. You will see some of these applications in the problem sets.

2 Reed–Solomon codes

Reed–Solomon codes are a family of linear codes, first proposed by Irving Reed and Gustave Solomon in 1960 [RS60].

Fix a block length n and a prime power $q \geq n$. The key idea behind Reed–Solomon codes is to first interpret a message $m = (m_0, m_1, \ldots, m_{k-1}) \in \mathbb{F}_q^k$ as a polynomial

$$f_m(x) = \sum_{i=0}^{k-1} m_i x^i \in \mathbb{F}_q[x]$$

of degree at most $k-1$. Then, to encode m we evaluate f_m on multiple points in $\mathbb{F}_q$. More precisely, fix n distinct *evaluation points* $\alpha_1, \ldots, \alpha_n \in \mathbb{F}_q$ (this why we enforce that $q \geq n$). Then, the encoding of m is

$$(f_m(\alpha_1), \ldots, f_m(\alpha_n)).$$

We are then led to the following definition.

Definition 1 (Reed–Solomon code) *Given a block length n, dimension $k \leq n$, prime power $q \geq n$, and vector of distinct evaluation points $\alpha = (\alpha_1, \ldots, \alpha_n) \in \mathbb{F}_q^n$, the* Reed–Solomon code $\mathsf{RS}_{q,\alpha}[n,k]$ *is given by*

$$\mathsf{RS}_{q,\alpha}[n,k] = \{(f_m(\alpha_1), \ldots, f_m(\alpha_n)) : m \in \mathbb{F}_q^k\}.$$

In other words, $\mathsf{RS}_{q,\alpha}[n,k]$ *is the set of evaluations of polynomials of degree at most* $k-1$ *on* $\alpha_1, \ldots, \alpha_n$.

It is not hard to show that Reed–Solomon codes are linear and to work out their minimum distance.

Theorem 1 *For any block length* n*, dimension* $k \leq n$*, prime power* $q \geq n$*, and vector of distinct evaluation points* $\alpha = (\alpha_1, \ldots, \alpha_n)$*, the code* $\mathsf{RS}_{q,\alpha}[n,k]$ *is linear and has minimum distance* $n-k+1$.

Proof: Linearity follows easily from the fact that $\alpha f_m(x) + \beta f_{m'}(x) = f_{\alpha m + \beta m'}(x)$ for all $\alpha, \beta, x \in \mathbb{F}_q$. To see the claim about the minimum distance, fix any two distinct $m, m' \in \mathbb{F}_q^k$. Then, $g(x) = f_m(x) - f_{m'}(x)$ is a nonzero polynomial of degree at most $k-1$, and so has at most $k-1$ roots by the degree mantra. It follows that $f_m(\alpha_i) \neq f_{m'}(\alpha_i)$ for at least $n-(k-1) = n-k+1$ of the α_i's (this also implies that no two messages are mapped to the same codeword, and so the code has size exactly q^k). ■

2.1 Maximum distance separable codes

As you may recall from the problem set, the *Singleton bound* says that any q-ary code of block length n and size q^k has minimum distance at most $n-k+1$. Therefore, Reed–Solomon codes attain the Singleton bound, and so provide an optimal tradeoff between code size and minimum distance. More generally, codes that attain the Singleton bound are called *Maximum Distance Separable* (MDS). MDS codes have many important applications, such as in cryptography and distributed data storage. You will see such an application in your problem set.

3 Efficient encoding of Reed–Solomon codes

We now proceed to discuss efficient encoding and decoding of Reed–Solomon codes. As we discussed before, linear codes come with efficient encoding by default. First, construct the corresponding generator matrix G. Then, to encode m we compute $c = Gm$. The generator matrix corresponding to the $\mathsf{RS}_{q,\alpha}[n,k]$ code is the $n \times k$ Vandermonde matrix

$$G = \begin{pmatrix} 1 & \alpha_1 & \alpha_1^2 & \ldots & \alpha_1^{k-1} \\ 1 & \alpha_2 & \alpha_2^2 & \ldots & \alpha_2^{k-1} \\ \vdots & \vdots & \vdots & \ldots & \vdots \\ 1 & \alpha_n & \alpha_n^2 & \ldots & \alpha_n^{k-1} \end{pmatrix}.$$

In the problem set you will work out the dual perspective on Reed–Solomon codes, which is also clean and useful.

There are a few details worth discussing. First, efficiently constructing G and efficiently computing Gm for an arbitrary $m \in \mathbb{F}_q^k$ both assume that we can efficiently perform arithmetic over $\mathbb{F}_q$. Indeed, we have very fast algorithms for this that run in time roughly $\log q$, but in turn they assume access to a "description" of $\mathbb{F}_q$. If $q = p^r$ for a prime p, this description would correspond to an irreducible

degree-r polynomial over $\mathbb{F}_p$. Whenever p is not very large compared to n (which is the usual regime for us), we know how to efficiently (and deterministically) construct such an irreducible polynomial. Second, there are even more efficient ways to encode a message m than naively computing the product Gm, through fast multipoint evaluation algorithms. If you are interested in learning more about this, see the excellent books by von zur Gathen and Gerhard [vzGG13] and by Shoup [Sho05].

From here onward we will ignore the above and compute complexity in terms of "$\mathbb{F}_q$-operations".

4 Efficient decoding of Reed–Solomon codes: The Berlekamp-Welch algorithm

From Theorem 1 we know that $\mathsf{RS}_{q,\alpha}[n,k]$ corrects any $t < \frac{n-k+1}{2}$ errors. In this section we will see how this can be done efficiently (i.e., in a number of operations polynomial in n), via the *Berlekamp-Welch* algorithm [BW83].

Suppose that we receive $y \in \mathbb{F}_q^n$, obtained by introducing any up to $t < \frac{n-k+1}{2}$ errors in a codeword of $\mathsf{RS}_{q,\alpha}[n,k]$. Then, this means that there exists a polynomial $P \in \mathbb{F}_q[x]$ of degree at most $k-1$ such that

$$P(\alpha_i) \neq y_i$$

for at most t of the α_i's. Our goal is to find P from y, as the correct message sent corresponds to the coefficients of P.

Towards this, we introduce the notion of an *error locator polynomial*. A nonzero polynomial $E \in \mathbb{F}_q[x]$ is an error locator polynomial if $E(\alpha_i) = 0$ whenever $P(\alpha_i) \neq y_i$ (note that we impose no constraint on the behavior of $E(x)$ outside the α_i's for which $P(\alpha_i) \neq y_i$; in particular, $E(x)$ may have more roots). We know that there exists an error locator polynomial of degree at most t,

$$E(x) = \prod_{i : P(\alpha_i) \neq y_i} (x - \alpha_i).$$

However, we do not know this polynomial. And, in fact, finding such an error locator polynomial E (of degree at most t) would allow us to recover P (do you see how?).

It is not hard to see that the polynomials P and E satisfy

$$y_i E(\alpha_i) = P(\alpha_i) E(\alpha_i), \quad \forall i \in \{1, \ldots, n\}. \tag{1}$$

If we interpret the coefficients of P and E as unknown variables, then Equation (1) gives a system of n quadratic equations on at most $k+t+1$ variables. Solving this system of equations would allow us to find P, and we know that a solution exists. However, solving systems of quadratic equations is very hard in general.

The key insight behind the Berlekamp-Welch decoding algorithm is to consider a *linearized* version of this system of equations, and show that solving this linearized system still allows us to recover P correctly. Let's recap. The original problem discussed above is the following: Given $y \in \mathbb{F}_q^n$, find polynomials $E, P \in \mathbb{F}_q[x]$ such that:

1. $E(x)$ is nonzero and has degree at most t;
2. $P(x)$ has degree at most $k-1$;
3. $y_i E(\alpha_i) = P(\alpha_i)E(\alpha_i)$ for all $i \in \{1, \ldots, n\}$.

Instead, we consider the following easier *linearized problem*: Given $y \in \mathbb{F}_q^n$, find polynomials $E, N \in \mathbb{F}_q[x]$ such that:

1. $E(x)$ is monic and has degree exactly t;[1]
2. $N(x)$ has degree at most $t+k-1$;
3. $yE(\alpha_i) = N(\alpha_i)$ for all $i \in \{1, \ldots, n\}$.

Intuitively, $N(x)$ plays the role of $E(x) \cdot P(x)$. Replacing $E(x) \cdot P(x)$ by $N(x)$ turns the problem into solving a system of *linear* equations on the coefficients of E and N, which we can solve efficiently via Gaussian elimination. There are $n+1$ equations (the n equations $yE(\alpha_i) = N(\alpha_i)$, plus the equation $E_t = 1$, where E_t denotes the degree-t coefficient of E, since E is required to be monic of degree t), and the number of variables is

$$(t+1) + (t+k) = 2t + k + 1 < (n-k+1) + k + 1 = n + 2$$

by the assumption on the number of errors t. In sum, there are at least as many equations as variables in this system of linear equations. An important caveat is that, at least at first sight, the N we find may not satisfy the desired equality $N(x) = E(x)P(x)$...

We will now show that this algorithm always returns $E(x)$ and $N(x)$ satisfying $N(x) = E(x)P(x)$. In turn, this means that we can recover $P(x)$ efficiently from y, since (1) we obtain $E(x)$ and $N(x)$ by solving a system of linear equations, and (2) we obtain $P(x)$ from $E(x)$ and $N(x)$ by, say, long division of polynomials, which can be performed efficiently.

The algorithm's correctness claim follows by combining two lemmas. The first lemma states that there exists a solution $(E^\star, N^\star)$ such that $N^\star(x) \equiv E^\star(x)P(x)$. The second lemma states that any solution (E, N) satisfies $N(x) \equiv E(x)P(x)$.

Lemma 1 *There exists a solution $(E^\star, N^\star)$ to the linearized problem above satisfying $N^\star(x) \equiv E^\star(x)P(x)$.*

Proof: First, we observe that there exists a monic error locator polynomial of degree exactly t, namely

$$E^\star(x) = x^{t - d(y, (P(\alpha_i))_{i \in \{1,\ldots,n\}})} \prod_{i : y_i \neq P(\alpha_i)} (x - \alpha_i).$$

[1]The requirement that $E(x)$ is monic and has degree exactly t (instead of just requiring that $E(x)$ has degree at most t) is just a simple way of avoiding the trivial solution where $E(x)$ is the zero polynomial by imposing an additional linear constraint.

Then, set

$$N^\star(x) = E^\star(x)P(x).$$

It follows from Equation (1) that $y_i E^\star(\alpha_i) = E^\star(\alpha_i)P(\alpha_i) = N(\alpha_i)$ for all i, and so $(E^\star, N^\star)$ is a solution to the linearized problem. ■

Lemma 2 *Fix an arbitrary solution (E, N) of the linearized problem. Then, $N(x) \equiv P(x)E(x)$.*

Proof: Consider the polynomial

$$R(x) = N(x) - P(x)E(x).$$

We will show that $R(x) \equiv 0$, from which it follows that $N(x) \equiv P(x)E(x)$, as desired.

We will compare the degree of $R(x)$ with how many roots it has. First, note that $R(x)$ has degree at most $t + k - 1$. On the other hand, using the fact that $y_i E(\alpha_i) = N(\alpha_i)$ for every solution (E, N), we have

$$R(\alpha_i) = N(\alpha_i) - P(\alpha_i)E(\alpha_i) = E(\alpha_i)(y_i - P(\alpha_i))$$

for every $i \in \{1, \ldots, n\}$. In particular, this means that

$$P(\alpha_i) = y_i \implies R(\alpha_i) = 0.$$

Since $P(\alpha_i) = y_i$ for at least $n - t$ indices i, we conclude that $R(x)$ has at least $n - t$ roots. Now, note that

$$n - t > t + k - 1 \iff n - k + 1 > 2t,$$

which holds by our assumption that $t < \frac{n-k+1}{2}$. Therefore, by the degree mantra, we conclude that $R(x) \equiv 0$. ■

References

[BW83] Elwyn R. Berlekamp and Lloyd R. Welch. Error correction for algebraic block codes, 1983. U.S. Patent US4633470A.

[RS60] Irving S. Reed and Gustave Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, 1960.

[Sho05] Victor Shoup. *A Computational Introduction to Number Theory and Algebra*. Cambridge University Press, 2005. Available at https://shoup.net/ntb/.

[vzGG13] Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra*. Cambridge University Press, 2013.