15:853: Algorithms in the Real World
Fall 2019

Lecture 5

September 17 2019

Lecturer: Rashmi Vinayak *Scribe: Sam Westrick*

5.1 Recap

A codeword of a (n, k, d) **R**eed-**S**olomon (RS) code consists of n distinct sample points of a polynomial of degree $k-1$. Given message a consisting of a_i for $i = 0, \ldots, k-1$ where $a_i \in \mathbb{F}_{q^r}$, we interpret the message as a representation of the degree $k-1$ polynomial P, defined by

$$P(x) = \sum_{i=0}^{k-1} a_i x^i.$$

This polynomial is then encoded into the codeword c defined by $c_i = P(\alpha_i)$ for $i = 1, \ldots, n$, using some n fixed sample points α_i. The n sample points α_i *must be distinct*, which is only possible when $q^r \geq n$.

Note that for the sake of our discussion, the choices of α_i do not matter, **so from now on we will assume** $\alpha_i = i$.

The $k \times n$ *generator* matrix G for RS codes has Vandermonde form, as shown below on the left (a). It may be helpful to examine the generic version of this structure, written in terms of generic sample points α_i, as shown below on the right (b). For any input message a and corresponding P, observe that $aG = [P(\alpha_1)\ P(\alpha_2)\ \cdots\ P(\alpha_n)]$.

$$G = \begin{bmatrix} 1 & 1 & \ldots & 1 \\ 1 & 2 & \ldots & n \\ 1 & 2^2 & \ldots & n^2 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 2^{k-1} & \ldots & n^{k-1} \end{bmatrix}$$

(a) RS generator for sample points $\alpha_i = i$

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

(b) RS generator for generic sample points α_i

Finally, recall that RS codes are **M**aximum-**D**istance-**S**eparable (MDS), meaning that they have minimum distance $d = n - k + 1$.

5.2 Overview

In this lecture we covered two topics:

1. decoding RS codes
2. **concatenation** codes, a technique for composing code strategies, allowing us to create new codes from existing codes

5.3 Decoding RS Codes

5.3.1 Preliminaries

Theorem 5.1. *For any $k+1$ points (x_i, y_i) with distinct x_i, there is* exactly one *degree-k polynomial P that satisfies $P(x_i) = y_i$ for all i.*

Proof: First, we prove that there is *at most* one, then we prove that there is *at least* one.

- At most one: suppose we have two degree-k polynomials P and Q that agree on $k+1$ points $P(x_i) = Q(x_i) = y_i$. We want to show $P = Q$. Consider polynomial $P - Q$. It has $k+1$ roots, but degree at most k. So it must be that $P - Q = 0$.
- At least one: we can construct a degree-k polynomial that satisfies $P(x_i) = y_i$ for all i using Lagrange interpolation on the points (x_i, y_i).

□

5.3.2 Decoding

RS codes are based on the following remarkably simple idea: by Theorem 5.1, it should be possible to encode a polynomial as a set of its points, because these points are sufficient to recover the original polynomial. Furthermore, if we use more points than necessary, the encoding strategy has redundancy and therefore should be useful as an error-correcting code.

In particular, if we encode a degree-$(k-1)$ polynomial with n points, then we can trivially recover from up to $n-k$ erasures with Lagrange interpolation. With up to $n-k$ erasures we are left with at least k valid points, which are sufficient to interpolate a polynomial of degree $k-1$, and by Theorem 5.1 we know such polynomial is unique, so it must be our original polynomial.

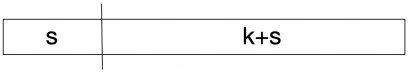
Another way of looking at this is to consider the generator matrix G, and observe that G is full-rank. Therefore we can take any k columns from G to form a matrix $\widetilde{G}$ of dimension $k \times k$ which is invertible. In particular, we can choose the k columns at the same positions as the k non-erased points. Recovering from erasures is then essentially just solving the equation $a\widetilde{G} = \widetilde{c}$ where a is the input message (the coefficients of the polynomial) and $\widetilde{c}$ is the noisy codeword with all erasures removed.

However, there is a tricky problem that we need to solve in order for this approach to be feasible: if some of the points are erroneous, how do we efficiently recover the original polynomial?

5.3.3 Decoding under Errors

Suppose we have up to s errors, i.e. $d \geq 2s+1$. Because RS codes are MDS, we know $d = n-k+1$, and so therefore $n \geq k+2s$. Let's consider the case where $n = k+2s$. How can we decode?

Naïve Algorithm. Visually, we can think of our codeword as being divided into s errors and $k+s$ valid points:



A correct—but terribly inefficient—algorithm is to guess the locations of the $k+s$ valid points, and check if these points agree on a degree-$(k-1)$ polynomial (specifically, interpolate a polynomial using k of the points and then check that the remaining s points agree). If this succeeds, then we are done. But if the $k+s$ guessed points do not agree, we have to guess again. In the worst case, we will have to do up to $\binom{n}{s}$ guesses! We can do better.

Berlekamp-Welch Algorithm. Let c be the correct codeword ($c_i = P(i)$ for $i \in [n]$), and c' be the received codeword. Consider the set $S = \{i \in [n] \mid c_i \neq c'_i\}$ of erroneous positions. Wouldn't it be nice if there was a magical "error-locator" polynomial E where $E(i) = 0$ for every $i \in S$? Surely it exists, and has degree $s = |S|$, for example $E(x) = \prod_{i \in S}(x-i)$. The problem is that we don't know S.

One thing we do know is that the following equalities hold:

$$P(i)E(i) = c'_i E(i), \quad i \in [n]$$

So, let $R(i) = P(i)E(i)$. This polynomial R has degree $k+s-1$, and so it has $k+s$ "degrees of freedom" (unknown coefficients). Similarly, E has $s+1$ degrees of freedom. So, the system of equations $R(i) = c'_i E(i)$ has $(k+s)+(s+1) = n+1$ degrees of freedom.

The Berlekamp-Welch algorithm is simply to solve this system of equations, which produces R and E, and then compute $P = \frac{R}{E}$.

At first it might seem like there is a problem, because we have n equations and $n+1$ degrees of freedom, so the system is *under-constrained*, and there could be many such solutions (R, E). It turns out, however, that for any two possible solutions (R_1, E_1) and (R_2, E_2), we have $\frac{R_1}{E_1} = \frac{R_2}{E_2}$. *(Proof left as an exercise!)*

Alternatively, we can side-step the problem entirely by further constraining E, so that we have exactly n degrees of freedom. We already know that there exists a particular E which has a leading coefficient of 1 (recall from above that $E(x) = \prod_{i \in S}(x-i)$ suffices). So, by constraining this leading coefficient, we have exactly n degrees of freedom, and then there is a unique solution (R, E) to the system of equations.

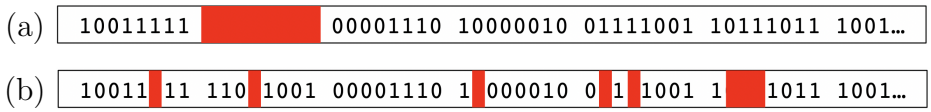


Figure 5.1. Good (a) and bad (b) cases for a $(255, 253, 3)_{256}$ RS code

5.4 Concatenation Codes

RS codes are good for correcting "burst" bit errors, i.e. when many bit errors are clustered together contiguously within messages. This is because RS relies on having a large alphabet size (recall that $q^r \geq n$ is necessary); therefore to encode a message of bits, we have to cluster the bits into larger message symbols. For example, we might cluster a bitstream into a stream of bytes. RS codes then are useful for detecting when entire bytes are incorrect, for example as depicted in case (a) of Figure 5.1.

But what do we do if the typical errors we need to handle are not bursty? For example, what if it is common to periodically see one bit error, as shown in case (b) of Figure 5.1? RS codes alone do not handle such cases well.

Perhaps we could design another code specially for this scenario. But more generally, we need a scheme for generating codes for any possible error behavior.

5.4.1 Concatenation

Concatenation is a technique for combining two codes, to gain the benefit of both. Suppose we have two codes, $(N, K, D)_{q^k}$ and $(n, k, d)_q$, which we will call the "outer" and "inner" code, respectively. Each symbol of the outer code consists of k symbols of the inner code. So, after encoding with the outer code, we can then further encode each resulting symbol with the inner code. This is shown in Figure 5.2. The resulting code is $(Nn, Kk, Dd)_q$.

Why is the minimum distance of the resulting code Dd? Because the outer code guarantees a distance of D outer symbols, and for each of these symbols, the inner code guarantees a distance of d inner symbols.

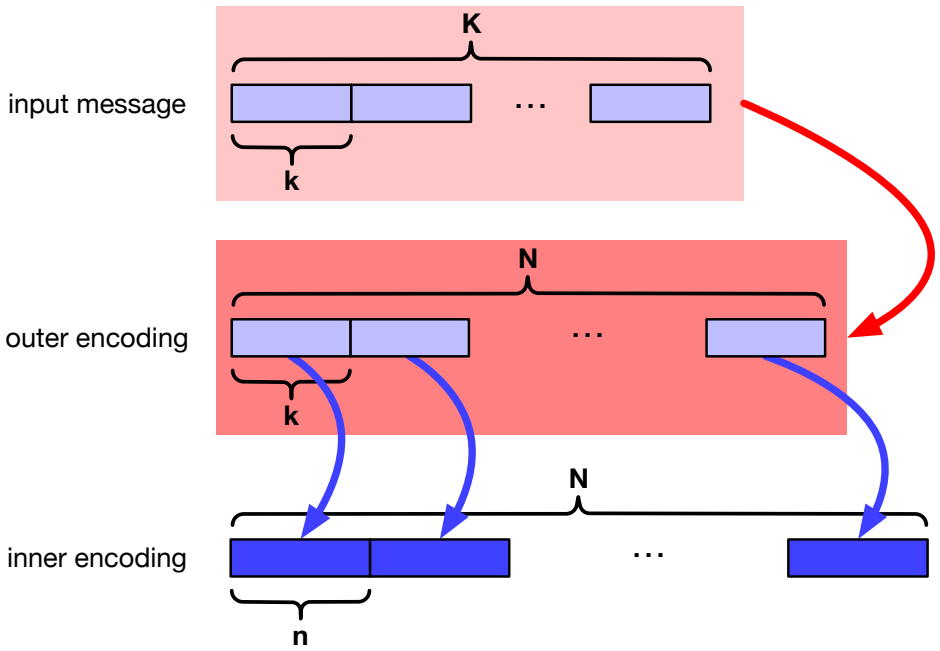


Figure 5.2. Concatenation of outer $(N, K, D)_{q^k}$ code and inner $(n, k, d)_q$ code, producing a $(Nn, Kk, Dd)_q$ code.