1 List Decoding up to Capacity

Previously we looked at how a generalization of the Welch–Berlekamp could be used to efficiently list decode Reed-Solomon codes up to the Johnson bound of $1 - \sqrt{R}$ fraction of errors. However, this still falls short of the list decoding capacity of $1 - R$ we hope to achieve. The main problem with Reed-Solomon codes is that because we have a large block length, we have to be able to handle a large number of different error patterns. To circumvent this we introduced Folded Reed-Solomon codes (FRS) which pack together the codewords into blocks. This decreases the block length, reducing the number of error patterns to handle, at the cost of increasing the alphabet size.

2 Folded Reed-Solomon Codes

Given a finite field $\mathbb{F}$ of size $|\mathbb{F}| = q$ recall that a Reed-Solomon code $RS_{\mathbb{F}}[\mathbb{F}^*, n, k] : \mathbb{F}^{k+1} \to \mathbb{F}^n$ gives an encoding by viewing $f \in \mathbb{F}^{k+1}$ as a degree k polynomial[1] in $\mathbb{F}[x]$ via the evaluation map:

$$f(x) \mapsto (f(\alpha))_{\alpha \in \mathbb{F}^*} = (f(1), f(\gamma), ..., f(\gamma^{n-1}))$$

where γ is a generator of the multiplicative group $\mathbb{F}^*$ and $n = q - 1$.

An m-Folded Reed-Solomon code $FRS_{\mathbb{F}}^{(m)}[k] : \mathbb{F}^k \to (\mathbb{F}^m)^{n/m}$ simply takes the above codewords and groups them into blocks of size m where for simplicity assume m divides n:

$$f(x) \mapsto \left(\begin{bmatrix} f(1) \\ f(\gamma) \\ \vdots \\ f(\gamma^{m-1}) \end{bmatrix}, \begin{bmatrix} f(\gamma^m) \\ f(\gamma^{m+1}) \\ \vdots \\ f(\gamma^{2m-1}) \end{bmatrix}, \ldots, \begin{bmatrix} f(\gamma^{n-m}) \\ f(\gamma^{n-m+1}) \\ \vdots \\ f(\gamma^{n-1}) \end{bmatrix} \right).$$

Clearly this code has the same rate as the original Reed-Solomon code as we have scaled up the alphabet dimension (from $1 = \log_q q$ to $m = \log_q q^m$) and scaled down the block length (from n to $N = n/m$) by the same factor. Hence FRS also meets the singleton bound $\delta = 1 - R$. This change in perspective seems innocuous at first, but its real power comes from the fact that the alphabet $\mathbb{F}^m$ has nice geometric properties that the lower dimensional space $\mathbb{F}$ lacks. In particular, saying two vectors agree in $\mathbb{F}^m$ is a much stronger condition than saying two elements in $\mathbb{F}$ agree as the former can differ in many more ways. Namely, we are essentially forcing an error model on Reed-Solomon codes where we have long contiguous sections of error free-regions, and errors are occurring in blocked regions rather than uniformly throughout. This also makes this code attractive in error models where errors are assumed to come in dense blocks rather than uniformly [GR06].

[1] Note here we differ from our usual convention of considering degree $k - 1$ polynomials.

3 Linear Algebraic List Decoding of Folded Reed-Solomon Codes

Our main result is that Folded Reed-Solomon codes achieve list decoding capacity. The proof reduces the problem of decoding to solving a pair of carefully set up linear systems which naturally implies the efficiency of decodability [GW13].

Theorem 19.1. *For any $s \in [[1, m]]$ the code $FRS_{\mathbb{F}}^{(m)}[k]$ is efficiently (ρ, L)- list decodable up to error fraction:*

$$\rho := \frac{s}{s+1}\left(1 - \frac{mR}{m-s+1}\right)$$

with list size $L \leq q^{s-1}$.

Remark 19.2. In fact, the codeword list will be a subspace of $(\mathbb{F}^m)^{n/m}$ and will often be much smaller than q^{s-1}.

Before we prove the theorem, let us consider the consequences of a few choices of s. For $s = 1$ we have by the above theorem $\rho = \frac{1}{2}(1 - R)$ and $L = 1$, and hence we recover the original unique decodability bound of decoding up to half the distance. For $s = \frac{1}{\epsilon}$ and $m = \frac{1}{\epsilon^2}$ we get $\rho \geq 1 - R - \epsilon$ and $L \leq q^{O(1/\epsilon)}$ over an alphabet of size $q^{O(1/\epsilon)}$. Namely, if we allow for a polynomially large alphabet size then FRS achieves list capacity with a polynomial list size as desired (remember our block length $n = q - 1$). Moreover, this code is efficiently decodable unlike the random codes we had encountered earlier that share these properties.

As before, the proof of this theorem is motivated by the original Welch-Berlekamp algorithm: we interpolate a polynomial and force certain constraints which guarantee the true codeword is among one of the solutions as long as there are not too many errors. Remember, the original algorithm searched for an "error-detector" polynomial $Q(x, y) = A_0(x) + A_1(x)y$ satisfying the constraints $Q(\alpha_i, y_i) = 0$. By limiting the degree of the polynomials A_0, A_1 this forced the true codeword $f(x)$ to satisfy $Q(x, f(x)) = 0$, aka $y - f(x)$ divides $Q(x, y)$, as long as there were not too many errors. From here, $f(x)$ could be recovered from A_0, A_1. The new algorithm simply replaces Q by a multivariate polynomial, but is a lot more careful on how it defines its interpolation constraints.

3.1 Description of Algorithm

Proof Theorem 19.1: First we describe the decoding algorithm and later prove its correctness and analyze the running time. Define a multivariate polynomial:

$$Q(x, Y_1, \ldots, Y_s) := A_0(x) + Y_1 A_1(x) + \ldots + Y_s A_s(x).$$

If $FRS(f(x))$ is the true codeword, we aim to constrain Q in such a way so that:

$$R_f(x) := Q(x, f(x), f(\gamma x), \ldots, f(\gamma^{s-1} x)) \equiv 0$$

as long as there are not too many errors. For a fixed integer $D > 0$, to be chosen later, since $f(x)$ is degree k we can force $R_f(x)$ to have degree at most D by requiring:

$$degree(A_0) \leq D \quad and \quad degree(A_i) \leq D - k \quad for \quad i = 1, \ldots, s.$$

Note that in total our polynomial Q has $(D+1) + (D-k+1) \cdot s$ coefficients in it across all the A_i. If $y \in (\mathbb{F}^m)^{n/m} \simeq \mathbb{F}^n$ is the received word, we further constrain Q (and hence also R_f) by requiring

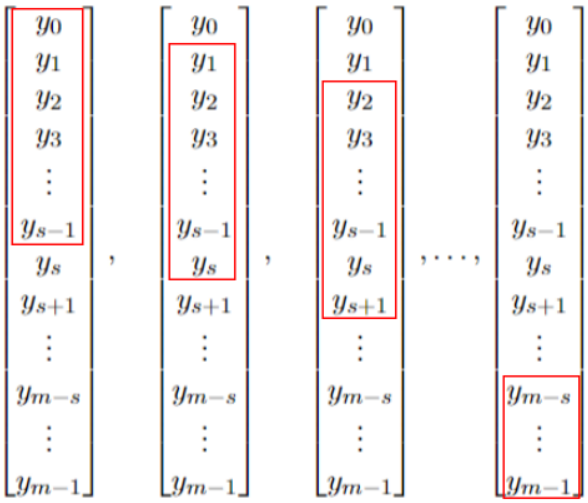

Figure 19.1: For each of the N columns of the received word y we generate $m - s + 1$ constraints on Q by sliding a contiguous window of size s down the column. An example of this process on the first column of y is shown above where the red box represents the window.

the following size s "sliding window" interpolation conditions for all $j \in \{0, m, 2m, ..., n\}$:

$$\begin{aligned} Q(\gamma^j, y_j, y_{j+1}, ..., y_{j+s-1}) &= 0 \\ Q(\gamma^{j+1}, y_{j+1}, y_{j+2}, ..., y_{j+s}) &= 0 \\ &\dots \\ Q(\gamma^{j+m-s}, y_{j+m-s}, y_{j+m-s+1}, ..., y_{j+m-1}) &= 0 \end{aligned}$$

Namely, for each of the N columns of the received word, we slide a contiguous window of size s down the column to generate $m - s + 1$ constraints on Q. In total, this gives $N \cdot (m - s + 1)$ constraints on Q.[2] This process is depicted and Figure 19.1.

Now that Q and the constraints are defined, the algorithm is simple:

1. Find a nonzero polynomial Q of the form described above satisfying the interpolation conditions.

2. Output a list of all polynomials $g(x)$ of degree at most k satisfying $R_g(x) \equiv 0$.

With the algorithm described, we move on to analyzing its correctness and running time.

3.2 Correctness

Finding a Q satisfying the desired constraints merely involves solving a linear system for the $(D + 1) + (D - k + 1) \cdot s$ coefficients in Q. To ensure this system is not over-determined we require there to be more unknowns (our coefficients) than there are constraints (our interpolation

[2]Note in the case $s = 1$ this reduces to the constraints $Q(\gamma^i, y_i) = 0$ of the original Welsh-Berlekamp algorithm.

conditions):

$$(D+1)+(D-k+1)\cdot s > N\cdot(m-s+1) \quad \text{which implies} \quad D \geq k\frac{s}{s+1}+\frac{N\cdot(m-s+1)}{s+1}.$$

Next, we show if there are not too many errors then the true codeword $FRS(f(x))$ satisfies $R_f \equiv 0$. Suppose the true codeword $FRS(f(x))$ agrees with the received word y in at least t columns (e.g. there are at most $N-t$ errors). For each agreeing column $j \in \{0, m, 2m, ..., n\}$ the interpolation constraints imply:

$$R_f(\gamma^j) = R_f(\gamma^{j+1}) = \ldots = R_f(\gamma^{j+m-s}) = 0.$$

Namely, each of the t agreements force R_f to have $(m-s+1)$ zeros. Since R_f is a polynomial of degree at most D, then if $t(m-s+1) > D$ we must have $R_f \equiv 0$. Using the lowerbound we found on D above and the fact $k+1 = Rn = RNm$ implies it suffices to have:

$$t(m-s+1) > D \geq k\frac{s}{s+1}+\frac{N\cdot(m-s+1)}{s+1} \quad \text{implying} \quad 1-\frac{t}{N} \leq \frac{s}{s+1}\left(1-\frac{mR}{m-s+1}\right).$$

This shows as long as the error fraction falls below the ρ mentioned in the theorem statement, the true codeword will be within the list outputted by the algorithm.

Finally, we prove the algorithm produces the desired list size. The desired result is an immediate consequence of the following lemma.

Lemma 19.3. *For fixed polynomials $A_0, A_1, ..., A_s$, the set of solutions to:*

$$R_g(x) = Q(x, g(x), g(\gamma x), ..., g(\gamma^{s-1}x)) \equiv 0, \quad degree(g) \leq k$$

form an affine subspace over $\mathbb{F}$ of dimension at most $s-1$.

Proof of Lemma 19.3: First clearly the set of solutions is an affine subspace as it is the solution set to an inhomogenous linear system of equations. Namely we are forcing:

$$A_1(x)g(x) + ... + A_s(x)g(\gamma^{s-1}) = -A_0(x) \tag{19.1}$$

and the left hand side is linear in the coefficients of g. So now we just need to find the dimension of this space.

Without loss of generality we can assume not all of $A_1, ..., A_s$ are divisible by x, as if this were true then x would have to divide A_0 as well as a consequence of equation 19.1. Since none of our interpolation conditions require $x = 0$ we can divide out x. Hence suppose that:

$$A_i = a_{i0} + a_{i1}x + ... \quad \text{and} \quad g(x) = g_0 + g_1x + ... + g_kx^k$$

Then the fact that $R_g \equiv 0$ forces a relationship between all these coefficients just by collecting like terms:

$$\begin{aligned} 0 &= R_g(x) \\ &= A_0(x) + A_1(x)g(x) + ... + A_s(x)g(\gamma^{s-1}x) \\ &= (a_{00} + a_{10}g_0 + ... + a_{s0}g_0) + (a_{01} + [a_{10}\gamma g_1 + a_{11}g_0] + ... + [a_{s0}\gamma^{s-1}g_1 + a_{s1}g_0])x + ... \end{aligned}$$

Since $R_g \equiv 0$ all its coefficients must be zero. Setting the first term to zero in the last expression above gives us a relationship for g_0. Once we know this, the second term gives us a relationship for g_1, and so on. If we let $B(x) := a_{10} + ... + a_{s0}x^{s-1}$ these relationships look like:

$$\begin{aligned}
B(\gamma^0)g_0 &= -a_{00} \\
B(\gamma^1)g_1 &= -a_{01} - (a_{11} + ... + a_{s1})g_0 \\
&\dots \\
B(\gamma^\ell)g_\ell &= -a_{0\ell} - (\dots)g_1 - (\dots)g_2 - \dots - (\dots)g_{\ell-1} \\
&\dots
\end{aligned}$$

Note that by our assumption that x does not divide all the A_i we know B is not identically zero as if x does not divide A_i then $a_{i0} \neq 0$. Thus B has at most $s-1$ zeros as it is degree at most $s-1$, so $B(\gamma^\ell) = 0$ for at most $s-1$ choices of ℓ. Once we fix the g_ℓ for such ℓ, the other coefficients g_i are completely determined by the above relationship. Hence the affine subspace has dimension at most $s-1$. □

3.3 Running Time

Lastly, we analyze the running time of this algorithm. As mentioned earlier, finding Q requires solving a linear system with less than $D(s+1)$ unknowns and $Nm = n$ constraints. Note our algorithm requires $D < t(m-s+1) \leq n$, so clearly this will be $poly(n)$. As discussed in the proof of the preceding lemma, finding solutions g to $R_g(x) \equiv 0$ again amounts to solving an inhomogeneous linear system so this too will be $poly(n)$.

This completes the proof of the theorem. □

4 Improvements on List Size and Running Time Analysis

The above decoding scheme guarantees only that the list size is polynomial in the block length. However, as briefly mentioned in the discussion of Lemma 19.3 the bound on the list size is quite rough as it takes the entire span of candidate codewords. Recent work improves upon this bound to get a constant upperbound on the list size by using the subspace found in Lemma 19.3 as the input to a zero-error list recovery problem. Namely, we know the true list $\mathcal{L}$ is a subset of an affine subspace $V + x_0$ of dimension at most $s-1$, and careful pruning of this subspace can drastically reduce the number of candidate codewords it contains [KRZSW18].

Lastly, note the running time analysis above is very rough. By exploiting some of the special structure of the linear systems involved in this algorithm, the analysis can be greatly improved. Section 18.3.3 of the course text discusses these ideas further.

References

[GR06] Venkatesan Guruswami and Atri Rudra. Achieving list decoding capacity using folded reed-solomon codes. *Scopus*, 2006. 2

[GW13] Venkatesan Guruswami and Carol Wang. Linear-algebraic list decoding for variants of reed–solomon codes. *IEEE Transactions on Information Theory*, 59:3257–3268, 2013. 3

[KRZSW18] Swastik Kopparty, Noga Ron-Zewi, Shubhangi Saraf, and Mary Wootters. Improved decoding of folded reed-solomon and multiplicity codes, 2018. 4