# Foreword

This chapter is based on lecture notes from coding theory courses taught by Venkatesan Guruswami at University at Washington and CMU; by Atri Rudra at University at Buffalo, SUNY and by Madhu Sudan at MIT.

This version is dated **May 1, 2013**. For the latest version, please go to

http://www.cse.buffalo.edu/ atri/courses/coding-theory/book/

The material in this chapter is supported in part by the National Science Foundation under CAREER grant CCF-0844796. Any opinions, findings and conclusions or recomendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation (NSF).

# Chapter 14

# Efficiently Achieving List Decoding Capacity

In the previous chapters, we have seen these results related to list decoding:

- Reed-Solomon codes of rate $R > 0$ can be list-decoded in polynomial time from $1-\sqrt{R}$ errors (Theorem 13.2.6). This is the best algorithmic list decoding result we have seen so far.
- There exist codes of rate $R > 0$ that are $\left(1-R-\varepsilon, O\left(\frac{1}{\varepsilon}\right)\right)$-list decodable for $q \geq 2^{\Omega(\frac{1}{\varepsilon})}$ (and in particular for $q = \text{poly}(n)$) (Theorem 7.4.1 and Proposition 3.3.2). This of course is the best possible combinatorial result.

Note that there is a gap between the algorithmic result and the best possible combinatorial result. This leads to the following natural question:

**Question 14.0.1.** *Are there explicit codes of rate $R > 0$ that can be list-decoded in polynomial time from $1-R-\varepsilon$ fraction of errors for $q \leq poly(n)$?*

In this chapter, we will answer Question 14.0.1 in the affirmative.

## 14.1 Folded Reed-Solomon Codes

We will now introduce a new type of code called the Folded Reed-Solomon codes. These codes are constructed by combining every $m$ consecutive symbols of a regular Reed-Solomon code into one symbol from a larger alphabet. Note that we have already seen such a folding trick when we instantiated the outer code in the concatenated code that allowed us to efficiently achieve the $\text{BSC}_p$ capacity (Section 12.4.1). For a Reed-Solomon code that maps $\mathbb{F}_q^k \to \mathbb{F}_q^n$, the corresponding Folded Reed-Solomon code will map $\mathbb{F}_q^k \to \mathbb{F}_{q^m}^{n/m}$. We will analyze Folded Reed-Solomon codes that are derived from Reed-Solomon codes with evaluation $\{1, \gamma, \gamma^2, \gamma^3, \ldots, \gamma^{n-1}\}$, where $\gamma$ is the generator of $\mathbb{F}_q^*$ and $n \leq q-1$. Note that in the Reed-Solomon code, a message is encoded as in Figure 14.1.

| | | | | | | |
|---|---|---|---|---|---|---|
| $f(1)$ | $f(\gamma)$ | $f(\gamma^2)$ | $f(\gamma^3)$ | $\cdots$ | $f(\gamma^{n-2})$ | $f(\gamma^{n-1})$ |

Figure 14.1: Encoding $f(X)$ of degree $\le k-1$ and coefficients in $\mathbb{F}_q$ corresponding to the symbols in the message.

For $m = 2$, the conversion from Reed-Solomon to Folded Reed-Solomon can be visualized as in Figure 14.2 (where we assume $n$ is even).

| | | | | | | |
|---|---|---|---|---|---|---|
| $f(1)$ | $f(\gamma)$ | $f(\gamma^2)$ | $f(\gamma^3)$ | $\cdots$ | $f(\gamma^{n-2})$ | $f(\gamma^{n-1})$ |

$$\Downarrow$$

| | | | |
|---|---|---|---|
| $f(1)$<br>$f(\gamma)$ | $f(\gamma^2)$<br>$f(\gamma^3)$ | $\cdots$ | $f(\gamma^{n-2})$<br>$f(\gamma^{n-1})$ |

Figure 14.2: Folded Reed-Solomon code for $m = 2$

For general $m \ge 1$, this transformation will be as in Figure 14.3 (where we assume that $m$ divides $n$).

| | | | | | | |
|---|---|---|---|---|---|---|
| $f(1)$ | $f(\gamma)$ | $f(\gamma^2)$ | $f(\gamma^3)$ | $\cdots$ | $f(\gamma^{n-2})$ | $f(\gamma^{n-1})$ |

$$\Downarrow$$

| | | | | |
|---|---|---|---|---|
| $f(1)$<br>$f(\gamma)$<br>$\vdots$<br>$f(\gamma^{m-1})$ | $f(\gamma^m)$<br>$f(\gamma^{m+1})$<br>$\vdots$<br>$f(\gamma^{2m-1})$ | $f(\gamma^{2m})$<br>$f(\gamma^{2m+1})$<br>$\vdots$<br>$f(\gamma^{3m-1})$ | $\cdots$ | $f(\gamma^{n-m})$<br>$f(\gamma^{n-m+1})$<br>$\vdots$<br>$f(\gamma^{n-1})$ |

Figure 14.3: Folded Reed-Solomon code for general $m \ge 1$

More formally, here is the definition of folded Reed-Solomon codes:

**Definition 14.1.1** (Folded Reed-Solomon Code)**.** The $m$-folded version of an $[n,k]_q$ Reed-Solomon code $C$ (with evaluation points $\{1,\gamma,\ldots,\gamma^{n-1}\}$), call it $C'$, is a code of block length $N = n/m$ over $\mathbb{F}_{q^m}$, where $n \le q-1$. The encoding of a message $f(X)$, a polynomial over $\mathbb{F}_q$ of degree at most $k-1$, has as its $j$'th symbol, for $0 \le j < n/m$, the $m$-tuple $\left(f\left(\gamma^{jm}\right), f\left(\gamma^{jm+1}\right), \cdots, f\left(\gamma^{jm+m-1}\right)\right)$. In other words, the codewords of $C'$ are in one-one correspondence with those of the Reed-Solomon code $C$ and are obtained by bundling together consecutive $m$-tuple of symbols in codewords of $C$.

### 14.1.1 The Intuition Behind Folded Reed-Solomon Codes

We first make the simple observation that the folding trick above cannot *decrease* the list decodability of the code. (We have already seen this argument earlier in Section 12.4.1.)

**Claim 14.1.1.** *If the Reed-Solomon code can be list-decoded from $\rho$ fraction of errors, then the corresponding folded Reed-Solomon code with folding parameter $m$ can also be list-decoded from $\rho$ fraction of errors.*

*Proof.* The idea is simple: If the Reed-Solomon code can be list decoded from $\rho$ fraction of errors (by say an algorithm $\mathscr{A}$), the Folded Reed-Solomon code can be list decoded by "unfolding" the received word and then running $\mathscr{A}$ on the unfolded received word and returning the resulting set of messages. Algorithm 16 has a more precise statement.

---

**Algorithm 16** Decoding Folded Reed-Solomon Codes by Unfolding

---

INPUT: $\mathbf{y} = ((y_{1,1}, \ldots, y_{1,m}), \ldots, (y_{n/m,1}, \ldots, y_{n/m,m})) \in \mathbb{F}_{q^m}^{n/m}$

OUTPUT: A list of messages in $\mathbb{F}_q^k$

1: $\mathbf{y}' \leftarrow (y_{1,1}, \ldots, y_{1,m}, \ldots, y_{n/m,1}, \ldots, y_{n/m,m}) \in \mathbb{F}_q^n$.
2: RETURN $\mathscr{A}(\mathbf{y}')$

---

The reason why Algorithm 16 works is simple. Let $\mathbf{m} \in \mathbb{F}_q^k$ be a message. Let $\mathrm{RS}(\mathbf{m})$ and $\mathrm{FRS}(\mathbf{m})$ be the corresponding Reed-Solomon and folded Reed-Solomon codewords. Now for every $i \in [n/m]$, if $\mathrm{FRS}(\mathbf{m})_i \neq (y_{i,1}, \ldots, y_{i,n/m})$ then in the worst-case for every $j \in [n/m]$, $\mathrm{RS}(\mathbf{m})_{(i-1)n/m+j} \neq y_{i,j}$: i.e. one symbol disagreement over $\mathbb{F}_{q^m}$ can lead to at most $m$ disagreements over $\mathbb{F}_q$. See Figure 14.4 for an illustration.

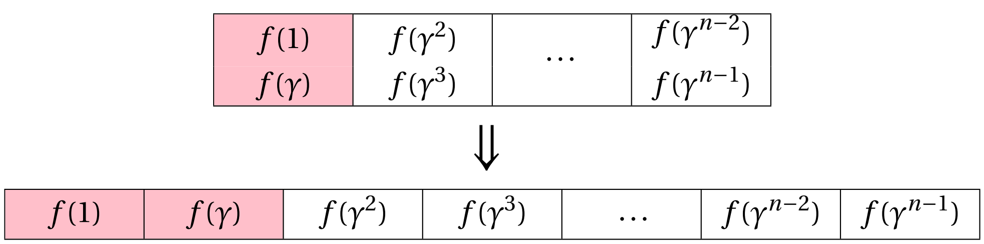


Figure 14.4: Error pattern after unfolding. A pink cell means an error: for the Reed-Solomon code it is for $\mathrm{RS}(\mathbf{m})$ with $\mathbf{y}'$ and for folded Reed-Solomon code it is for $\mathrm{FRS}(\mathbf{m})$ with $\mathbf{y}$

This implies that for any $\mathbf{m} \in \mathbb{F}_q^k$ if $\Delta(\mathbf{y}, \mathrm{FRS}(\mathbf{m})) \leq \rho \cdot \frac{n}{m}$, then $\Delta(\mathbf{y}', \mathrm{RS}(\mathbf{m})) \leq m \cdot \rho \cdot \frac{n}{m} = \rho \cdot n$, which by the properties of algorithm $\mathscr{A}$ implies that Step 2 will output $\mathbf{m}$, as desired. $\square$

The intuition for a *strict* improvement by using Folded Reed-Solomon codes is that if the fraction of errors due to folding increases beyond what it can list-decode from, that error pattern does not need to be handled and can be ignored. For example, suppose a Reed-Solomon

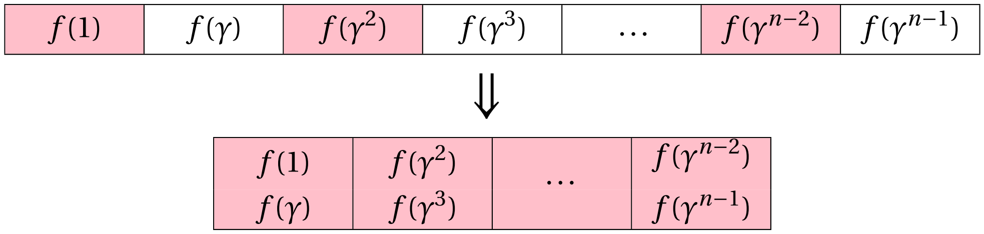


Figure 14.5: An error pattern after folding. The pink cells denotes the location of errors

code that can be list-decoded from up to $\frac{1}{2}$ fraction of errors is folded into a Folded Reed-Solomon code with $m = 2$. Now consider the error pattern in Figure 14.5.

The error pattern for Reed-Solomon code has $\frac{1}{2}$ fraction of errors, so any list decoding algorithm must be able to list-decode from this error pattern. However, for the Folded Reed-Solomon code the error pattern has 1 fraction of errors which is too high for the code to list-decode from. Thus, this “folded" error pattern case can be discarded from the ones that a list decoding algorithm for folded Reed-Solomon code needs to consider. This is of course one example– however, it turns out that this folding operation actually rules out *a lot* of error patterns that a list decoding algorithm for folded Reed-Solomon code needs to handle (even beyond the current best $1-\sqrt{R}$ bound for Reed-Solomon codes). Put another way, an algorithm for folded Reed-Solomon codes has to solve the list decoding problem for the Reed-Solomon codes where the error patterns are “bunched" together (technically they're called *bursty* errors). Of course, converting this intuition into a theorem takes more work and is the subject of this chapter.

**Wait a second...** The above argument has a potential hole– what if we take the argument to the extreme and "cheat" by setting $m = n$ where *any* error pattern for the Reed-Solomon code will result in an error pattern with 100% errors for the Folded Reed-Solomon code: thus, we will only need to solve the problem of error detection for Reed-Solomon codes (which we can easily solve for any linear code and in particular for Reed-Solomon codes)? It is a valid concern but we will “close the loophole" by only using a constant $m$ as the folding parameter. This will still keep $q$ to be polynomially large in $n$ and thus, we would still be on track to answer Question 14.0.1. Further, if we insist on smaller list size (e.g. one independent of $n$), then we can use code concatenation to achieve capacity achieving results for codes over smaller alphabets. (See Section 14.4 for more.)

**General Codes.** We would like to point out that the folding argument used above is not specific to Reed-Solomon codes. In particular, the argument for the reduction in the number of error patterns holds for any code. In fact, one can prove that for general random codes, with high probability, folding does strictly improve the list decoding capabilities of the original code. (The proof is left as an exercise.)

## 14.2 List Decoding Folded Reed-Solomon Codes: I

We begin with an algorithm for list decoding folded Reed-Solomon codes that works with agreement $t \sim mRN$. Note that this is a factor $m$ larger than the $RN$ agreement we ultimately want. In the next section, we will see how to knock off the factor of $m$.

Before we state the algorithm, we formally (re)state the problem we want to solve:

- **Input**: An agreement parameter $0 \le t \le N$ and the received word:

$$\mathbf{y} = \begin{pmatrix} y_0 & y_m & & y_{n-m} \\ \vdots & \vdots & \cdots & \vdots \\ y_{m-1} & y_{2m-1} & & y_{n-1} \end{pmatrix} \in \mathbb{F}_q^{m \times N}, \quad N = \frac{n}{m}$$

- **Output**: Return all polynomials $f(X) \in \mathbb{F}_q[X]$ of degree at most $k-1$ such that for at least $t$ values of $0 \le i < N$

$$\begin{pmatrix} f\left(\gamma^{mi}\right) \\ \vdots \\ f\left(\gamma^{m(i+1)-1}\right) \end{pmatrix} = \begin{pmatrix} y_{mi} \\ \vdots \\ y_{m(i+1)-1} \end{pmatrix} \tag{14.1}$$

The algorithm that we will study is a generalization of the Welch-Berlekamp algorithm (Algorithm 12). However unlike the previous list decoding algorithms for Reed-Solomon codes (Algorithms 13, 14 and 15), this new algorithm has more similarities with the Welch-Berlekamp algorithm. In particular, for $m = 1$, the new algorithm is *exactly* the Welch-Berlekamp algorithm. Here are the new ideas in the algorithm for the two-step framework that we have seen in the previous chapter:

- **Step 1**: We interpolate using $(m+1)$-variate polynomial, $Q(X, Y_1, \ldots, Y_m)$, where degree of each variable $Y_i$ is exactly one. In particular, for $m = 1$, this interpolation polynomial is exactly the one used in the Welch-Berlekamp algorithm.

- **Step 2**: As we have done so far, in this step, we output all "roots" of $Q$. Two remarks are in order. First, unlike Algorithms 13, 14 and 15, the roots $f(X)$ are no longer simpler linear factors $Y - f(X)$, so one cannot use a factorization algorithm to factorize $Q(X, Y_1, \ldots, Y_m)$. Second, the new insight in this algorithm, is to show that all the roots form an (affine) subspace,[1] which we can use to compute the roots.

Algorithm 17 has the details.

[1] An affine subspace of $\mathbb{F}_q^k$ is a set $\{\mathbf{v} + \mathbf{u} | \mathbf{u} \in S\}$, where $S \subseteq \mathbb{F}_q^k$ is a linear subspace and $\mathbf{v} \in \mathbb{F}_q^k$.

**Algorithm 17** The First List Decoding Algorithm for Folded Reed-Solomon Codes

INPUT: An agreement parameter $0 \le t \le N$, parameter $D \ge 1$ and the received word:

$$\mathbf{y} = \begin{pmatrix} y_0 & y_m & & y_{n-m} \\ \vdots & \vdots & \cdots & \vdots \\ y_{m-1} & y_{2m-1} & & y_{n-1} \end{pmatrix} \in \mathbb{F}_q^{m \times N}, \quad N = \frac{n}{m}$$

OUTPUT: All polynomials $f(X) \in \mathbb{F}_q[X]$ of degree at most $k-1$ such that for at least $t$ values of $0 \le i < N$

$$\begin{pmatrix} f\left(\gamma^{mi}\right) \\ \vdots \\ f\left(\gamma^{m(i+1)-1}\right) \end{pmatrix} = \begin{pmatrix} y_{mi} \\ \vdots \\ y_{m(i+1)-1} \end{pmatrix} \tag{14.2}$$

1: Compute a non-zero $Q(X, Y_1, \ldots, Y_m)$ where

$$Q(X, Y_1, \ldots, Y_m) = A_0(X) + A_1(X)Y_1 + A_2(X)Y_2 + \cdots + A_m(X)Y_m$$

with $\deg(A_0) \le D + k - 1$ and $\deg(A_j) \le D$ for $1 \le j \le m$ such that

$$Q(\gamma^{mi}, y_{mi}, \cdots, y_{m(i+1)-1}) = 0, \ \ \forall 0 \le i < N \tag{14.3}$$

2: Ł $\leftarrow \emptyset$
3: FOR every $f(X) \in \mathbb{F}_q[X]$ such that $Q(X, f(X), f(\gamma X), f(\gamma^2 X), \ldots, f(\gamma^{m-1}X)) = 0$ DO
4:     IF $\deg(f) \le k-1$ and $f(X)$ satisfies (14.2) for at least $t$ values of $i$ THEN
5:         Add $f(X)$ to Ł.
6: RETURN Ł

**Correctness of Algorithm 17.** In this section, we will only concentrate on the correctness of the algorithm and analyze its error correction capabilities. We will defer the analysis of the algorithm (and in particular, proving a bound on the number of polynomials that are output by Step 6) till the next section.

We first begin with the claim that there always exists a non-zero choice for $Q$ in Step 1 using the same arguments that we have used to prove the correctness of Algorithms 14 and 15:

**Claim 14.2.1.** *If $(m+1)(D+1) + k - 1 > N$, then there exists a non-zero $Q(X, Y_1, \ldots Y_m)$ that satisfies the required properties of Step* 1.

*Proof.* As in the proof of correctness of Algorithms 13, 14 and 15, we will think of the constraints in (14.3) as linear equations. The variables are the coefficients of $A_i(X)$ for $0 \le i \le m$. With the stipulated degree constraints on the $A_i(X)$'s, note that the number of variables participating in (14.3) is

$$D + k + m(D+1) = (m+1)(D+1) + k - 1.$$

The number of equations is $N$. Thus, the condition in the claim implies that we have strictly more variables then equations and thus, there exists a non-zero $Q$ with the required properties. $\square$

Next, we argue that the root finding step works (again using an argument very similar to those that we have seen for Algorithms 13, 14 and 15):

**Claim 14.2.2.** *If $t > D+k-1$, then all polynomial $f(X) \in \mathbb{F}_q[X]$ of degree at most $k-1$ that agree with the received word in at least $t$ positions is returned by Step* 6.

*Proof.* Define the univariate polynomial

$$R(X) = Q\left(X, f(X), f(\gamma X), \ldots\ldots f\left(\gamma^{m-1}X\right)\right).$$

Note that due to the degree constraints on the $A_i(X)$'s and $f(X)$, we have

$$\deg(R) \le D + k - 1,$$

since $\deg(f(\gamma^i X)) = \deg(f(X))$. On the other hand, for every $0 \le i < N$ where (14.1) is satisfied we have

$$R\left(\gamma^{mi}\right) = Q\left(\gamma^{mi}, y_{mi}, \ldots, y_{m(i+1)-1}\right) = 0,$$

where the first equality follows from (14.1), while the second equality follows from (14.3). Thus $R(X)$ has at least $t$ roots. Thus, the condition in the claim implies that $R(X)$ has more roots then its degree and thus, by the degree mantra (Proposition 5.2.3) $R(X) \equiv 0$, as desired. $\square$

Note that Claims 14.2.1 and 14.2.2 prove the correctness of the algorithm. Next we analyze the fraction of errors the algorithm can correct. Note that the condition in Claim 14.2.1 is satisfied if we pick

$$D = \left\lfloor \frac{N-k+1}{m+1} \right\rfloor.$$

This in turn implies that the condition in Claim 14.2.2 is satisfied if

$$t > \frac{N-k+1}{m+1} + k - 1 = \frac{N + m(k-1)}{m+1}.$$

Thus, the above would be satisfied if

$$t \ge \frac{N}{m+1} + \frac{mk}{m+1} = N\left(\frac{1}{m+1} + mR\left(\frac{m}{m+1}\right)\right),$$

where the equality follows from the fact that $k = mRN$.

Note that when $m = 1$, the above bound exactly recovers the bound for the Welch-Berlekamp algorithm (Theorem 13.1.4). Thus, we have shown that

**Theorem 14.2.3.** *Algorithm 17 can list decode folded Reed-Solomon code with folding parameter $m \ge 1$ and rate $R$ up to $\frac{m}{m+1}(1 - mR)$ fraction of errors.*

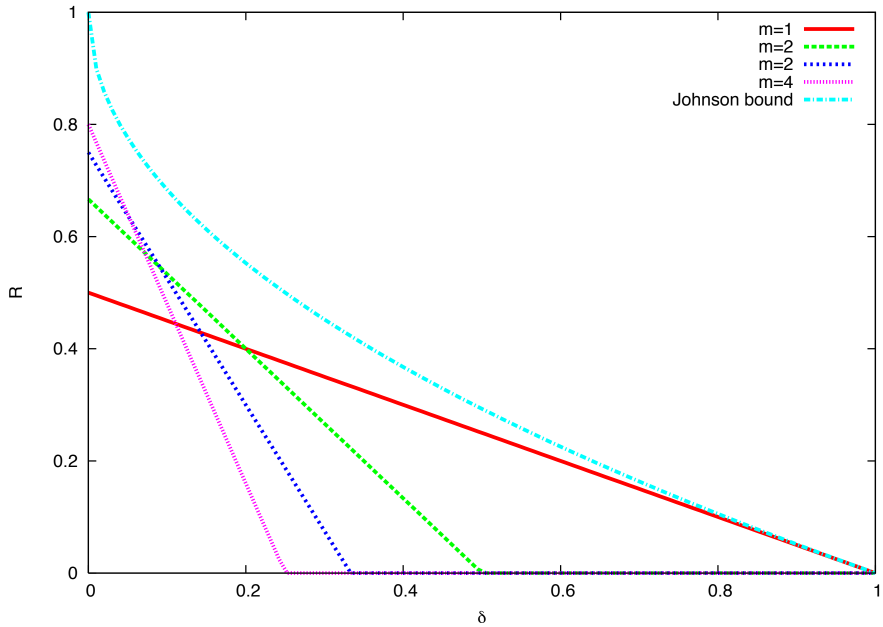


Figure 14.6: The tradeoff between rate $R$ and the fraction of errors that can be corrected by Algorithm 17 for folding parameter $m = 1, 2, 3$ and 4. The Johnson bound is also plotted for comparison. Also note that the bound for $m = 1$ is the Unique decoding bound achieved by Algorithm 12.

See Figure 14.2 for an illustration of the tradeoff for $m = 1, 2, 3, 4$.

Note that if we can replace the $mR$ factor in the bound from Theorem 14.2.3 by just $R$ then we can approach the list decoding capacity bound of $1 - R$. (In particular, we would be able to correct $1 - R - \varepsilon$ fraction of errors if we pick $m = O(1/\varepsilon)$.) Further, we need to analyze the number of polynomials output by the root finding step of the algorithm (and then analyze the runtime of the algorithm). In the next section, we show how we can "knock-off" the extra factor $m$ (and we will also bound the list size).

## 14.3 List Decoding Folded Reed-Solomon Codes: II

In this section, we will present the final version of the algorithm that will allow us to answer Question 14.0.1 in the affirmative. We start off with the new idea that allows us to knock off the factor of $m$. (It would be helpful to keep the proof of Claim 14.2.2 in mind.)

To illustrate the idea let us consider the folding parameter to be $m = 3$. Let $f(X)$ be a polynomial of degree at most $k - 1$ that needs to be output and let $0 \le i < N$ be a position where it agrees with the received word. (See Figure 14.7 for an illustration.)

The idea is to "exploit" this agreement over *one* $\mathbb{F}_q^3$ symbol and convert it into *two* agreements over $\mathbb{F}_{q^2}$. (See Figure 14.8 for an illustration.)

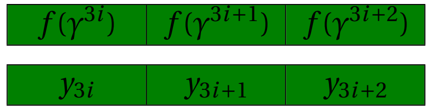


Figure 14.7: An agreement in position $i$.


Figure 14.8: More agreement with a sliding window of size 2.

Thus, in the proof of Claim 14.2.2, for each agreement we can now get *two* roots for the polynomial $R(X)$. In general for an agreement over one $\mathbb{F}_{q^m}$ symbols translates into $m-s+1$ agreement over $\mathbb{F}_q^s$ for any $1 \le s \le m$ (by "sliding a window" of size $s$ over the $m$ symbols from $\mathbb{F}_q$). Thus, in this new idea the agreement is $m-s+1$ times more than before which leads to the $mR$ term in Theorem 14.2.3 going down to $\frac{mR}{m-s+1}$. Then making $s$ smaller than $m$ but still large enough we can get down the relative agreement to $R+\varepsilon$, as desired. There is another change that needs to be done to make the argument go through: the interpolation polynomial $Q$ now has to be $(s+1)$-variate instead of the earlier $(m+1)$-variate polynomial. Algorithm 18 has the details.

**Correctness of Algorithm 18.** Next, we analyze the correctness of Algorithm 18 as well as compute its list decoding error bound. We begin with the result showing that there exists a $Q$ with the required properties for Step 1.

**Lemma 14.3.1.** *If* $D \ge \left\lfloor \frac{N(m-s+1)-k+1}{s+1} \right\rfloor$, *then there exists a non-zero polynomial* $Q(X, Y_1, ..., Y_s)$ *that satisfies* Step 1 *of the above algorithm.*

*Proof.* Let us consider all coefficients of all polynomials $A_i$ as variables. Then the number of variables will be

$$D + k + s(D+1) = (s+1)(D+1) + k - 1.$$

On the other hand, the number of constraints in (14.5), i.e. the number of equations when all coefficients of all polynomials $A_i$ are considered variables) will be $N(m-s+1)$.

Note that if we have more variables than equations, then there exists a non-zero $Q$ that satisfies the required properties of Step 1. Thus, we would be done if we have:

$$(s+1)(D+1) + k - 1 > N(m-s+1),$$

which is equivalent to:

$$D > \frac{N(m-s+1)-k+1}{s+1} - 1.$$

The choice of $D$ in the statement of the claim satisfies the condition above, which complete the proof. □

**Algorithm 18** The Second List Decoding Algorithm for Folded Reed-Solomon Codes

INPUT: An agreement parameter $0 \le t \le N$, parameter $D \ge 1$ and the received word:

$$\mathbf{y} = \begin{pmatrix} y_0 & y_m & & y_{n-m} \\ \vdots & \vdots & \cdots & \vdots \\ y_{m-1} & y_{2m-1} & & y_{n-1} \end{pmatrix} \in \mathbb{F}_q^{m \times N}, \quad N = \frac{n}{m}$$

OUTPUT: All polynomials $f(X) \in \mathbb{F}_q[X]$ of degree at most $k-1$ such that for at least $t$ values of $0 \le i < N$

$$\begin{pmatrix} f\left(\gamma^{mi}\right) \\ \vdots \\ f\left(\gamma^{m(i+1)-1}\right) \end{pmatrix} = \begin{pmatrix} y_{mi} \\ \vdots \\ y_{m(i+1)-1} \end{pmatrix} \tag{14.4}$$

1: Compute non-zero polynomial $Q(X, Y_1, .., Y_s)$ as follows:

$$Q(X, Y_1, .., Y_s) = A_0(X) + A_1(X)Y_1 + A_2(X)Y_2 + .. + A_s(X)Y_s,$$

with $\deg[A_0] \le D + k - 1$ and $\deg[A_i] \le D$ for every $1 \le i \le s$ such that for all $0 \le i < N$ and $0 \le j \le m - s$, we have

$$Q(\gamma^{im+j}, y_{im+j}, ..., y_{im+j+s-1}) = 0. \tag{14.5}$$

2: Ł $\leftarrow \emptyset$

3: FOR every $f(X) \in \mathbb{F}_q[X]$ such that

$$Q\left(X, f(X), f\left(\gamma X\right), f\left(\gamma^2 X\right), \ldots, f\left(\gamma^{s-1} X\right)\right) \equiv 0 \tag{14.6}$$

DO

4: IF $\deg(f) \le k-1$ and $f(X)$ satisfies (14.2) for at least $t$ values of $i$ THEN

5: Add $f(X)$ to Ł.

6: RETURN Ł

Next we argue that the root finding step works.

**Lemma 14.3.2.** *If $t > \frac{D+k-1}{m-s+1}$, then every polynomial $f(X)$ that needs to be output satisfies (14.6).*

*Proof.* Consider the polynomial $R(X) = Q\left(X, f(X), f\left(\gamma X\right), ..., f\left(\gamma^{s-1} X\right)\right)$. Because the degree of $f\left(\gamma^{\ell} X\right)$ (for every $0 \le \ell \le s-1$) is at most $k-1$,

$$\deg(R) \le D + k - 1. \tag{14.7}$$

Let f(X) be one of the polynomials of degree at most $k-1$ that needs to be output, and $f(X)$ agrees with the received word at column $i$ for some $0 \le i < N$, that is:

$$\begin{pmatrix} f\left(\gamma^{mi}\right) \\ f\left(\gamma^{mi+1}\right) \\ \cdot \\ \cdot \\ \cdot \\ f\left(\gamma^{m(i+1)-1}\right) \end{pmatrix} = \begin{pmatrix} y_{mi} \\ y_{mi+1} \\ \cdot \\ \cdot \\ \cdot \\ y_{m(i+1)-1} \end{pmatrix},$$

then for all $0 \le j \le m-s$, we have:

$$\begin{aligned} R\left(\gamma^{mi+j}\right) &= Q\left(\gamma^{mi+j}, f\left(\gamma^{mi+j}\right), f\left(\gamma^{mi+1+j}\right), \dots, f\left(\gamma^{mi+s-1+j}\right)\right) \\ &= Q\left(\gamma^{mi+j}, y_{mi+j}, y_{mi+1+j}, \dots, y_{mi+s-1+j}\right) = 0. \end{aligned}$$

In the above, the first equality follows as $f(X)$ agrees with $\mathbf{y}$ in column $i$ while the second equality follows from (14.5). Thus, the number of roots of $R(X)$ is at least

$$t(m-s+1) > D+k-1 \ge \deg(R),$$

where the first inequality follows from the assumption in the claim and the second inequality follows from (14.7). Hence, by the degree mantra $R(X) \equiv 0$, which shows that $f(X)$ satisfies (14.6), as desired. $\square$

### 14.3.1 Error Correction Capability

Now we analyze the the fraction of errors the algorithm above can handle. (We will come back to the thorny issue of proving a bound on the output list size for the root finding step in Section 14.3.2.)

The argument for the fraction of errors follows the by now standard route. To satisfy the constraint in Lemma 14.3.1, we pick

$$D = \left\lfloor \frac{N(m-s+1)-k+1}{s+1} \right\rfloor.$$

This along with the constraint in Lemma 14.3.2, implies that the algorithm works as long as

$$t > \left\lfloor \frac{D+k-1}{m-s+1} \right\rfloor.$$

The above is satisfied if we choose

$$t > \frac{\frac{N(m-s+1)-k+1}{s+1}+k-1}{m-s+1} = \frac{N(m-s+1)-k+1+(k-1)(s+1)}{(m-s+1)(s+1)} = \frac{N(m-s+1)+s(k-1)}{(s+1)(m-s+1)}.$$

Thus, we would be fine if we pick

$$t > \frac{N}{s+1} + \frac{s}{s+1} \cdot \frac{k}{m-s+1} = N\left(\frac{1}{s+1} + \left(\frac{s}{s+1}\right)\left(\frac{m}{m-s+1}\right) \cdot R\right),$$

where the equality follows from the fact that $k = mRN$. This implies the following result:

**Theorem 14.3.3.** *Algorithm 18 can list decode folded Reed-Solomon code with folding parameter $m \geq 1$ and rate $R$ up to $\frac{s}{s+1}(1 - mR/(m-s+1))$ fraction of errors.*

See Figure 14.3.1 for an illustration of the bound above.

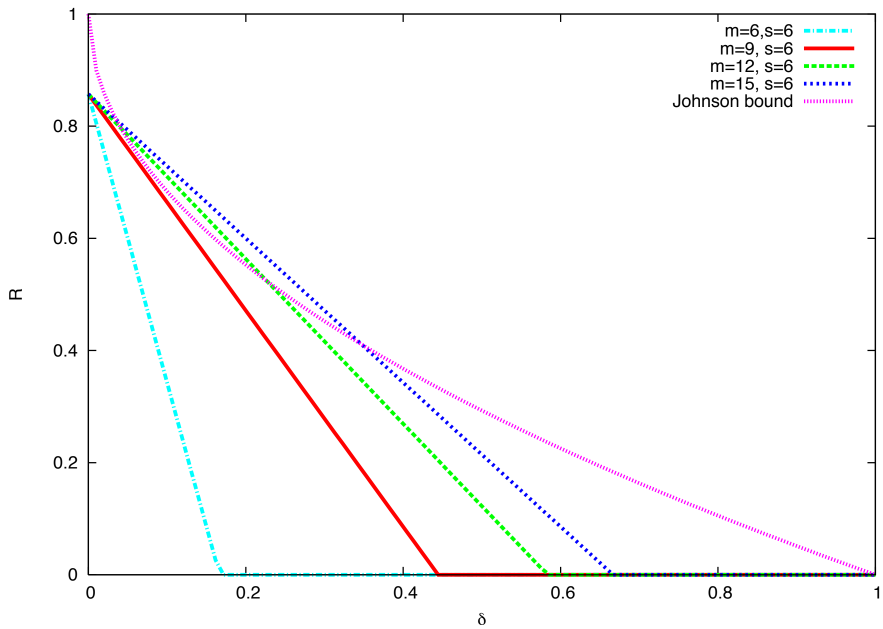


Figure 14.9: The tradeoff between rate $R$ and the fraction of errors that can be corrected by Algorithm 18 for $s = 6$ and folding parameter $m = 6, 9, 12$ and 15. The Johnson bound is also plotted for comparison.

### 14.3.2 Bounding the Output List Size

We finally address the question of bounding the output list size in the root finding step of the algorithm. We will present a proof that will immediately lead to an algorithm to implement the root finding step. We will show that there are at most $q^{s-1}$ possible solutions for the root finding step.

The main idea is the following: think of the coefficients of the output polynomial $f(X)$ as variables. Then the constraint (14.6) implies $D+k$ linear equations on these $k$ variables. It turns out that if one picks only $k$ out of these $D+k$ constraints, then the corresponding constraint matrix has rank at least $k-s+1$, which leads to the claimed bound. Finally, the claim on the rank of the constraint matrix follows by observing (and this is the crucial insight) that the constraint matrix is upper triangular. Further, the diagonal elements are evaluation of a non-zero polynomial of degree at most $s-1$ in $k$ distinct elements. By the degree mantra (Proposition 5.2.3), this polynomial can have at most $s-1$ roots, which implies that at least $k-s+1$ elements of the

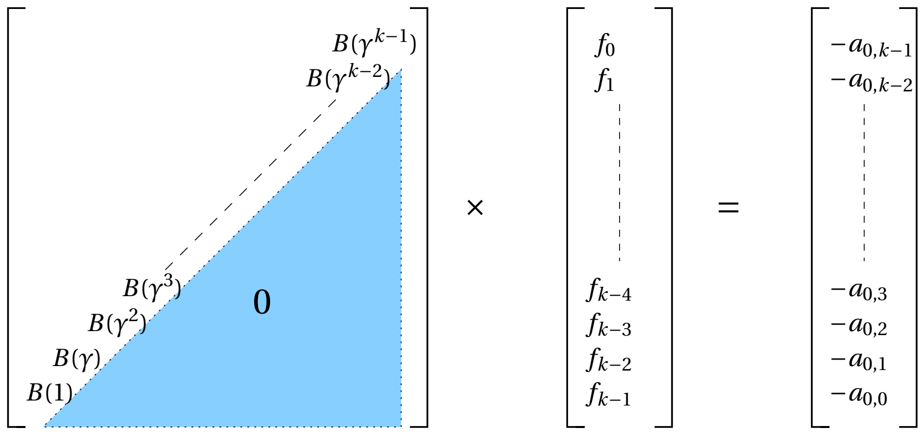


Figure 14.10: The system of linear equations with the variables $f_0, \ldots, f_{k-1}$ forming the coefficients of the polynomial $f(X) = \sum_{i=0}^{k-1} f_i X^i$ that we want to output. The constants $a_{j,0}$ are obtained from the interpolating polynomial from Step 1. $B(X)$ is a non-zero polynomial of degree at most $s-1$.

diagonal are non-zero, which then implies the claim. See Figure 14.10 for an illustration of the upper triangular system of linear equations.

Next, we present the argument above in full detail. (Note that the constraint on (14.8) is the same as the one in (14.6) because of the constraint on the structure of $Q$ imposed by Step 1.)

**Lemma 14.3.4.** *There are at most $q^{s-1}$ solutions to $f_0, f_1, .., f_{k-1}$ (where $f(X) = f_0 + f_1 X + ... + f_{k-1} X^{k-1}$) to the equations*

$$A_0(X) + A_1(X) f(X) + A_2(X) f\left(\gamma X\right) + ... + A_s(X) f\left(\gamma^{s-1} X\right) \equiv 0 \tag{14.8}$$

*Proof.* First we assume that X does not divide all of the polynomials $A_0, A_1, ..., A_s$. Then it implies that there exists $i^* > 0$ such that the constant term of the polynomial $A_{i^*}(X)$ is not zero. (Because otherwise, since $X | A_1(X), ..., A_s(X)$, by (14.8), we have $X$ divides $A_0(X)$ and hence $X$ divide all the $A_i(X)$ polynomials, which contradicts the assumption.)

To facilitate the proof, we define few auxiliary variables $a_{ij}$ such that

$$A_i(X) = \sum_{j=0}^{D+k-1} a_{ij} X^j \text{ for every } 0 \le i \le s,$$

and define the following univariate polynomial:

$$B(X) = a_{1,0} + a_{2,0} X + a_{3,0} X^2 + ... + a_{s,0} X^{s-1}. \tag{14.9}$$

Notice that $a_{i^*,0} \neq 0$, so $B(X)$ is non-zero polynomial. And because degree of $B(X)$ is at most $s-1$, by the degree mantra (Proposition 5.2.3), $B(X)$ has at most $s-1$ roots. Next, we claim the following:

**Claim 14.3.5.** *For every* $0 \le j \le k-1$*:*

- *If* $B(\gamma^j) \neq 0$*, then* $f_j$ *is uniquely determined by* $f_{j-1}, f_{j-2}, \ldots, f_0$.
- *If* $B(\gamma^j) = 0$*, then* $f_j$ *is unconstrained, i.e.* $f_j$ *can take any of the* $q$ *values in* $\mathbb{F}_q$.

We defer the proof of the claim above for now. Suppose that the above claim is correct. Then as $\gamma$ is a generator of $\mathbb{F}_q$, $1, \gamma, \gamma^2, ..., \gamma^{k-1}$ are distinct (since $k-1 \le q-2$). Further, by the degree mantra (Proposition 5.2.3) at most $s-1$ of these elements are roots of the polynomial $B(X)$. Therefore by Claim 14.3.5, the number of solutions to $f_0, f_1, ..., f_{k-1}$ is at most $q^{s-1}$. [2]

We are almost done except we need to remove our earlier assumption that $X$ does not divide every $A_i$. Towards this end, we essentially just factor out the largest common power of $X$ from all of the $A_i$'s, and proceed with the reduced polynomial. Let $l \ge 0$ be the largest $l$ such that $A_i(X) = X^l A'_i(X)$ for $0 \le i \le s$; then $X$ does not divide all of $A'_i(X)$ and we have:

$$X^l \left(A'_0(X) + A'_1(X) f(X) + \cdots + A'_s(X) f(\gamma^{s-1} X)\right) \equiv 0.$$

Thus we can do the entire argument above by replacing $A_i(X)$ with $A'_i(X)$ since the above constraint implies that $A'_i(X)$'s also satisfy (14.8). □

Next we prove Claim 14.3.5.

*Proof of Claim 14.3.5.* Recall that we can assume that $X$ does not divide all of $\{A_0(X), \ldots, A_s(X)\}$.

Let $C(X) = A_0(X) + A_1(X) f(X) + \cdots + A_s f\left(\gamma^{s-1} X\right)$. Recall that we have $C(X) \equiv 0$. If we expand out each polynomial multiplication, we have:

$$\begin{aligned} C(X) = {} & a_{0,0} + a_{0,1} X + \cdots + a_{0,D+k-1} X^{D+k-1} \\ & + \left(a_{1,0} + a_{1,1} X + \cdots + a_{1,D+k-1} X^{D+k-1}\right)\left(f_0 + f_1 X + f_2 X^2 + \cdots + f_{k-1} X^{k-1}\right) \\ & + \left(a_{2,0} + a_{2,1} X + \cdots + a_{2,D+k-1} X^{D+k-1}\right)\left(f_0 + f_1 \gamma X + f_2 \gamma^2 X^2 + \cdots + f_{k-1} \gamma^{k-1} X^{k-1}\right) \\ & \vdots \\ & + \left(a_{s,0} + a_{s,1} X + \cdots + a_{s,D+k-1} X^{D+k-1}\right)\left(f_0 + f_1 \gamma^{s-1} X + f_2 \gamma^{2(s-1)} X^2 + \cdots + f_{k-1} \gamma^{(k-1)(s-1)} X^{k-1}\right) \end{aligned} \tag{14.10}$$

Now if we collect terms of the same degree, we will have a polynomial of the form:

$$C(X) = c_0 + c_1 X + c_2 X^2 + \cdots + c_{D+k-1} X^{D+k-1}.$$

[2] Build a "decision tree" with $f_0$ as the root and $f_j$ in the $j$th level: each edge is labeled by the assigned value to the parent node variable. For any internal node in the $j$th level, if $B(\gamma^j) \neq 0$, then the node has a single child with the edge taking the unique value promised by Claim 14.3.5. Otherwise the node has $q$ children with $q$ different labels from $\mathbb{F}_q$. By Claim 14.3.5, the number of solutions to $f(X)$ is upper bounded by the number of nodes in the $k$th level in the decision tree, which by the fact that $B$ has at most $s-1$ roots is upper bounded by $q^{s-1}$.

So we have $D+k$ linear equations in variables $f_0, \dots, f_{k-1}$, and we are seeking those solutions such that $c_j = 0$ for every $0 \le j \le D+k-1$. We will only consider the $0 \le j \le k-1$ equations. We first look at the equation for $j=0$: $c_0 = 0$. This implies the following equalities:

$$0 = a_{0,0} + f_0 a_{1,0} + f_0 a_{2,0} + \cdots + f_0 a_{s,0} \tag{14.11}$$

$$0 = a_{0,0} + f_0 \left( a_{1,0} + a_{2,0} + \cdots + a_{s,0} \right) \tag{14.12}$$

$$0 = a_{0,0} + f_0 B(1). \tag{14.13}$$

In the above (14.11) follows from (14.10), (14.12) follows by simple manipulation while (14.13) follows from the definition of $B(X)$ in (14.9).

Now, we have two possible cases:

- **Case 1:** $B(1) \neq 0$. In this case, (14.13) implies that $f_0 = \frac{-a_{0,0}}{B(1)}$. In particular, $f_0$ is fixed.
- **Case 2:** $B(1) = 0$. In this case $f_0$ has no constraint (and hence can take on any of the $q$ values in $\mathbb{F}_q$).

Now consider the equation for $j=1$: $c_1 = 0$. Using the same argument as we did for $j=0$, we obtain the following sequence of equalities:

$$0 = a_{0,1} + f_1 a_{1,0} + f_0 a_{1,1} + f_1 a_{2,0}\gamma + f_0 a_{2,1} + \cdots + f_1 a_{s,0}\gamma^{s-1} + f_0 a_{s,1}$$

$$0 = a_{0,1} + f_1 \left( a_{1,0} + a_{2,0}\gamma + \cdots + a_{s,0}\gamma^{s-1} \right) + f_0 \left( \sum_{l=1}^{s} a_{l,1} \right)$$

$$0 = a_{0,1} + f_1 B(\gamma) + f_0 b_0^{(1)} \tag{14.14}$$

where $b_0^{(1)} = \sum_{l=1}^{s} a_{l,1}$ is a constant. We have two possible cases:

- **Case 1:** $B(\gamma) \neq 0$. In this case, by (14.14), we have $f_1 = \frac{-a_{0,1} - f_0 b_0^{(1)}}{B(\gamma)}$ and there is a unique choice for $f_1$ given fixed $f_0$.
- **Case 2:** $B(\gamma) = 0$. In this case, $f_1$ is unconstrained.

Now consider the case of arbitrary $j$: $c_j = 0$. Again using similar arguments as above, we get:

$$\begin{aligned}
0 = a_{0,j} &+ f_j(a_{1,0} + a_{2,0}\gamma^j + a_{3,0}\gamma^{2j} + \cdots + a_{s,0}\gamma^{j(s-1)}) \\
&+ f_{j-1}(a_{1,1} + a_{2,1}\gamma^{j-1} + a_{3,1}\gamma^{2(j-1)} + \cdots + a_{s,1}\gamma^{(j-1)(s-1)}) \\
&\vdots \\
&+ f_1(a_{1,j-1} + a_{2,j-1}\gamma + a_{3,j-1}\gamma^2 + \cdots + a_{s,j-1}\gamma^{s-1}) \\
&+ f_0(a_{1,j} + a_{2,j} + a_{3,j} + \cdots + a_{s,j})
\end{aligned}$$

$$0 = a_{0,j} + f_j B(\gamma^j) + \sum_{l=0}^{j-1} f_l b_l^{(j)} \tag{14.15}$$

where $b_l^{(j)} = \sum_{\iota=1}^{s} a_{\iota,j-l} \cdot \gamma^{l(\iota-1)}$ are constants for $0 \le j \le k-1$.

We have two possible cases:

- **Case 1:** $B(\gamma^j) \neq 0$. In this case, by (14.15), we have

$$f_j = \frac{-a_{0,j} - \sum_{l=0}^{j-1} f_l b_l^{(j)}}{B(\gamma^j)} \tag{14.16}$$

  and there is a unique choice for $f_j$ given fixed $f_0, \ldots, f_{j-1}$.

- **Case 2:** $B(\gamma^j) = 0$. In this case $f_j$ is unconstrained.

This completes the proof. □

We now revisit the proof above and make some algorithmic observations. First, we note that to compute all the tuples $(f_0, \ldots, f_{k-1})$ that satisfy (14.8) one needs to solve the linear equations (14.15) for $j = 0, \ldots, k-1$. One can state this system of linear equation as (see also Figure 14.10)

$$C \cdot \begin{pmatrix} f_0 \\ \vdots \\ f_{k-1} \end{pmatrix} = \begin{pmatrix} -a_{0,k-1} \\ \vdots \\ -a_{0,0} \end{pmatrix},$$

where $C$ is a $k \times k$ upper triangular matrix. Further each entry in $C$ is either a 0 or $B(\gamma^j)$ or $b_l^{(j)}$– each of which can be computed in $O(s \log s)$ operations over $\mathbb{F}_q$. Thus, we can setup this system of equations in $O\left(s \log s k^2\right)$ operations over $\mathbb{F}_q$.

Next, we make the observation that all the solutions to (14.8) form an affine subspace. Let $0 \leq d \leq s-1$ denote the number of roots of $B(X)$ in $\{1, \gamma, \ldots, \gamma^{k-1}\}$. Then since there will be $d$ unconstrained variables among $f_0, \ldots, f_{k-1}$ (one of every $j$ such that $B(\gamma^j) = 0$), it is not too hard to see that all the solutions will be in the set $\left\{M \cdot \mathbf{x} + \mathbf{z} | \mathbf{x} \in \mathbb{F}_q^d\right\}$, for some $k \times d$ matrix $M$ and some $\mathbf{z} \in \mathbb{F}_q^k$. Indeed every $\mathbf{x} \in \mathbb{F}_q^d$ corresponds to an assignment to the $d$ unconstrained variables among $f_0, \ldots, f_j$. The matrix $M$ and the vector $\mathbf{z}$ are determined by the equations in (14.16). Further, since $C$ is upper triangular, both $M$ and $\mathbf{z}$ can be computed with $O\left(k^2\right)$ operations over $\mathbb{F}_q$.

The discussion above implies the following:

**Corollary 14.3.6.** *The set of solutions to (14.8) are contained in an affine subspace* $\left\{M \cdot \mathbf{x} + \mathbf{z} | \mathbf{x} \in \mathbb{F}_q^d\right\}$ *for some* $0 \leq d \leq s-1$ *and* $M \in \mathbb{F}_q^{k \times d}$ *and* $\mathbf{z} \in \mathbb{F}_q^k$. *Further,* $M$ *and* $\mathbf{z}$ *can be computed from the polynomials* $A_0(X), \ldots, A_s(X)$ *with* $O(s \log s k^2)$ *operations over* $\mathbb{F}_q$.

### 14.3.3 Algorithm Implementation and Runtime Analysis

In this sub-section, we discuss how both the interpolation and root finding steps of the algorithm can be implemented and analyze the run time of each step.

Step 1 involves solving $Nm$ linear equation in $O(Nm)$ variables and can e.g. be solved by Gaussian elimination in $O((Nm)^3)$ operations over $\mathbb{F}_q$. This is similar to what we have seen for Algorithms 13, 14 and 15. However, the fact that the interpolation polynomial has total degree of one in the variables $Y_1, \ldots, Y_s$ implies a much faster algorithm. In particular, one can perform the interpolation in $O(Nm \log^2(Nm) \log\log(Nm))$ operations over $\mathbb{F}_q$.

The root finding step involves computing all the "roots" of $Q$. The proof of Lemma 14.3.4 actually suggests Algorithm 19.

**Algorithm 19** The Root Finding Algorithm for Algorithm 18

INPUT: $A_0(X), \ldots, A_s(X)$

OUTPUT: All polynomials $f(X)$ of degree at most $k-1$ that satisfy (14.8)

1: Compute $\ell$ such that $X^\ell$ is the largest common power of $X$ among $A_0(X), \ldots, A_s(X)$.
2: FOR every $0 \le i \le s$ DO
3: $\quad A_i(X) \leftarrow \frac{A_i(X)}{X^\ell}$.
4: Compute $B(X)$ according to (14.9)
5: Compute $d$, $\mathbf{z}$ and $M$ such that the solutions to the $k$ linear system of equations in (14.15) lie in the set $\left\{M \cdot \mathbf{x} + \mathbf{z} | \mathbf{x} \in \mathbb{F}_q^d\right\}$.
6: $Ł \leftarrow \emptyset$
7: FOR every $\mathbf{x} \in \mathbb{F}_q^d$ DO
8: $\quad (f_0, \ldots, f_{k-1}) \leftarrow M \cdot \mathbf{x} + \mathbf{z}$.
9: $\quad f(X) \leftarrow \sum_{i=0}^{k-1} f_i \cdot X^i$.
10: $\quad$ IF $f(X)$ satisfies (14.8) THEN
11: $\quad\quad$ Add $f(X)$ to Ł.
12: RETURN Ł

Next, we analyze the run time of the algorithm. Throughout, we will assume that all polynomials are represented in their standard coefficient form.

Step 1 just involves figuring out the smallest power of $X$ in each $A_i(X)$ that has a non-zero coefficient from which one can compute the value of $\ell$. This can be done with $O(D + k + s(D + 1)) = O(Nm)$ operations over $\mathbb{F}_q$. Further, given the value of $\ell$ one just needs to "shift" all the coefficients in each of the $A_i(X)$'s to the right by $\ell$, which again can be done with $O(Nm)$ operations over $\mathbb{F}_q$.

Now we move to the root finding step. The run time actually depends on what it means to "solve" the linear system. If one is happy with a succinct description of a set of possible solution that contains the actual output then one can halt Algorithm 19 after Step 5 and Corollary 14.3.6 implies that this step can be implemented in $O\left(s \log s k^2\right) = O\left(s \log s (Nm)^2\right)$ operations over $\mathbb{F}_q$. However, if one wants the actual set of polynomials that need to be output, then the only known option so far is to check all the $q^{s-1}$ potential solutions as in Steps 7-11. (However, we'll see a twist in Section 14.4.) The latter would imply a total of $O(s \log s (Nm)^2) + O(q^{s-1} \cdot (Nm)^2)$ operations over $\mathbb{F}_q$.

Thus, we have the following result:

**Lemma 14.3.7.** *With $O(s \log s (Nm)^2)$ operations over $\mathbb{F}_q$, the algorithm above can return an affine subspace of dimension $s-1$ that contains all the polynomials of degree at most $k-1$*

*that need to be output. Further, the exact set of solution can be computed in with additional $O(q^{s-1} \cdot (Nm)^2)$ operations over $\mathbb{F}_q$.*

### 14.3.4 Wrapping Up

By Theorem 14.3.3, we know that we can list decode a folded Reed-Solomon code with folding parameter $m \geq 1$ up to

$$\frac{s}{s+1} \cdot \left(1 - \frac{m}{m-s+1} \cdot R\right) \tag{14.17}$$

fraction of errors for any $1 \leq s \leq m$.

To obtain our desired bound $1 - R - \varepsilon$ fraction of errors, we instantiate the parameter $s$ and $m$ such that

$$\frac{s}{s+1} \geq 1 - \varepsilon \text{ and } \frac{m}{m-s+1} \leq 1 + \varepsilon. \tag{14.18}$$

It is easy to check that one can choose

$$s = \Theta(1/\varepsilon) \text{ and } m = \Theta(1/\varepsilon^2)$$

such that the bounds in (14.18) are satisfied. Using the bounds from (14.18) in (14.17) implies that the algorithm can handle at least

$$(1-\varepsilon)(1-(1+\varepsilon)R) = 1 - \varepsilon - R + \varepsilon^2 R > 1 - R - \varepsilon$$

fraction of errors, as desired.

We are almost done since Lemma 14.3.7 shows that the run time of the algorithm is $q^{O(s)}$. The only thing we need to choose is $q$: for the final result we pick $q$ to be the smallest power of 2 that is larger than $Nm + 1$. Then the discussion above along with Lemma 14.3.7 implies the following result (the claim on strong explicitness follows from the fact that Reed-Solomon codes are strongly explicit):

**Theorem 14.3.8.** *There exist* strongly explicit *folded Reed-Solomon codes of rate $R$ that for large enough block length $N$ can be list decoded from $1 - R - \varepsilon$ fraction of errors (for any small enough $\varepsilon > 0$) in time $\left(\frac{N}{\varepsilon}\right)^{O(1/\varepsilon)}$. The worst-case list size is $\left(\frac{N}{\varepsilon}\right)^{O(1/\varepsilon)}$ and the alphabet size is $\left(\frac{N}{\varepsilon}\right)^{O(1/\varepsilon^2)}$.*

## 14.4 Bibliographic Notes and Discussion

There was no improvement to the Guruswami-Sudan result (Theorem 13.2.6) for about seven years till Parvaresh and Vardy showed that "Correlated" Reed-Solomon codes can be list-decoded up to $1 - (mR)^{\frac{1}{m+1}}$ fraction of errors for $m \geq 1$ [43]. Note that for $m = 1$, correlated Reed-Solomon codes are equivalent to Reed-Solomon codes and the result of Parvaresh and Vardy recovers Theorem 13.2.6. Immediately, after that Guruswami and Rudra [24] showed that Folded Reed-Solomon codes can achieve the list-decoding capacity of $1 - R - \varepsilon$ and hence, answer

Question 14.0.1 in the affirmative. Guruswami [19] reproved this result but with a much simpler proof. In this chapter, we studied the proof due to Guruswami. Guruswami in [19] credits Salil Vadhan for the the interpolation step. An algorithm presented in Brander's thesis [2] shows that for the special interpolation in Algorithm 18, one can perform the interpolation in $O(Nm\log^2(Nm)\log\log(Nm))$ operations over $\mathbb{F}_q$. The idea of using the "sliding window" for list decoding Folded Reed-Solomon codes is originally due to Guruswami and Rudra [23].

The bound of $q^{s-1}$ on the list size for folded Reed-Solomon codes was first proven in [23] by roughly the following argument. One reduced the problem of finding roots to finding roots of a *univariate* polynomial related to $Q$ over $\mathbb{F}_{q^k}$. (Note that each polynomial in $\mathbb{F}_q[X]$ of degree at most $k-1$ has a one to one correspondence with elements of $\mathbb{F}_{q^k}$– see e.g. Theorem 11.2.1.) The list size bound follows from the fact that this new univariate polynomial had degree $q^{s-1}$. Thus, implementing the algorithm entails running a root finding algorithm over a big extension field, which in practice has terrible performance.

**Discussion.** For constant $\varepsilon$, Theorem 14.3.8 answers Question 14.0.1 in the affirmative. However, from a practical point of view, there are three issues with the result: alphabet, list size and run time. Below we tackle each of these issues.

**Large Alphabet.** Recall that one only needs an alphabet of size $2^{O(1/\varepsilon)}$ to be able to list decode from $1-R-\varepsilon$ fraction of errors, which is independent of $N$. It turns out that combining Theorem 14.3.8 along with code concatenation and expanders allows us to construct codes over alphabets of size roughly $2^{O(1/\varepsilon^4)}$ [23]. (The idea of using expanders and code concatenation was not new to [23]: the connection was exploited in earlier work by Guruswami and Indyk [22].)

The above however, does not answer the question of achieving list decoding capacity for *fixed* $q$, say e.g. $q=2$. We know that there exists binary code of rate $R$ that are $(H^{-1}(1-R-\varepsilon), O(1/\varepsilon))$-list decodable codes (see Theorem 7.4.1). The best known explicit codes with efficient list decoding algorithms are those achieved by concatenating folded Reed-Solomon codes with suitable inner codes achieve the so called *Blokh-Zyablov* bound [25]. However, the tradeoff is far from the list decoding capacity. As one sample point, consider the case when we want to list decode from $\frac{1}{2}-\varepsilon$ fraction of errors. Then the result of [25] gives codes of rate $\Theta(\varepsilon^3)$ while the codes on list decoding capacity has rate $\Omega(\varepsilon^2)$. The following fundamental question is still very much wide open:

**Open Question 14.4.1.** *Do there exist explicit binary codes with rate $R$ that can be list decoded from $H^{-1}(1-R-\varepsilon)$ fraction of errors with polynomial list decoding algorithms?*

The above question is open even if we drop the requirement on efficient list decoding algorithm or we only ask for a code that can list decode from $1/2-\varepsilon$ fraction of errors with rate $\Omega(\varepsilon^a)$ for some $a<3$. It is known (combinatorially) that concatenated codes can achieve the list decoding capacity but the result is via a souped up random coding argument and does not give much information about an efficient decoding algorithm [26].

**List Size.** It is natural to wonder if the bound on the list size in Lemma 14.3.4 above can be improved as that would show that folded Reed-Solomon codes can be list decoded up to the list decoding capacity but with a smaller output list size than Theorem 14.3.8. Guruswami showed that in its full generality the bound cannot be improved [19]. In particular, he exhibits explicit polynomials $A_0(X),\ldots,A_s(X)$ such that there are at least $q^{s-2}$ solutions for $f(X)$ that satisfy (14.8). However, these $A_i(X)$'s are not known to be the output for an actual interpolation instance. In other words, the following question is still open:

**Open Question 14.4.2.** *Can folded Reed-Solomon codes of rate $R$ be list decoded from $1-R-\varepsilon$ fraction of errors with list size $f(1/\varepsilon)\cdot N^c$ for some increasing function $f(\cdot)$ and absolute constant $c$?*

Even the question above with $N^{(1/\varepsilon)^{o(1)}}$ is still open.

However, if one is willing to consider codes other than folded Reed-Solomon codes in order to answer to achieve list decoding capacity with smaller list size (perhaps with one only dependent on $\varepsilon$), then there is good news. Guruswami in the same paper that presented the algorithm in this chapter also present a *randomized* construction of codes of rate $R$ that are $(1-R-\varepsilon, O(1/\varepsilon^2))$-list decodable codes [19]. This is of course worse than what we know from the probabilistic method. However, the good thing about the construction of Guruswami comes with an $O(N/\varepsilon)^{O(1/\varepsilon)}$-list decoding algorithm.

Next we briefly mention the key ingredient in the result above. To see the potential for improvement consider Corollary 14.3.6. The main observation is that all the potential solutions lie in an affine subspace of dimension $s-1$. The key idea in [19] was use the folded Reed-Solomon encoding for a special subset of the message space $\mathbb{F}_q^k$. Call a subspace $S\subseteq\mathbb{F}_q^k$ to be a $(q,k,\varepsilon,\ell,L)$-*subspace evasive subset* if

1. $|S|\geq q^{k(1-\varepsilon)}$; and
2. For any (affine) subspace $T\subseteq\mathbb{F}_q^k$ of dimension $\ell$, we have $|S\cap T|\leq L$.

The code in [19], applies the folded Reed-Solomon encoding on a $\left(q,k,s,O\left(s^2\right)\right)$-subspace evasive subset (such a subset can be shown to exist via the probabilistic method). The reason why this sub-code of folded Reed-Solomon code works is as follows: Condition (1) ensures that the new code has rate at least $R(1-\varepsilon)$, where $R$ is the rate of the original folded Reed-Solomon code and condition (2) ensures that the number of output polynomial in the root finding step of the algorithm we considered in the last section is at most $L$. (This is because by Corollary 14.3.6 the output message space is an affine subspace of dimension $s-1$ in $\mathbb{F}_Q^k$. However, in the new code by condition 2, there can be at most $O\left(s^2\right)$ output solutions.)

The result above however, has two shortcomings: (i) the code is no longer explicit and (ii) even though the worst case list size is $O\left(\frac{1}{\varepsilon^2}\right)$, it was not know how to obtain this output without listing all the $q^{s-1}$ possibilities and pruning them against $S$. The latter meant that the decoding runtime did not improve over the one achieved in Theorem 14.3.8.

**Large Runtime.** We finally address the question of the high run time of all the list decoding algorithms so far. Dvir and Lovett [9], presented a construction of an *explicit* $(q, k, \varepsilon, s, s^{O(s)})$-subspace evasive subset $S^*$. More interestingly, given any affine subspace $T$ of dimension at most $s$, it can compute $S \cap T$ in time proportional to the output size. Thus, this result along with the discussion above implies the following result:

**Theorem 14.4.1.** *There exist* strongly explicit *codes of rate $R$ that for large enough block length $N$ can be list decoded from $1-R-\varepsilon$ fraction of errors (for any small enough $\varepsilon > 0$) in time $O\left(\left(\frac{N}{\varepsilon^2}\right)^2\right) + \left(\frac{1}{\varepsilon}\right)^{O(1/\varepsilon)}$. The worst-case list size is $\left(\frac{1}{\varepsilon}\right)^{O(1/\varepsilon)}$ and the alphabet size is $\left(\frac{N}{\varepsilon}\right)^{O(1/\varepsilon^2)}$.*

The above answers Question 14.0.1 pretty satisfactorily. However, to obtain a completely satisfactory answer one would have to solve the following open question:

**Open Question 14.4.3.** *Are there explicit codes of rate $R > 0$ that are $\left(1 - R - \varepsilon, (1/\varepsilon)^{O(1)}\right)$-list decodable that can be list-decoded in time* $\text{poly}(N, 1/\varepsilon)$ *over alphabet of size $q \le poly(n)$?*

The above question, without the requirement of explicitness, has been answered by Guruswami and Xing [29].