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 13

Efficient Decoding of Reed-Solomon Codes

So far in this book, we have shown how to efficiently decode explicit codes up to half of the Zyablov bound (Theorem 11.3.3) and how to efficiently achieve the capacity of the BSC_p (Theorem 12.4.1). The proofs of both of these results assumed that one can efficiently do unique decoding for Reed-Solomon codes up to half its distance (Theorem 11.2.1). In this chapter, we present such a unique decoding algorithm. Then we will generalize the algorithm to a list decoding algorithm that efficiently achieves the Johnson bound (Theorem 7.3.1).

13.1 Unique decoding of Reed-Solomon codes

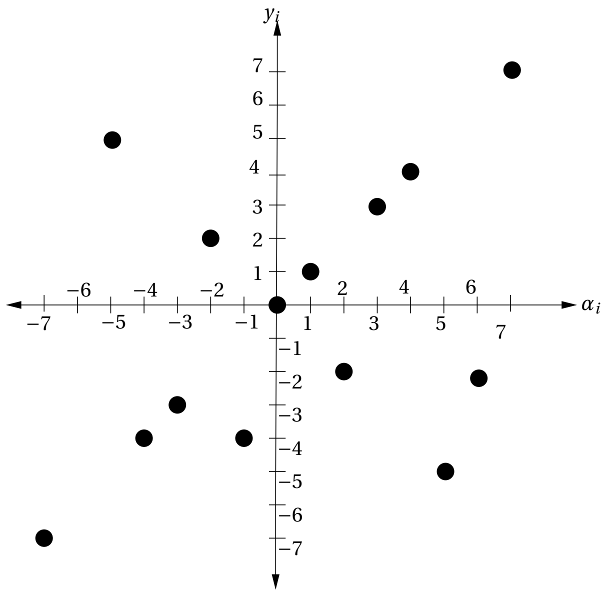
Consider the $[n,k,d=n-k+1]_q$ Reed-Solomon code with evaluation points $(\alpha_1,\cdots,\alpha_n)$. (Recall Definition 5.2.1.) Our goal is to describe an algorithm that corrects up to $e<\frac{n-k+1}{2}$ errors in polynomial time. Let $\mathbf{y}=\left(y_1,\cdots,y_n\right)\in\mathbb{F}_q^n$ be the received word. We will now do a syntactic shift that will help us better visualize all the decoding algorithms in this chapter better. In particular, we will also think of $\mathbf{y}$ as the set of ordered pairs $\{(\alpha_1,y_1),(\alpha_2,y_2),\ldots,(\alpha_n,y_n)\}$, that is, we think of $\mathbf{y}$ as a collection of “points" in “2-D space." See Figure 13.1 for an illustration. From now on, we will switch back and forth between our usual vector interpretation of $\mathbf{y}$ and this new geometric notation.

Further, let us assume that there exists a polynomial $P(X)$ of degree at most $k-1$ such that $\Delta\left(\mathbf{y},(P(\alpha_i))_{i=1}^n\right)\le e$. (Note that if such a $P(X)$ exists then it is unique.) See Figure 13.2 for an illustration.

We will use reverse engineering to design a unique decoding algorithm for Reed-Solomon codes. We will assume that we somehow know $P(X)$ and then prove some identities involving (the coefficients of) $P(X)$. Then to design the algorithm we will just use the identities and try to solve for $P(X)$. Towards this end, let us assume that we also magically got our hands on a polynomial $E(X)$ such that

$$E(\alpha_i)=0 \text{ if and only if } y_i\neq P(\alpha_i).$$

$E(X)$ is called an *error-locator polynomial.* We remark that there exists such a polynomial of



$n = 14, k = 2, e = 6$

Figure 13.1: An illustration of a received word for a [14,2] Reed-Solomon code (we have implicitly embedded the field $\mathbb{F}_q$ in the set $\{-7,\dots,7\}$). The evaluations points are $(-7,-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7)$ and the received word is $(-7,5,-4,-3,2,-4,0,1,-2,3,4,-5,-2,7)$.

degree at most e. In particular, consider the polynomial:

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

See Figure 13.3 for an illustration of the $E(X)$ corresponding to the received word in Figure 13.1.

Now we claim that for every $1 \le i \le n$,

$$y_i E(\alpha_i) = P(\alpha_i) E(\alpha_i). \tag{13.1}$$

To see why (13.1) is true, note that if $y_i \neq P(\alpha_i)$, then both sides of (13.1) are 0 (as $E(\alpha_i) = 0$). On the other hand, if $y_i = P(\alpha_i)$, then (13.1) is obviously true.

All the discussion above does not seem to have made any progress as both $E(X)$ and $P(X)$ are unknown. Indeed, the task of the decoding algorithm is to find $P(X)$! Further, if $E(X)$ is known then one can easily compute $P(X)$ from **y** (the proof is left as an exercise). However, note that we can now try and do reverse engineering. If we think of coefficients of $P(X)$ (of which there are k) and the coefficients of $E(X)$ (of which there are $e+1$) as variables, then we have n equations from (13.1) in $e+k+1$ variables. From our bound on e, this implies we have more equations than variables. Thus, if we could solve for these unknowns, we would be done.

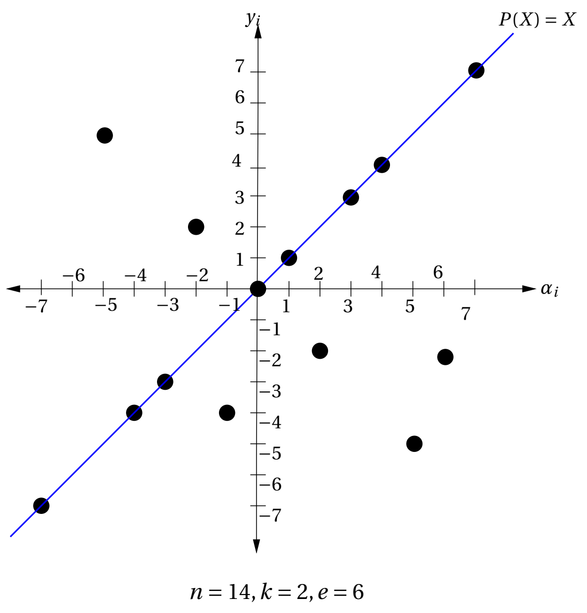


Figure 13.2: An illustration of the closest codeword $P(X) = X$ for the received word from Figure 13.1. Note that we are considering polynomials of degree 1, which are "lines."

However, there is a catch– these n equations are quadratic equations, which in general are NP-hard to solve. However, note that for our choice of e, we have $e+k-1 \ll n$. Next, we will exploit this with a trick that is sometimes referred to as linearization. The idea is to introduce new variables so that we can convert the quadratic equations into linear equations. Care must be taken so that the number of variables after this linearization step is still smaller than the (now linear) n equations. Now we are in familiar territory as we know how to solve linear equations over a field (e.g. by Gaussian elimination). (See section 13.4 for some more discussion on the hardness of solving quadratic equations and the linearization technique.)

To perform linearization, define $N(X) \overset{\text{def}}{=} P(X) \cdot E(X)$. Note that $N(X)$ is a polynomial of degree less than or equal to $e+k-1$. Further, if we can find $N(X)$ and $E(X)$, then we are done. This is because we can compute $P(X)$ as follows:

$$P(X) = \frac{N(X)}{E(X)}.$$

The main idea in the Welch-Berlekamp algorithm is to "forget" what $N(X)$ and $E(X)$ are meant to be (other than the fact that they are degree bounded polynomials).

13.1.1 Welch-Berlekamp Algorithm

Algorithm 12 formally states the Welch-Berlekamp algorithm.

As we have mentioned earlier, computing $E(X)$ is as hard as finding the solution polynomial $P(X)$. Also in some cases, finding the polynomial $N(X)$ is as hard as finding $E(X)$. E.g., given

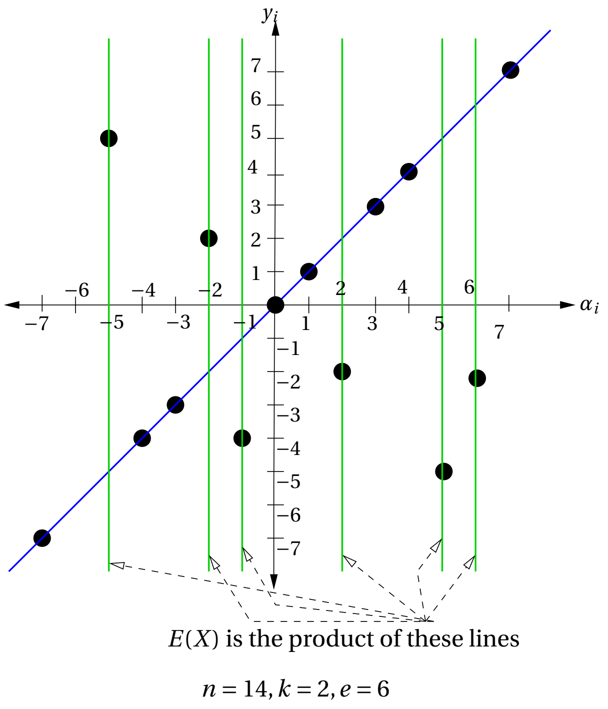


Figure 13.3: An illustration of the the error locator polynomial $E(X) = (X+5)(X+2)(X+1)(X-2)(X-5)(X-6)$ for the received word from Figure 13.1. Note that $E(X)$ is the product of the green lines.

$N(X)$ and $\mathbf{y}$ (such that $y_i \neq 0$ for $1 \le i \le n$) one can find the error locations by checking positions where $N(\alpha_i) = 0$. While each of the polynomials $E(X)$, $N(X)$ is hard to find individually, the main insight in the Welch-Berlekamp algorithm is that computing them together is easier.

Next we analyze the correctness and run time of Algorithm 12.

Correctness of Algorithm 12. Note that if Algorithm 12 does not output fail, then the algorithm produces a correct output. Thus, to prove the correctness of Algorithm 12, we just need the following result.

Theorem 13.1.1. *If $(P(\alpha_i))_{i=1}^n$ is transmitted (where $P(X)$ is a polynomial of degree at most $k-1$) and at most $e < \frac{n-k+1}{2}$ errors occur (i.e. $\Delta(\mathbf{y}, (P(\alpha_i))_{i=1}^n) \le e$), then the Welch-Berlekamp algorithm outputs $P(X)$.*

The proof of the theorem above follows from the subsequent claims.

Claim 13.1.2. *There exist a pair of polynomials $E^*(X)$ and $N^*(X)$ that satisfy* Step 1 *such that $\frac{N^*(X)}{E^*(X)} = P(X)$.*

Note that now it suffices to argue that $\frac{N_1(X)}{E_1(X)} = \frac{N_2(X)}{E_2(X)}$ for any pair of solutions $((N_1(X), E_1(X))$ and $(N_2(X), E_2(X))$ that satisfy Step 1, since Claim 13.1.2 above can then be used to see that ratio must be $P(X)$. Indeed, we will show this to be the case:

Algorithm 12 Welch-Berlekamp Algorithm

```
INPUT: $n \ge k \ge 1$, $0 < e < \frac{n-k+1}{2}$ and $n$ pairs $\{(\alpha_i, y_i)\}_{i=1}^n$ with $\alpha_i$ distinct
OUTPUT: Polynomial $P(X)$ of degree at most $k-1$ or fail.

1: Compute a non-zero polynomial $E(X)$ of degree exactly $e$, and a polynomial $N(X)$ of degree
   at most $e+k-1$ such that
       $y_i E(\alpha_i) = N(\alpha_i) \quad 1 \le i \le n.$        (13.2)
2: IF $E(X)$ and $N(X)$ as above do not exist or $E(X)$ does not divide $N(X)$ THEN
3:     RETURN fail
4: $P(X) \leftarrow \frac{N(X)}{E(X)}$.
5: IF $\Delta(\mathbf{y}, (P(\alpha_i))_{i=1}^n) > e$ THEN
6:     RETURN fail
7: ELSE
8:     RETURN $P(X)$
```

Claim 13.1.3. *If any two distinct solutions* $(E_1(X), N_1(X)) \neq (E_2(X), N_2(X))$ *satisfy* Step 1*, then they will satisfy*

$$\frac{N_1(X)}{E_1(X)} = \frac{N_2(X)}{E_2(X)}.$$

Proof of Claim 13.1.2. We just take $E^*(X)$ to be the error-locating polynomial for $P(X)$ and let $N^*(X) = P(X)E^*(X)$ where $\deg(N^*(X)) \le \deg(P(X)) + \deg(E^*(X)) \le e+k-1$. In particular, define $E^*(X)$ as the following polynomial of degree exactly e:

$$E^*(X) = X^{e-\Delta\left(\mathbf{y},(P(\alpha_i))_{i=1}^n\right)} \prod_{1\le i\le n \mid y_i \neq P(\alpha_i)} (X - \alpha_i). \tag{13.3}$$

By definition, $E^*(X)$ is a non-zero polynomial of degree e with the following property:

$$E^*(\alpha_i) = 0 \quad \text{iff} \quad y_i \neq P(\alpha_i).$$

We now argue that $E^*(X)$ and $N^*(X)$ satisfy (13.2). Note that if $E^*(\alpha_i) = 0$, then $N^*(\alpha_i) = P(\alpha_i)E^*(\alpha_i) = y_i E^*(\alpha_i) = 0$. When $E^*(\alpha_i) \neq 0$, we know $P(\alpha_i) = y_i$ and so we still have $P(\alpha_i)E^*(\alpha_i) = y_i E^*(\alpha_i)$, as desired. □

Proof of Claim 13.1.3. Note that the degrees of the polynomials $N_1(X)E_2(X)$ and $N_2(X)E_1(X)$ are at most $2e+k-1$. Let us define polynomial $R(X)$ with degree at most $2e+k-1$ as follows:

$$R(X) = N_1(X)E_2(X) - N_2(X)E_1(X). \tag{13.4}$$

Furthermore, from Step 1 we have, for every $i \in [n]$,

$$N_1(\alpha_i) = y_i E_1(\alpha_i) \quad \text{and} \quad N_2(\alpha_i) = y_i E_2(\alpha_i). \tag{13.5}$$

Substituting (13.5) into (13.4) we get for $1 \le i \le n$:

$$\begin{aligned} R(\alpha_i) &= (y_i E_1(\alpha_i))E_2(\alpha_i) - (y_i E_2(\alpha_i))E_1(\alpha_i) \\ &= 0. \end{aligned}$$

The polynomial $R(X)$ has n roots and

$$\begin{aligned} \deg(R(X)) &\le e + k - 1 + e \\ &= 2e + k - 1 \\ &< n, \end{aligned}$$

Where the last inequality follows from the upper bound on e. Since $\deg(R(X)) < n$, by the degree mantra (Proposition 5.2.3) we have $R(X) \equiv 0$. This implies that $N_1(X)E_2(X) \equiv N_2(X)E_1(X)$. Note that as $E_1(X) \ne 0$ and $E_2(X) \ne 0$, this implies that $\frac{N_1(X)}{E_1(X)} = \frac{N_2(X)}{E_2(X)}$, as desired. □

Implementation of Algorithm 12. In Step 1, $N(X)$ has $e+k$ unknowns and $E(X)$ has $e+1$ unknowns. For each $1 \le i \le n$, the constraint in (13.2) is a linear equation in these unknowns. Thus, we have a system of n linear equations in $2e+k+1 < n+2$ unknowns. By claim 13.1.2, this system of equations have a solution. The only extra requirement is that the degree of the polynomial $E(X)$ should be exactly e. We have already shown $E(X)$ in equation (13.3) to satisfy this requirement. So we add a constraint that the coefficient of X^e in $E(X)$ is 1. Therefore, we have $n+1$ linear equation in at most $n+1$ variables, which we can solve in time $O(n^3)$, e.g. by Gaussian elimination.

Finally, note that Step 4 can be implemented in time $O(n^3)$ by "long division." Thus, we have proved

Theorem 13.1.4. *For any $[n,k]_q$ Reed-Solomon code, unique decoding can be done in $O(n^3)$ time up to $\frac{d-1}{2} = \frac{n-k}{2}$ number of errors.*

Recall that the above is a restatement of the error decoding part of Theorem 11.2.1. Thus, this fills in the final missing piece from the proofs of Theorem 11.3.3 (decoding certain concatenated codes up to half of their design distance) and Theorem 12.4.1 (efficiently achieving the BSC_p capacity).

13.2 List Decoding Reed-Solomon Codes

Recall Question 7.4.3, which asks if there is an efficient list decoding algorithm for a code of rate $R > 0$ that can correct $1 - \sqrt{R}$ fraction of errors. Note that in the question above, explicitness is not an issue as e.g., a Reed-Solomon code of rate R by the Johnson bound is $(1-\sqrt{R}, O(n^2))$-list decodable (Theorem 7.3.1).

We will study an efficient list decoding algorithm for Reed-Solomon codes that can correct up to $1-\sqrt{R}$ fraction of errors. To this end, we will present a sequence of algorithms for (list) decoding Reed-Solomon codes that ultimately will answer Question 7.4.3.

Before we talk about the algorithms, we restate the (list) decoding problem for Reed-Solomon codes. Consider any $[n,k]_q$ Reed-Solomon code that has the evaluation set $\{\alpha_1,\dots,\alpha_m\}$. Below is a formal definition of the decoding problem for Reed-Solomon codes:

- **Input**: Received word $\mathbf{y} = (y_1,\dots,y_n)$, $y_i \in \mathbb{F}_q$ and error parameter $e = n - t$.
- **Output**: All polynomials $P(X) \in \mathbb{F}_q[X]$ of degree at most $k-1$ such that $P(\alpha_i) = y_i$ for at least t values of i.

Our main goal of course is to make t as small as possible.

We begin with the unique decoding regime, where $t > \dfrac{n+k}{2}$. We looked at the Welch-Berlekamp algorithm in Algorithm 12, which we restate below in a slightly different form (that will be useful in developing the subsequent list decoding algorithms).

- **Step 1:** Find polynomials $N(X)$ of degree $k+e-1$, and $E(X)$ of degree e such that

$$N(\alpha_i) = y_i E(\alpha_i), \text{ for every } 1 \le i \le n$$

- **Step 2:** If $Y - P(X)$ divides $Q(X,Y) = YE(X) - N(X)$, then output $P(X)$ (assuming $\Delta(\mathbf{y}, (P(\alpha_i))_{i=1}^n) \le e$).

Note that $Y - P(X)$ divides $Q(X,Y)$ in **Step 2** above if and only if $P(X) = \frac{N(X)}{E(X)}$, which is exactly what Step 4 does in Algorithm 12.

13.2.1 Structure of list decoding algorithms for Reed-Solomon

Note that the Welch-Berlekamp Algorithm has the following general structure:

- **Step 1:** (Interpolation Step) Find non-zero $Q(X,Y)$ such that $Q(\alpha_i, y_i) = 0, 1 \le i \le n$.
- **Step 2:** (Root Finding Step) If $Y - P(X)$ is a factor of $Q(X,Y)$, then output $P(X)$ (assuming it is close enough to the received word).

In particular, in the Welch-Berlekamp algorithm we have $Q(X,Y) = YE(X) - N(X)$ and hence, **Step 2** is easy to implement.

All the list decoding algorithms that we will consider in this chapter will have the above two-step structure. The algorithms will differ in how exactly **Step 1** is implemented. Before we move on to the details, we make one observation that will effectively "take care of" **Step 2** for us. Note that **Step 2** can be implemented if one can factorize the bivariate polynomial $Q(X,Y)$ (and then only retain the linear factors of the form $Y - P(X)$). Fortunately, it is known that factoring bivariate polynomials can be done in polynomial time (see e.g. [35]). We will not prove this result in the book but will use this fact as a given.

Finally, to ensure the correctness of the two-step algorithm above for Reed-Solomon codes, we will need to ensure the following:

- **Step 1** requires solving for the co-efficients of $Q(X,Y)$. This can be done as long as the number of coefficients is greater than the the number of constraints. (The proof of this fact is left as an exercise.) Also note that this argument is a departure from the corresponding argument for the Welch-Berlekamp algorithm (where the number of coefficients is upper bounded by the number of constraints).

- In **Step 2**, to ensure that for every polynomial $P(X)$ that needs to be output $Y - P(X)$ divides $Q(X,Y)$, we will add restrictions on $Q(X,Y)$. For example, for the Welch-Berlekamp algorithm, the constraint is that $Q(X,Y)$ has to be of the form $YE(X) - N(X)$, where $E(X)$ and $N(X)$ are non-zero polynomials of degree e and at most $e+k-1$ respectively.

Next, we present the first instantiation of the algorithm structure above, which leads us to our first list decoding algorithm for Reed-Solomon codes.

13.2.2 Algorithm 1

The main insight in the list decoding algorithm that we will see is that if we carefully control the degree of the polynomial $Q(X,Y)$, then we can satisfy the required conditions that will allow us to make sure **Step 1** succeeds. Then we will see that the degree restrictions, along with the degree mantra (Proposition 5.2.3) will allow us to show **Step 2** succeeds too. The catch is in defining the correct notion of degree of a polynomial. We do that next.

First, we recall the definition of maximum degree of a variable.

Definition 13.2.1. $\deg_X(Q)$ is the maximum degree of X in $Q(X,Y)$. Similarly, $\deg_Y(Q)$ is the maximum degree of Y in $Q(X,Y)$

For example, for $Q(X,Y) = X^2Y^3 + X^4Y^2$ $\deg_X(Q) = 4$ and $\deg_Y(Q) = 3$. Given $\deg_X(Q) = a$ and $\deg_Y(Q) = b$, we can write

$$Q(X,Y) = \sum_{\substack{0 \le i \le a,\\ 0 \le j \le b}} c_{ij} X^i Y^j,$$

where the coefficients $c_{ij} \in \mathbb{F}_q$. Note that the number of coefficients is equal to $(a+1)(b+1)$.

The main idea in the first list decoding algorithm for Reed-Solomon code is to place bounds on $\deg_X(Q)$ and $\deg_Y(Q)$ for **Step 1**. The bounds are chosen so that there are enough variables to guarantee the existence of a $Q(X,Y)$ with the required properties. We will then use these bound along with the degree mantra (Proposition 5.2.3) to argue that **Step 2** works. Algorithm 13 presents the details. Note that the algorithm generalizes the Welch-Berlekamp algorithm (and follows the two step skeleton outlined above).

Correctness of Algorithm 13. To ensure the correctness of Step 1, we will need to ensure that the number of coefficients for $Q(X,Y)$ (which is $(\ell+1)(n/\ell+1)$) is larger than the number of constraints in (13.6 (which is n). Indeed,

$$(\ell+1)\cdot\left(\frac{n}{\ell}+1\right) > \ell\cdot\frac{n}{\ell} = n.$$

Algorithm 13 The First List Decoding Algorithm for Reed-Solomon Codes

```
INPUT: $n \ge k \ge 1$, $\ell \ge 1$, $e = n - t$ and $n$ pairs $\{(\alpha_i, y_i)\}_{i=1}^n$
OUTPUT: (Possibly empty) list of polynomials $P(X)$ of degree at most $k-1$

1: Find a non-zero $Q(X,Y)$ with $\deg_X(Q) \le \ell, \deg_Y(Q) \le \frac{n}{\ell}$ such that
       $$Q(\alpha_i, y_i) = 0, 1 \le i \le n. \quad (13.6)$$
2: Ł ← ∅
3: FOR every factor $Y - P(X)$ of $Q(X,Y)$ DO
4:     IF $\Delta(\mathbf{y}, (P(\alpha_i))_{i=1}^n) \le e$ and $\deg(P) \le k-1$ THEN
5:         Add $P(X)$ to Ł.
6: RETURN Ł
```

To argue that the final Ł in Step 6 contains all the polynomials $P(X)$ that need to be output. In other words, we need to show that if $P(X)$ of degree $\le k-1$ agrees with Y in at least t positions, then $Y - P(X)$ divides $Q(X,Y)$. Towards this end, we define

$$R(X) \overset{\text{def}}{=} Q(X, P(X)).$$

Note that $Y - P(X)$ divides $Q(X,Y)$ if and only if $R(X) \equiv 0$. Thus, we need to show $R(X) \equiv 0$. For the sake of contradiction, assume that $R(X) \not\equiv 0$. Note that

$$\begin{aligned} \deg(R) &\le \deg_X(Q) + \deg(P) \cdot \deg_Y(Q) && (13.7)\\ &\le \ell + \frac{n(k-1)}{\ell}. && (13.8) \end{aligned}$$

On the other hand, if $P(\alpha_i) = y_i$ then (13.6) implies that

$$Q(\alpha_i, y_i) = Q(\alpha_i, P(\alpha_i)) = 0.$$

Thus, α_i is a root of $R(X)$. In other words R has at least t roots. Note that the degree mantra (Proposition 5.2.3) this will lead to a contradiction if $t > \deg(R)$, which will be true if

$$t > \ell + \frac{n(k-1)}{\ell}.$$

If we pick $\ell = \sqrt{n(k-1)}$, we will have $t > 2\sqrt{n(k-1)}$. Thus, we have shown

Theorem 13.2.1. *Algorithm 13 can list decode Reed-Solomon codes of rate R from* $1 - 2\sqrt{R}$ *fraction of errors. Further, the algorithm can be implemented in polynomial time.*

The claim on the efficient run time follows as Step 1 can be implemented by Gaussian elimination and for Step 3, all the factors of $Q(X,Y)$ (and in particular all linear factors of the form $Y - P(X)$) can be computed using e.g. the algorithm from [35].

The bound $1 - 2\sqrt{R}$ is better than the unique decoding bound of $\frac{1-R}{2}$ for $R < 0.07$. This is still far from the $1 - \sqrt{R}$ fraction of errors guaranteed by the Johnson bound. See Figure 13.2.2 for an illustration.

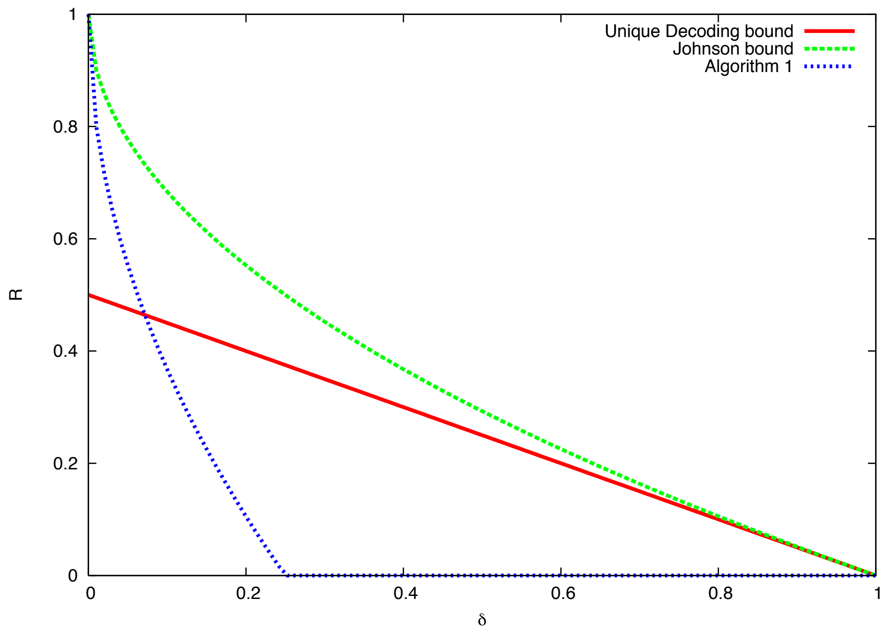


Figure 13.4: The tradeoff between rate R and the fraction of errors that can be corrected by Algorithm 13.

13.2.3 Algorithm 2

To motivate the next algorithm, recall that in Algorithm 13, in order to prove that the root finding step (Steps 3-6 in Algorithm 13) works, we defined a polynomial $R(X) \overset{\text{def}}{=} Q(X, P(X))$. In particular, this implied that $\deg(R) \le \deg_X(Q) + (k-1) \cdot \deg_Y(Q)$ (and we had to select $t > \deg_X(Q) + (k-1) \cdot \deg_Y(Q)$). One shortcoming of this approach is that the maximum degree of X and Y might not occur in the same term. For example, in the polynomial $X^2Y^3 + X^4Y^2$, the maximum X and Y degrees do not occur in the same monomial. The main insight in the new algorithm is to use a more "balanced" notion of degree of $Q(X, Y)$:

Definition 13.2.2. The $(1, w)$ weighted degree of the monomial X^iY^j is $i + wj$. Further, the $(1, w)$-weighted degree of $Q(X, Y)$ (or just its $(1, w)$ degree) is the maximum $(1, w)$ weighted degree of its monomials.

For example, the $(1,2)$-degree of the polynomial $XY^3 + X^4Y$ is $\max(1+3\cdot 2, 4+2\cdot 1) = 7$. Also note that the $(1,1)$-degree of a bivariate polynomial $Q(X, Y)$ is its total degree (or the "usual" definition of degree of a bivariate polynomial). Finally, we will use the following simple lemma (whose proof we leave as an exercise):

Lemma 13.2.2. *Let $Q(X, Y)$ be a bivariate polynomial of $(1, w)$ degree D. Let $P(X)$ be a polynomial such that $\deg(P) \le w$. Then we have*

$$\deg(Q(X, P(X))) \le D.$$

Note that a bivariate polynomial $Q(X,Y)$ of $(1,w)$ degree at most D can be represented as follows:

$$Q(X,Y) \overset{\text{def}}{=} \sum_{\substack{i+wj\le D\\ i,j\ge 0}} c_{i,j} X^i Y^j,$$

where $c_{i,j} \in \mathbb{F}_q$.

The new algorithm is basically the same as Algorithm 13, except that in the interpolation step, we compute a bivariate polynomial of bounded $(1,k-1)$ degree. Before we state the precise algorithm, we will present the algorithm via an example. Consider the received word in Figure 13.5.

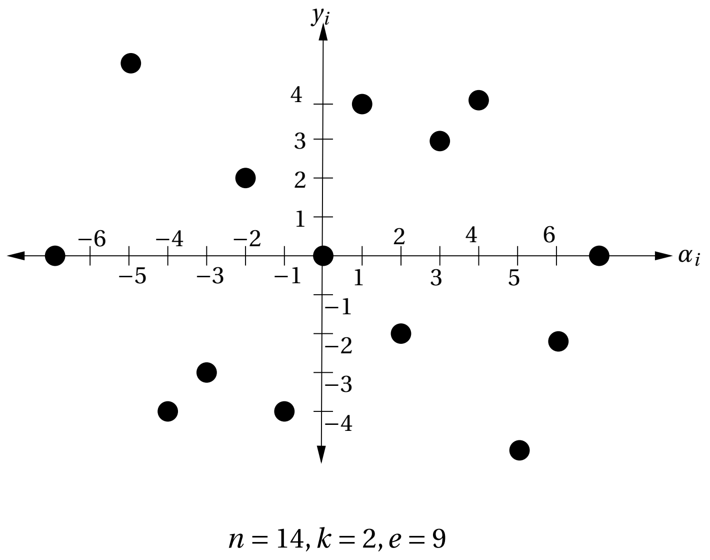


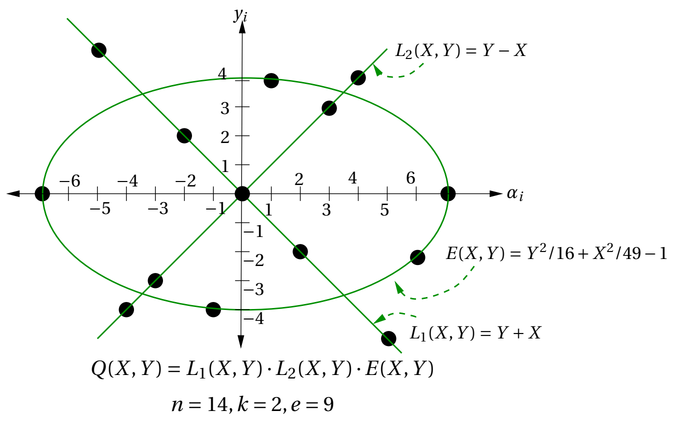
Figure 13.5: An illustration of a received word for the [14,2] Reed-Solomon code from Figure 13.1 (where again we have implicitly embedded the field $\mathbb{F}_q$ in the set $\{-7,\dots,7\}$). Here we have considered $e = 9$ errors which is more than what Algorithm 12 can handle. In this case, we are looking for lines that pass through at least 5 points.

Now we want to interpolate a bivariate polynomial $Q(X,Y)$ with a $(1,1)$ degree of 4 that "passes" through all the 2-D points corresponding to the received word from Figure 13.5. Figure 13.6 shows such an example.

Finally, we want to factorize all the linear factors $Y - P(X)$ of the $Q(X,Y)$ from Figure 13.6. Figure 13.7 shows the two polynomials X and $-X$ such that $Y - X$ and $Y + X$ are factors of $Q(X,Y)$ from Figure 13.6.

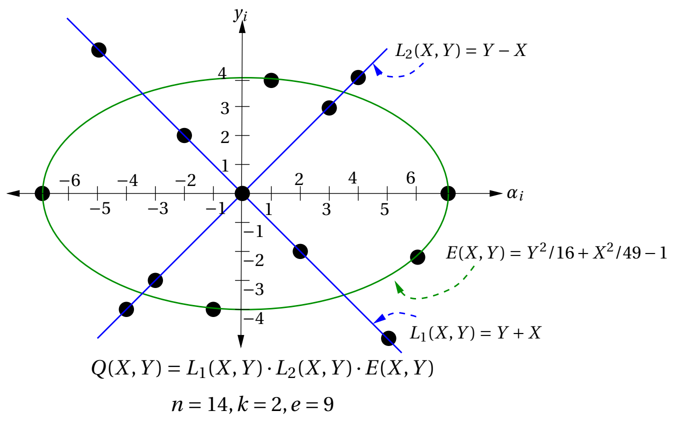
We now precisely state the new list decoding algorithm in Algorithm 14.

Proof of Correctness of Algorithm 14. As in the case of Algorithm 13, to prove the correctness of Algorithm 14, we need to do the following:



y_i
α_i
$L_2(X,Y) = Y - X$
$E(X,Y) = Y^2/16 + X^2/49 - 1$
$L_1(X,Y) = Y + X$
$Q(X,Y) = L_1(X,Y) \cdot L_2(X,Y) \cdot E(X,Y)$
$n = 14, k = 2, e = 9$

Figure 13.6: An interpolating polynomial $Q(X,Y)$ for the received word in Figure 13.5.



y_i
α_i
$L_2(X,Y) = Y - X$
$E(X,Y) = Y^2/16 + X^2/49 - 1$
$L_1(X,Y) = Y + X$
$Q(X,Y) = L_1(X,Y) \cdot L_2(X,Y) \cdot E(X,Y)$
$n = 14, k = 2, e = 9$

Figure 13.7: The two polynomials that need to be output are shown in blue.

- (Interpolation Step) Ensure that the number of coefficients of $Q(X,Y)$ is strictly greater than n.
- (Root Finding Step) Let $R(X) \overset{\text{def}}{=} Q(X, P(X))$. We want to show that if $P(\alpha_i) \geq y_i$ for at least t values of i, then $R(X) \equiv 0$.

To begin with, we argue why we can prove the correctness of the root finding step. Note that since $Q(X,Y)$ has $(1, k-1)$ degree at most D, Lemma 13.2.2 implies that

$$\deg(R) \leq D.$$

Then using the same argument as we used for the correctness of the root finding step of Algorithm 13, we can ensure $R(X) \equiv 0$ if we pick

$$t > D.$$

Algorithm 14 The Second List Decoding Algorithm for Reed-Solomon Codes

```
INPUT: $n \ge k \ge 1$, $D \ge 1$, $e = n - t$ and $n$ pairs $\{(\alpha_i, y_i)\}_{i=1}^n$
OUTPUT: (Possibly empty) list of polynomials $P(X)$ of degree at most $k-1$

1: Find a non-zero $Q(X,Y)$ with $(1, k-1)$ degree at most $D$, such that
       $$Q(\alpha_i, y_i) = 0, 1 \le i \le n. \quad (13.9)$$
2: Ł ← ∅
3: FOR every factor $Y - P(X)$ of $Q(X,Y)$ DO
4:     IF $\Delta(\mathbf{y}, (P(\alpha_i))_{i=1}^n) \le e$ and $\deg(P) \le k-1$ THEN
5:         Add $P(X)$ to Ł.
6: RETURN Ł
```

Thus, we would like to pick D to be as small as possible. On the other hand, Step 1 will need D to be large enough (so that the number of variables is more than the number of constraints in (13.9). Towards that end, let the number of coefficients of $Q(X,Y)$ be

$$\mathcal{N} = \left|\left\{(i,j) | i + (k-1)j \le D, i, j \in \mathbb{Z}^+\right\}\right|$$

To bound $\mathcal{N}$, we first note that in the definition above, $j \le \left\lfloor \frac{D}{k-1} \right\rfloor$. (For notational convenience, define $\ell = \left\lfloor \frac{D}{k-1} \right\rfloor$.) Consider the following sequence of relationships

$$\begin{aligned}
\mathcal{N} &= \sum_{j=1}^{\ell} \sum_{i=0}^{D-(k-1)j} 1 \\
&= \sum_{j=0}^{\ell} (D - (k-1)j + 1) \\
&= \sum_{j=0}^{\ell} (D+1) - (k-1)\sum_{j=0}^{\ell} j \\
&= (D+1)(\ell+1) - \frac{(k-1)\ell(\ell+1)}{2} \\
&= \frac{\ell+1}{2}(2D + 2 - (k-1)\ell) \\
&\ge \left(\frac{\ell+1}{2}\right)(D+2) && (13.10) \\
&\ge \frac{D(D+2)}{2(k-1)}. && (13.11)
\end{aligned}$$

In the above, (13.10) follows from the fact that $\ell \le \frac{D}{k-1}$ and (13.11) follows from the fact that $\frac{D}{k-1} - 1 \le \ell$.

Thus, the interpolation step succeeds (i.e. there exists a non-zero $Q(X,Y)$ with the required properties) if

$$\frac{D(D+2)}{2(k-1)} > n.$$

The choice

$$D = \left\lceil \sqrt{2(k-1)n} \right\rceil$$

suffices by the following argument:

$$\frac{D(D+2)}{2(k-1)} > \frac{D^2}{2(k-1)} \geq \frac{2(k-1)n}{2(k-1)} = n.$$

Thus for the root finding step to work, we need $t > \left\lceil \sqrt{2(k-1)n} \right\rceil$, which implies the following result:

Theorem 13.2.3. *Algorithm* 2 *can list decode Reed-Solomon codes of rate R from up to* $1-\sqrt{2R}$ *fraction of errors. Further, the algorithm runs in polynomial time.*

Algorithm 2 runs in polynomial time as Step 1 can be implemented using Gaussian elimination (and the fact that the number of coefficients is $O(n)$) while the root finding step can be implemented by any polynomial time algorithm to factorize bivariate polynomials. Further, we note that $1-\sqrt{2R}$ beats the unique decoding bound of $(1-R)/2$ for $R < 1/3$. See Figure 13.2.3 for an illustration.

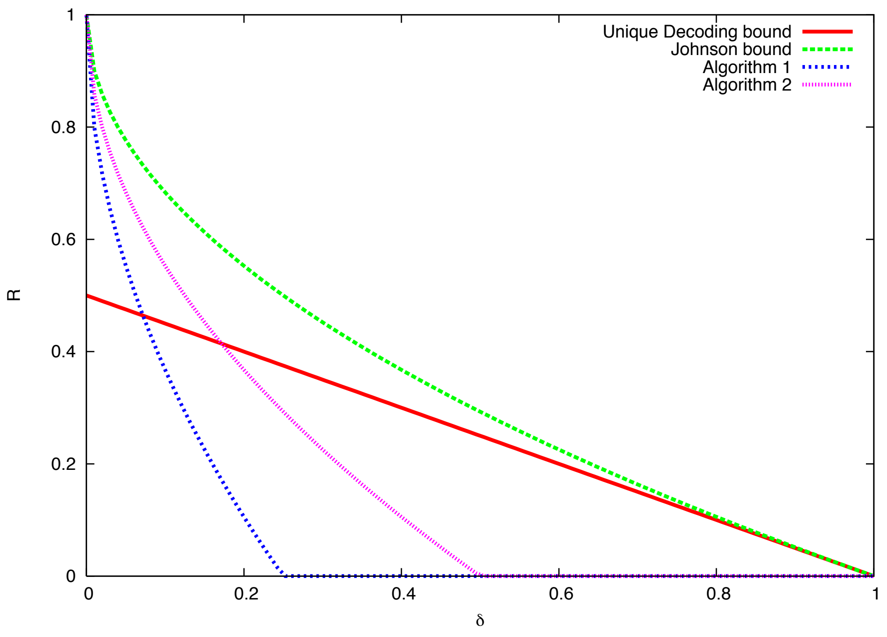


Figure 13.8: The tradeoff between rate R and the fraction of errors that can be corrected by Algorithm 13 and Algorithm 14.

13.2.4 Algorithm 3

Finally, we present the list decoding algorithm for Reed-Solomon codes, which can correct $1-\sqrt{R}$ fraction of errors. The main idea is to add more restrictions on $Q(X,Y)$ (in addition to its $(1,k-1)$-degree being at most D). This change will have the following implications:

- The number of constraints will increase but the number of coefficients will remain the same. This seems to be bad as this results in an increase in D (which in turn would result in an increase in t).
- However, this change will also increases the number of roots of $R(X)$ and this gain in the number of roots more than compensates for the increase in D.

In particular, the constraint is as follows. For some integer parameter $r \geq 1$, we will insist on $Q(X,Y)$ having r roots at (α_i, y_i), $1 \leq i \leq n$.

To motivate the definition of multiplicity of a root of a bivariate polynomial, let us consider the following simplified examples. In Figure 13.9 the curve $Q(X,Y) = Y - X$ passes through the

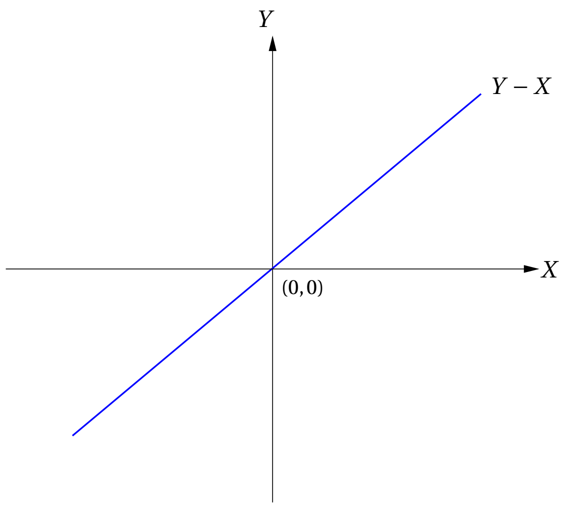


Figure 13.9: Multiplicity of 1

origin once and has no term of degree 0.

In Figure 13.10, the curve $Q(X,Y) = (Y-X)(Y+X)$ passes though the origin twice and has no term with degree at most 1.

In Figure 13.11, the curve $Q(X,Y) = (Y-X)(Y+X)(Y-2X)$ passes through the origin thrice and has no term with degree at most 2. More generally, if r lines pass through the origin, then note that the curve corresponding to their product has no term with degree at most $r-1$. This leads to the following more general definition:

Definition 13.2.3. $Q(X,Y)$ has r roots at $(0,0)$ if $Q(X,Y)$ doesn't have any monomial with degree at most $r-1$.

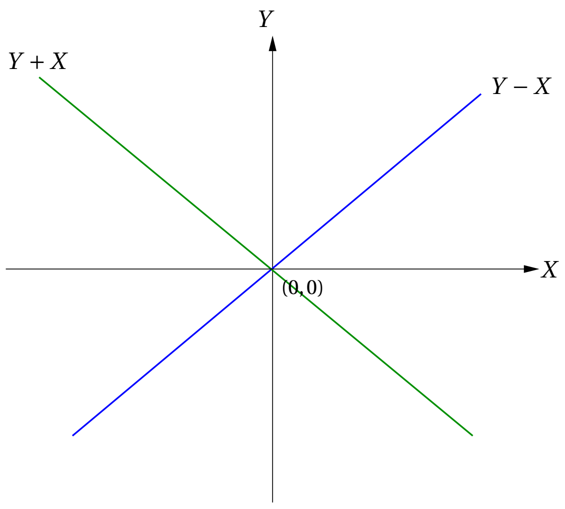


Figure 13.10: Multiplicity of 2

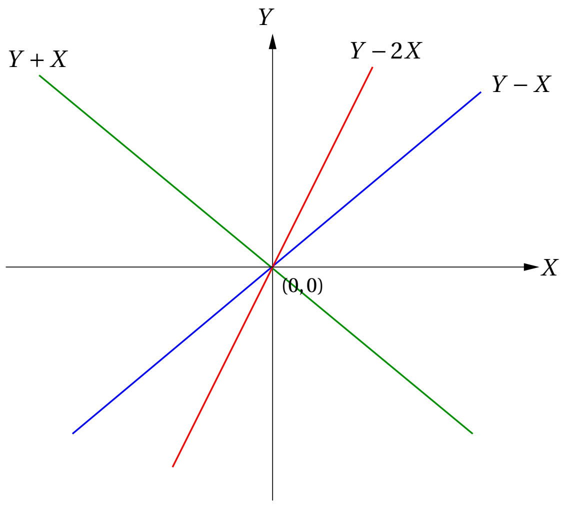


Figure 13.11: Multiplicity of 3

The definition of a root with multiplicity r at a more general point follows from a simple translation:

Definition 13.2.4. $Q(X,Y)$ has r roots at (α,β) if $Q_{\alpha,\beta}(X,Y) \overset{\text{def}}{=} Q(x+\alpha, y+\beta)$ has r roots at $(0,0)$.

Before we state the precise algorithm, we will present the algorithm with an example. Consider the received word in Figure 13.12.

Now we want to interpolate a bivariate polynomial $Q(X,Y)$ with $(1,1)$ degree 5 that “passes twice" through all the 2-D points corresponding to the received word from Figure 13.12. Figure 13.13 shows such an example.

Finally, we want to factorize all the linear factors $Y-P(X)$ of the $Q(X,Y)$ from Figure 13.13. Figure 13.14 shows the five polynomials of degree one are factors of $Q(X,Y)$ from Figure 13.13.

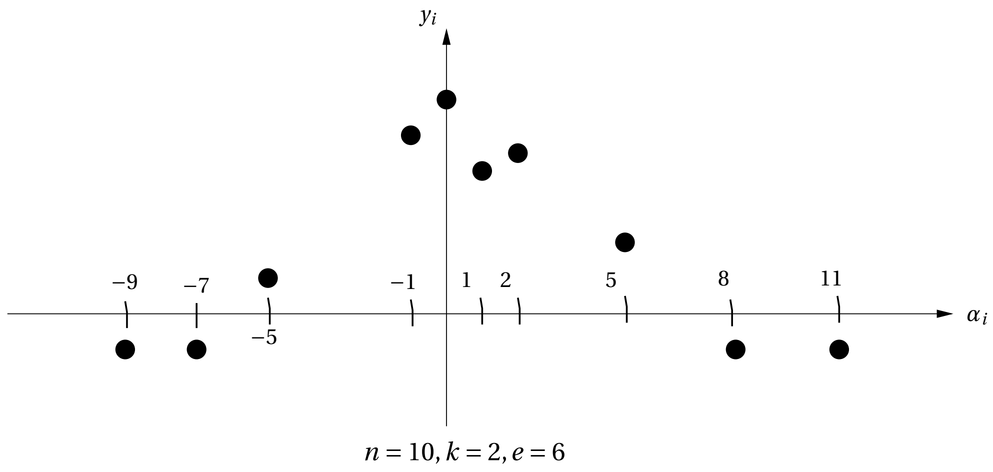


Figure 13.12: An illustration of a received word for the [10,2] Reed-Solomon code (where we have implicitly embedded the field $\mathbb{F}_q$ in the set $\{-9,\dots,11\}$). Here we have considered $e = 6$ errors which is more than what Algorithm 14 can decode. In this case, we are looking for lines that pass through at least 4 points.

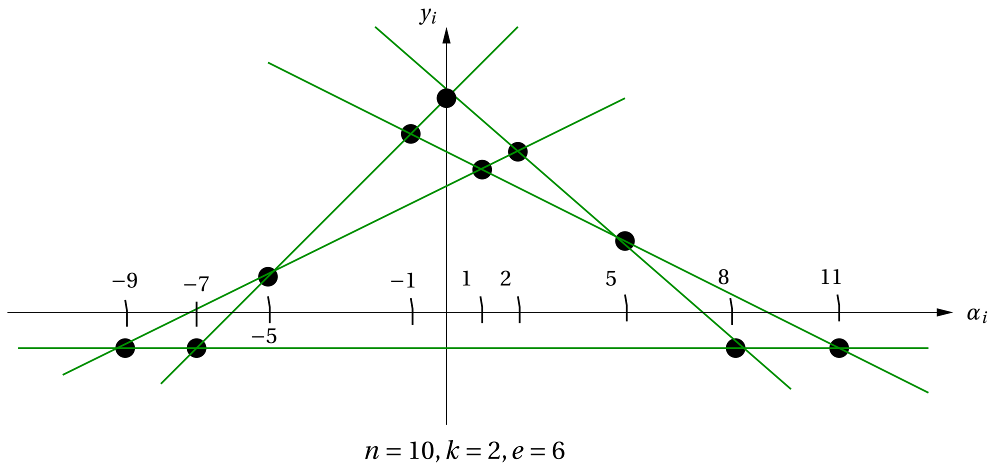


Figure 13.13: An interpolating polynomial $Q(X,Y)$ for the received word in Figure 13.12.

(In fact, $Q(X,Y)$ exactly decomposes into the five lines.)
Algorithm 15 formally states the algorithm.

Correctness of Algorithm 15. To prove the correctness of Algorithm 15, we will need the following two lemmas (we defer the proofs of the lemmas above to Section 13.2.4):

Lemma 13.2.4. *The constraints in (13.12) imply $\binom{r+1}{2}$ constraints for each i on the coefficients of $Q(X,Y)$.*

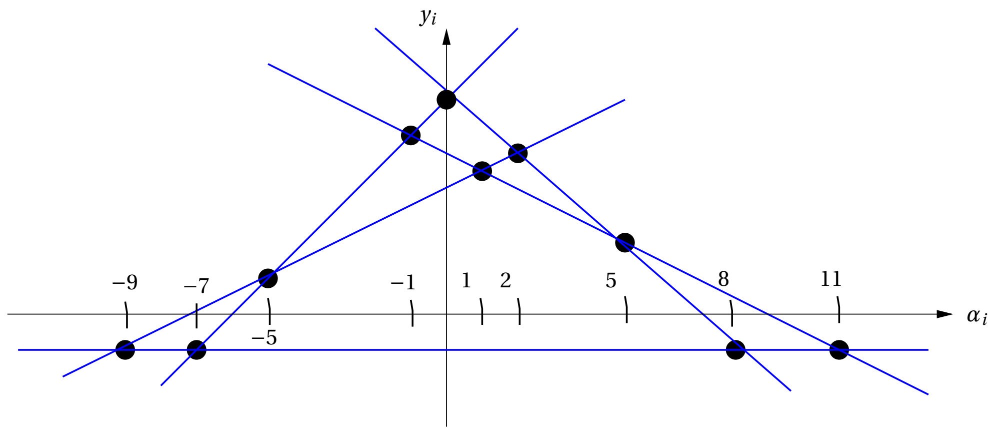


Figure 13.14: The five polynomials that need to be output are shown in blue.

Algorithm 15 The Third List Decoding Algorithm for Reed-Solomon Codes

```
INPUT: $n \ge k \ge 1$, $D \ge 1$, $r \ge 1$, $e = n - t$ and $n$ pairs $\{(\alpha_i, y_i)\}_{i=1}^n$
OUTPUT: (Possibly empty) list of polynomials $P(X)$ of degree at most $k-1$

1: Find a non-zero $Q(X, Y)$ with $(1, k-1)$ degree at most $D$, such that
       $Q(\alpha_i, y_i) = 0$, with multiplicity $r$ for every $1 \le i \le n$.        (13.12)
2: Ł ← ∅
3: FOR every factor $Y - P(X)$ of $Q(X, Y)$ DO
4:     IF $\Delta(\mathbf{y}, (P(\alpha_i))_{i=1}^n) \le e$ and $\deg(P) \le k-1$ THEN
5:         Add $P(X)$ to Ł.
6: RETURN Ł
```

Lemma 13.2.5. *$R(X) \overset{\text{def}}{=} Q(X, P(X))$ has r roots for every i such that $P(\alpha_i) = y_i$. In other words, $(X - \alpha_i)^r$ divides $R(X)$.*

Using arguments similar to those used for proving the correctness of Algorithm 14, to argue the correctness of the interpolations step we will need

$$\frac{D(D+2)}{2(k-1)} > n\binom{r+1}{2},$$

where the LHS is an upper bound on the number of coefficients of $Q(X, Y)$ as before from (13.11) and the RHS follows from Lemma 13.2.4. We note that the choice

$$D = \left\lceil \sqrt{(k-1)nr(r-1)} \right\rceil$$

works. Thus, we have shown the correctness of Step 1.

For the correctness of the root finding step, we need to show that the number of roots of $R(X)$ (which by Lemma 13.2.5 is at least rt) is strictly bigger than the degree of $R(X)$, which from Lemma 13.2.2 is D. That is we would be fine we if have,

$$tr > D,$$

which is the same as

$$t > \frac{D}{r},$$

which in turn will follow if we pick

$$t = \left\lceil \sqrt{(k-1)n\left(1-\frac{1}{r}\right)} \right\rceil.$$

If we pick $r = 2(k-1)n$, then we will need

$$t > \left\lceil \sqrt{(k-1)n - \frac{1}{2}} \right\rceil > \left\lceil \sqrt{(k-1)n} \right\rceil,$$

where the last inequality follows because of the fact that t is an integer. Thus, we have shown

Theorem 13.2.6. *Algorithm 15 can list decode Reed-Solomon codes of rate R from up to $1-\sqrt{R}$ fraction of errors. Further, the algorithm runs in polynomial time.*

The claim on the run time follows from the same argument that was used to argue the polynomial running time of Algorithm 14. Thus, Theorem 13.2.6 shows that Reed-Solomon codes can be efficiently decoded up to the Johnson bound. For an illustration of fraction of errors correctable by the three list decoding algorithms we have seen, see Figure 13.2.3.

A natural question to ask is if Reed-Solomon codes of rate R can be list decoded beyond $1-\sqrt{R}$ fraction of errors. The answer is still not known:

Open Question 13.2.1. *Given a Reed-Solomon code of rate R, can it be efficiently list decoded beyond $1-\sqrt{R}$ fraction of errors?*

Recall that to complete the proof of Theorem 13.2.6, we still need to prove Lemmas 13.2.4 and 13.2.5, which we do next.

Proof of key lemmas

Proof of Lemma 13.2.4. Let

$$Q(X,Y) = \sum_{\substack{i,j \\ i+(k-1)j \le D}} c_{i,j} X^i Y^j$$

and

$$Q_{\alpha,\beta}(X,Y) = Q(X+\alpha, Y+\beta) = \sum_{i,j} c_{i,j}^{\alpha,\beta} X^i Y^j.$$

We will show that

(i) $c_{i,j}^{\alpha,\beta}$ are homogeneous linear combinations of $c_{i,j}$'s.

(ii) If $Q_{\alpha,\beta}(X,Y)$ has no monomial with degree $< r$, then that implies $\binom{r+1}{2}$ constraints on $c_{i,j}^{\alpha,\beta}$'s.

Note that (i) and (ii) prove the lemma. To prove (i), note that by the definition:

$$Q_{\alpha,\beta}(X,Y) = \sum_{i,j} c_{i,j}^{\alpha,\beta} X^i Y^j \tag{13.13}$$

$$= \sum_{\substack{i',j' \\ i'+(k-1)j' \le D}} c_{i',j'}(X+\alpha)^{i'}(Y+\beta)^{j'} \tag{13.14}$$

Note that, if $i > i'$ or $j > j'$, then $c_{i,j}^{\alpha,\beta}$ doesn't depend on $c^{i',j'}$. By comparing coefficients of $X^i Y^j$ from (13.13) and (13.14), we obtain

$$c_{i,j}^{\alpha,\beta} = \sum_{\substack{i'>i \\ j'>j}} c_{i',j'} \binom{i'}{i}\binom{j'}{j} \alpha^i \beta^j,$$

which proves (i). To prove (ii), recall that by definition $Q_{\alpha,\beta}(X,Y)$ has no monomial of degree $< r$. In other words, we need to have constraints $c_{i,j}^{\alpha,\beta} = 0$ if $i + j \le r-1$. The number of such constraints is

$$|\{(i,j) | i + j \le r-1,\ i,j \in \mathbb{Z}^{\ge 0}\}| = \binom{r+1}{2},$$

where the equality follows from the following argument. Note that for every fixed value of $0 \le j \le r-1$, i can take $r-j$ values. Thus, we have that the number of constraints is

$$\sum_{j=0}^{r-1} r - j = \sum_{\ell=1}^{r} \ell = \binom{r+1}{2},$$

as desired. □

We now re-state Lemma 13.2.5 more precisely and then prove it.

Lemma 13.2.7. *Let $Q(X,Y)$ be computed by Step 1 in Algorithm 15. Let $P(X)$ be a polynomial of degree $\le k-1$, such that $P(\alpha_i) = y_i$ for at least $t > \frac{D}{r}$ many values of i, then $Y - P(X)$ divides $Q(X,Y)$.*

Proof. Define

$$R(X) \stackrel{\text{def}}{=} Q(X, P(X)).$$

As usual, to prove the lemma, we will show that $R(X) \equiv 0$. To do this, we will use the following claim.

Claim 13.2.8. *If $P(\alpha_i) = y_i$, then $(X - \alpha_i)^r$ divides $R(X)$, that is α_i is a root of $R(X)$ with multiplicity r.*

Note that by definition of $Q(X, Y)$ and $P(X)$, $R(X)$ has degree $\le D$. Assuming the above claim is correct, $R(X)$ has at least $t \cdot r$ roots. Therefore, by the degree mantra (Proposition 5.2.3), $R(X)$ is a zero polynomial as $t \cdot r > D$. We will now prove Claim 13.2.8. Define

$$P_{\alpha_i, y_i}(X) \stackrel{\text{def}}{=} P(X + \alpha_i) - y_i, \tag{13.15}$$

and

$$\begin{aligned} R_{\alpha_i, y_i}(X) &\stackrel{\text{def}}{=} R(X + \alpha_i) & (13.16)\\ &= Q(X + \alpha_i, P(X + \alpha_i)) & (13.17)\\ &= Q(X + \alpha_i, P_{\alpha_i, y_i}(X) + y_i) & (13.18)\\ &= Q_{\alpha_i, y_i}(X, P_{\alpha_i, y_i}(X)), & (13.19) \end{aligned}$$

where (13.17), (13.18) and (13.19) follow from the definitions of $R(X)$, $P_{\alpha_i, y_i}(X)$ and $Q_{\alpha_i, y_i}(X, Y)$ respectively.

By (13.16) if $R_{\alpha_i, y_i}(0) = 0$, then $R(\alpha_i) = 0$. So, if X divides $R_{\alpha_i, y_i}(X)$, then $X - \alpha_i$ divides $R(X)$. (This follows from a similar argument that we used to prove Proposition 5.2.3.) Similarly, if X^r divides $R_{\alpha_i, y_i}(X)$, then $(X - \alpha_i)^r$ divides $R(X)$. Thus, to prove the lemma, we will show that X^r divides $R_{\alpha_i, y_i}(X)$. Since $P(\alpha_i) = y_i$ when α_i agrees with y_i, we have $P_{\alpha_i, y_i}(0) = 0$. Therefore, X is a root of $P_{\alpha_i, y_i}(X)$, that is, $P_{\alpha_i, y_i}(X) = X \cdot g(X)$ for some polynomial $g(X)$ of degree at most $k - 1$. We can rewrite

$$R_{\alpha_i, y_i}(X) = \sum_{i', j'} c_{i', j'}^{\alpha_i, y_i} X^{i'} (P_{\alpha_i, y_i}(X))^{j'} = \sum_{i', j'} c_{i', j'}^{\alpha_i, y_i} X^{i'} (X g(X))^{j'}.$$

Now for every i', j' such that $c_{i', j'}^{\alpha_i, y_i} \neq 0$, we have $i' + j' \ge r$ as $Q_{\alpha_i, y_i}(X, Y)$ has no monomial of degree $< r$. Thus X^r divides $R_{\alpha_i, y_i}(X)$, since $R_{\alpha_i, y_i}(x)$ has no non-zero monomial X^ℓ for any $\ell < r$. □

13.3 Extensions

We now make some observations about Algorithm 15. In particular, the list decoding algorithm is general enough to solve more general problems than just list decoding. In this section, we present an overview of these extensions.

Recall that the constraint (13.12) states that $Q(X,Y)$ has $r \geq 0$ roots at (α_i, y_i), $1 \leq i \leq n$. However, our analysis did not explicitly use the fact that the multiplicity is same for every i. In particular, given non-zero integer multiplicities $w_i \geq 0$, $1 \leq i \leq n$, Algorithm 15 can be generalized to output all polynomials $P(X)$ of degree at most $k-1$, such that

$$\sum_{i:P(\alpha_i)=y_i} w_i > \sqrt{(k-1)n\sum_{i=0}^{n}\binom{w_i+1}{2}}.$$

(We leave the proof as an exercise.) Note that till now we have seen the special case $w_i = r$, $1 \leq i \leq n$.

Further, we claim that the α_i's need not be distinct for the all of the previous arguments to go through. In particular, one can generalize Algorithm 15 even further to prove the following (the proof is left as an exercise):

Theorem 13.3.1. *Given integer weights $w_{i,\alpha}$ for every $1 \leq i \leq n$ and $\alpha \in \mathbb{F}$, in polynomial time one can output all $P(X)$ of degree at most $k-1$ such that*

$$\sum_i w_{i,P(\alpha_i)} > \sqrt{(k-1)n\sum_{i=0}^{n}\sum_{\alpha\in\mathbb{F}}\binom{w_{i,\alpha}+1}{2}}.$$

This will be useful to solve the following generalization of list decoding called soft decoding.

Definition 13.3.1. Under soft decoding problem, the decoder is given as input a set of non-negative weights $w_{i,d}(1 \leq i \leq n, \alpha \in \mathbb{F}_q)$ and a threshold $W \geq 0$. The soft decoder needs to output all codewords $(c_1, c_2, \ldots, c_n)$ in q-ary code of block length n that satisfy:

$$\sum_{i=1}^{n} w_{i,c_i} \geq W.$$

Note that Theorem 13.3.1 solve the soft decoding problem with

$$W = \sqrt{(k-1)n\sum_{i=0}^{n}\sum_{\alpha\in\mathbb{F}}\binom{w_{i,\alpha}+1}{2}}.$$

Consider the following special case of soft decoding where $w_{i,y_i} = 1$ and $w_{i,\alpha} = 0$ for $\alpha \in \mathbb{F} \setminus \{y_i\}$ $(1 \leq i \leq n)$. Note that this is exactly the list decoding problem with the received word $(y_1, \ldots, y_n)$. Thus, list decoding is indeed a special case of soft decoding. Soft decoding has practical applications in settings where the channel is analog. In such a situation, the "quantizer" might not be able to pinpoint a received symbol y_i with 100% accuracy. Instead, it can use the weight $w_{i,\alpha}$ to denote its confidence level that ith received symbol was α.

Finally, we consider a special case of soft called list recovery, which has applications in designing list decoding algorithms for concatenated codes.

Definition 13.3.2 (List Recovery)**.** Given $S_i \subseteq \mathbb{F}_q$, $1 \leq i \leq n$ where $|S_i| \leq \ell$, output all $[n,k]_q$ codewords $(c_1, \ldots, c_n)$ such that $c_i \in S_i$ for at least t values of i.

We leave the proof that list decoding is a special case of soft decoding as an exercise. Finally, we claim that Theorem 13.3.1 implies the following result for list recovery (the proof is left as an exercise):

Theorem 13.3.2. *Given $t > \sqrt{(k-1)\ell n}$, the list recovery problem with agreement parameter t for $[n,k]_q$ Reed-Solomon codes can be solved in polynomial time.*

13.4 Bibliographic Notes

In 1960, before polynomial time complexity was regarded as an acceptable notion of efficiency, Peterson designed an $O(N^3)$ time algorithm for the unique decoding of Reed-Solomon codes [45]. This algorithm was the first efficient algorithm for unique decoding of Reed-Solomon codes. The Berlekamp-Massey algorithm, which used shift registers for multiplication, was even more efficient, achieving a computational complexity of $O\left(N^2\right)$. Currently, an even more efficient algorithm, with a computational complexity of $O\left(N\text{poly}(\log N)\right)$, is known. This algorithm is the topic of one of the research survey reports for the course this semester.

The Welch-Berlekamp algorithm, covered under US Patent [58], has a running time complexity of $O\left(N^3\right)$. We will follow a description of the Welch-Berlekamp algorithm provided by Gemmell and Sudan in [14].

Håstad, Philips and Safra showed that solving a system of quadratic equations (even those without any square terms like we have in (13.1)) over any filed $\mathbb{F}_q$ is NP-hard [31]. (In fact, it is even hard to approximately solve this problem: i.e. where on tries to compute an assignment that satisfies as many equations as possible.) Linearization is a trick that has been used many times in theoretical computer science and cryptography. See this blog post by Dick Lipton for more on this.

Algorithm 14 is due to Sudan [53] and Algorithm 15 is due to Guruswami and Sudan [28].

It is natural to ask whether Theorem 13.3.2 is tight for list recovery, i.e. generalize Open Question 13.2.1 to list recovery. It was shown by Guruswami and Rudra that Theorem 13.3.2 is indeed the best possible list recovery result for Reed-Solomon codes [23]. Thus, any algorithm that answers Open Question 13.2.1 in some sense has to exploit the fact that in the list decoding problem, the α_i's are distinct.