

ECE 7670

Lecture 7 – Decoding BCH codes

Objective: We will examine several ways of finding the error locator polynomial and using it to decode BCH or RS codes.

Reading:

- Chapter 9.

1 Decoding of BCH and RS codes: the general outline

In decoding BCH or RS codes, there are a few general steps which are always followed:

1. Computation of the syndrome. This is essentially the same for all techniques.
2. Determination of an *error locator polynomial*, whose roots provide an indication of *where* the errors are. There are several different ways of finding the locator polynomial. We will examine several of these. These methods include Peterson's algorithm for BCH codes, the Berlekamp-Massey algorithm for BCH codes; the Peterson-Gorenstein-Zierler algorithm for RS codes, the Berlekamp-Massey algorithm for RS codes, and the Euclidean algorithm. In addition, there are techniques based upon Galois-field Fourier transforms.
3. Finding the roots of the error locator polynomial. This is done using a Chien search, described below.
4. For RS codes, the error *values* must also be determined.

Note: throughout this lecture, the codes are assumed to be *narrow* sense BCH or RS codes. There are a few modifications that must take place to deal with non-narrow sense codes.

2 Computation of the syndrome

Since

$$g(\alpha) = g(\alpha^2) = \dots = g(\alpha^{2t}) = 0$$

it follows that for a codeword $\mathbf{c} = (c_0, \dots, c_{n-1})$ with polynomial $c(x) = c_0 + \dots + c_{n-1}x^{n-1}$ has

$$c(\alpha) = \dots = c(\alpha^{2t}) = 0.$$

For a received polynomial $r(x) = c(x) + e(x)$ we have

$$S_j = r(\alpha^j) = e(\alpha^j) = \sum_{k=0} e_k \alpha^{jk}, \quad j = 1, 2, \dots, 2t.$$

Thus, we get our syndrome by evaluating the received code polynomial at the values of α that are the roots of the generator.

Suppose that \mathbf{r} has ν errors in it which are at locations i_1, i_2, \dots, i_ν . The error values in these locations are e_{i_j} . Then

$$S_j = \sum_{i=1}^{\nu} e_{i_j} (\alpha^j)^{i_i} = \sum_{i=1}^{\nu} e_{i_j} (\alpha^{i_i})^j.$$

Let

$$X_l = \alpha^{i_l}.$$

Then we can write

$$S_j = \sum_{i=1}^{\nu} e_{i_j} X_l^j \quad j = 1, 2, \dots, 2t.$$

For *binary codes* we have $e_{i_j} = 1$ (if there is a non-zero error, it must be to 1). Then we have

$$S_j = \sum_{i=1}^{\nu} X_l^j. \quad (1)$$

Notice the following: if we know X_l , then we know the location of the error. For example, suppose we know that $X_1 = \alpha^4$. This means, by the definition of X_l that $i_1 = 4$; i.e., the error is in the received digit r_4 . For binary codes, this kind of information is all that we need to know. We call the X_l the **error locators**. We therefore want to find a way of finding all the X_l as the first step to decoding.

3 The error locator polynomial: binary codes

From (1) we obtain the following equations:

$$\begin{aligned} S_1 &= X_1 + X_2 + \dots + X_\nu \\ S_2 &= X_1^2 + X_2^2 + \dots + X_\nu^2 \\ &\vdots \\ S_{2t} &= X_1^{2t} + X_2^{2t} + \dots + X_\nu^{2t} \end{aligned}$$

The equations are said to be *power-sum symmetric functions*. This gives us $2t$ equations in the ν unknown error locators. In principle this could be solved by an exhaustive search. In practice, the solution cannot be obtained from these equations since they are nonlinear. We will employ some (very tricky) tricks to find linear equations for the error locators.

We begin the series of tricks by defining the **error locator polynomial**:

$$\Lambda(x) = \prod_{l=1}^{\nu} (1 - X_l x) = \Lambda_\nu x^\nu + \Lambda_{\nu-1} x^{\nu-1} + \dots + \Lambda_1 x + \Lambda_0.$$

Notice that if there is a value of x that is a root of $\Lambda(x)$ that it is the **reciprocal** (in the field) of the error locator. For example, if in $GF(16)$ we have $x_0 = \alpha^4$ as a root of $\Lambda(x)$, then there must be an error locator (say X_1) such that

$$X_1 \alpha^4 = 1,$$

or $X_1 = \alpha^{11}$. From this we conclude that there is an error in r_{11} .

4 Chien search

Assume for the moment that we actually have the error locator polynomial. (We will find out later how to get it.) Our next step is to find the roots of the error locator. Being in a finite field, we can examine every element of the field to determine if it is a root. (Other tricks, like Newton's methods cannot work.)

To be specific, suppose that our error locator polynomial is

$$\Lambda(x) = \Lambda_0 + \Lambda_1x + \Lambda_2x^2 + \Lambda_3x^3 = \Lambda_1x + \Lambda_2x^2 + \Lambda_3x^3$$

We try each value in succession: $x = 1, x = \alpha, x = \alpha^2, \dots$. This gives us the following:

$$\begin{aligned}\Lambda(1) &= 1 + \Lambda_1(1) + \Lambda_2(1)^2 + \Lambda_3(1)^3 \\ \Lambda(\alpha) &= 1 + \Lambda_1(\alpha) + \Lambda_2(\alpha)^2 + \Lambda_3(\alpha)^3 \\ \Lambda(\alpha^2) &= 1 + \Lambda_1(\alpha^2) + \Lambda_2(\alpha^2)^2 + \Lambda_3(\alpha^2)^3 \\ &\vdots\end{aligned}$$

We can simplify these by computing

$$\begin{aligned}\Lambda_1(1) + \Lambda_2(1)^2 + \Lambda_3(1)^3 \\ \Lambda_1(\alpha) + \Lambda_2(\alpha)^2 + \Lambda_3(\alpha)^3 \\ \Lambda_1(\alpha^2) + \Lambda_2(\alpha^2)^2 + \Lambda_3(\alpha^2)^3 \\ \vdots\end{aligned}$$

If a result is equal to 1, that means the error locator has a root. These computations can be done efficiently as follows. Set up registers containing $\Lambda_1, \Lambda_2, \dots, \Lambda_t$ as multipliers, connected to blocks containing $\alpha, \alpha^2, \dots, \alpha^t$, as shown. Each time the system of multipliers is clocked, the registers are loaded with the necessary values.

5 Finding the error locator: Peterson's approach

Let us return to the power-sum symmetric functions:

$$\begin{aligned}S_1 &= X_1 + X_2 + \dots + X_\nu \\ S_2 &= X_1^2 + X_2^2 + \dots + X_\nu^2 \\ &\vdots \\ S_{2t} &= X_1^{2t} + X_2^{2t} + \dots + X_\nu^{2t}\end{aligned}$$

As mentioned, these are nonlinear equations. However, we can find a *linear* relation of the syndromes to the coefficients of the error locator polynomial. We note that in

$$\Lambda(x) = \prod_{l=1}^{\nu} (1 - X_l x) = \Lambda_\nu x^\nu + \Lambda_{\nu-1} x^{\nu-1} + \dots + \Lambda_1 x + \Lambda_0$$

we must have

$$\begin{aligned}
 \Lambda_0 &= 1 \\
 \Lambda_1 &= X_1 + X_2 + \cdots + X_\nu \\
 \Lambda_2 &= \sum_{i < j} = X_1 X_2 + X_1 X_3 + \cdots + X_{\nu-1} X_\nu \\
 \Lambda_3 &= \sum_{i < j < k} X_i X_j X_k = X_1 X_2 X_3 + X_1 X_2 X_4 + \cdots + X_{\nu-2} X_{\nu-1} X_\nu \\
 &\vdots \\
 \Lambda_\nu &= \prod_{i=1}^{\nu} X_i = X_1 X_2 \cdots X_\nu.
 \end{aligned}$$

These equations are the *elementary symmetric functions*. There is an interesting relationship between the power-sum symmetric functions and the elementary symmetric functions known as *Newton's identities*.

5.1 Newton's identities

Newton's identities relate the coefficients of a polynomial to the power sum identities obtained from the roots of the polynomial. We will derive them here in the general case, then make application to the error locator polynomial.

Let $f(z)$ be a polynomial of degree n with roots z_1, z_2, \dots, z_n and coefficients a_0, a_1, \dots, a_n . Then we can write

$$f(z) = \prod_{i=1}^n (z - z_i) = a_0 + a_1 z + a_2 z^2 + \cdots + a_n z^n.$$

There are several steps in the development.

1. We first note that

$$f(z) \sum_{i=1}^n \frac{1}{z - z_i} = f'(z). \quad (2)$$

This is straightforward:

$$\frac{d}{dz} \log f(z) = \frac{f'(z)}{f(z)} = \sum_{i=1}^n \frac{1}{z - z_i}.$$

2. For z sufficiently large we can write

$$\begin{aligned}
 \frac{1}{z - z_i} &= \frac{z^{-1}}{1 - (z_i/z)} = z^{-1} (1 + (z_i/z) + (z_i/z)^2 + (z_i/z)^3 + \cdots) \\
 &= z^{-1} + z_i z^{-2} + z_i^2 z^{-3} + z_i^3 z^{-4} + \cdots.
 \end{aligned}$$

Thus

$$\sum_{i=1}^n \frac{1}{z - z_i} = n z^{-1} + s_1 z^{-2} + s_2 z^{-3} + \cdots \quad (3)$$

where

$$s_j = \sum_{i=1}^n z_i^j.$$

Note that $s_0 = n$. The s_j are the *power-sum symmetric functions*.

3. Now substitute (3) into (2), expand, and equate coefficients of equal powers. For the coefficients of negative powers of z we obtain the following:

$$\begin{aligned} z^{-1} : a_0 n + a_1 s_1 + a_2 s_2 + \cdots + a_n s_n &= 0 \\ z^{-2} : a_0 s_1 + a_1 s_2 + a_2 s_3 + \cdots + a_n s_{n+1} &= 0 \\ &\vdots \end{aligned}$$

In general, for $j = 0, 1, \dots$, we have

$$a_0 s_j + a_1 s_{j+1} + \cdots + a_n s_{n+j} = 0. \quad (4)$$

This is the first of the Newton identities.

For positive powers of z we have the following:

$$\begin{aligned} z^0 : a_1 n + a_2 s_1 + \cdots + a_n s_{n-1} &= a_1 \\ &\text{or } a_1(n-1) + a_2 s_1 + \cdots + a_n s_{n-1} = 0 \\ z^1 : a_2 n + a_3 s_1 + \cdots + a_n s_{n-2} &= 2a_2 \\ &\text{or } a_2(n-2) + a_3 s_1 + \cdots + a_n s_{n-2} = 0 \\ z^2 : a_3 n + a_4 s_1 + \cdots + a_n s_{n-3} &= 3a_3 \\ &\text{or } a_3(n-3) + a_4 s_1 + \cdots + a_n s_{n-3} = 0 \\ &\vdots \\ z^{n-1} : a_{n-1} n + a_n s_1 &= (n-1)a_{n-1} \\ &\text{or } a_{n-1}(1) + a_n s_1 = 0. \end{aligned}$$

In general we can write for $m = 1, 2, \dots, n-1$

$$a_n s_m + \underbrace{a_{n-1} s_{m-1} + \cdots + a_{n-m+1} s_1}_{m-1 \text{ terms}} + m a_{n-m} = 0.$$

This is the second of the Newton identities.

Notice that the first Newton identity is equal to the second Newton identity when $j = 0$ in the first and $m = n$ in the second.

To apply the Newton identities to our error locator polynomial

$$\Lambda(x) = \prod_{l=1}^{\nu} (1 - X_l s)$$

we let $f(z) = x^\nu \Lambda(1/x) = \Lambda_0 x^\nu + \Lambda_1 x^{\nu-1} + \cdots + \Lambda_n$, so that

$$\Lambda_i = a_{\nu-i}.$$

Applying the Newton's identities, we obtain the following:

$$\begin{aligned} \Lambda_0 S_1 + \Lambda_1 &= 0 \\ \Lambda_0 S_2 + \Lambda_1 S_1 + 2\Lambda_2 &= 0 \\ \Lambda_0 S_3 + \Lambda_1 S_2 + \Lambda_2 S_1 + 3\Lambda_3 &= 0 \\ &\vdots \\ \Lambda_0 S_\nu + \Lambda_1 S_{\nu-1} + \Lambda_2 S_{\nu-2} + \cdots + \Lambda_{n-1} S_1 + \nu\Lambda_n &= 0 \\ S_{\nu+1} \Lambda_0 + \Lambda_1 S_\nu + \Lambda_2 S_{\nu-1} + \cdots + \Lambda_n S_1 &= 0 \\ &\vdots \\ \Lambda_0 S_{2t} + \Lambda_1 S_{2t-1} + \Lambda_2 S_{2t-2} + \cdots + \Lambda_n S_{2t-\nu} &= 0. \end{aligned}$$

If we know the syndromes $S_j, j = 1, 2, \dots, 2t$, we now have a linear set of equations for the coefficients of the error locator polynomial.

Note that $\Lambda_0 = 1$. For binary codes we can make some further simplifications. $nS_j = 0$ if n is even and $nS_j = S_j$ if n is odd. Furthermore we have

$$S_{2j} = S_j^2.$$

Assume that $\nu = t$ errors have occurred. We can write t equations as

$$\begin{aligned} S_1 + \Lambda_1 &= 0 \\ S_3 + \Lambda_1 S_2 + \Lambda_2 S_1 + \Lambda_3 &= 0 \\ &\vdots \\ S_{2t-1} + \Lambda_1 S_{2t-2} + \cdots + \Lambda_t S_{t-1} &= 0. \end{aligned}$$

We form the matrix equation

$$\begin{bmatrix} 1 & & & & & & & \\ S_2 S_1 & 1 & & & & & & \\ S_4 & S_3 & S_2 & S_1 & & & & \\ \vdots & & & & & & & \\ S_{2t-4} & S_{2t-5} & S_{2t-6} & \cdots & S_{t-2} & S_{t-3} & & \\ S_{2t-2} & S_{2t-3} & S_{2t-4} & \cdots & S_t & S_t & & \end{bmatrix} \begin{bmatrix} \Lambda_1 \\ \Lambda_2 \\ \vdots \\ \Lambda_t \end{bmatrix} = \begin{bmatrix} -S_1 \\ -S_3 \\ \vdots \\ -S_{2t-1} \end{bmatrix}.$$

If there are in fact t errors, the matrix is invertible, as we can determine by computing the determinant of the matrix. If it is not invertible, remove two rows and columns, and try again. Once Λ is found, we find its roots. If the roots are distinct and all lie in the right field, then we use these to determine the error locations. If they are not distinct or lie in the wrong field, then the received word is not within distance t of any code word. This means that we have a *decoder failure*.

For small numbers of errors, we can provide explicit formulas for the Λ s.

Single-error $\Lambda_1 = S_1$.

Double Error $\Lambda_1 = S_1$ and

$$\Lambda_2 = \frac{S_3 + S_1^3}{S_1}$$

etc.

This algorithm is, in general, quite complex. Computing the sequence of determinants to find the number of errors is costly. So is solving the system of equations once the number of errors is determined. We therefore look for more efficient techniques.

6 Berlekamp-Massey Algorithm

We observe from a Newton's identity that

$$\Lambda_n S_j + \Lambda_{n-1} S_{j+1} + \cdots + \Lambda_0 S_{j+\nu} = 0 \quad j = 0, 1, \dots$$

We can re-write this as

$$S_k = - \sum_{j=1}^{\nu} S_{k-j} \Lambda_j, \quad k = \nu + 1, \nu + 2, \dots, 2\nu.$$

This formula describes the output of a linear feedback shift register (LFSR) with coefficients $\Lambda_1, \Lambda_2, \dots, \Lambda_\nu$. In order for this formula to work, we must find the

Λ_j coefficients in such a way that the LFSR generates the known sequence of syndromes. Furthermore, we want the *shortest* such LFSR. Note that we can also write

$$\begin{bmatrix} S_1 & S_2 & S_3 & \cdots & S_\nu \\ S_2 & S_3 & S_4 & \cdots & S_{\nu+1} \\ S_3 & S_4 & S_5 & \cdots & S_{\nu+2} \\ \vdots & & & & \\ S_\nu & S_{\nu+1} & S_{\nu+2} & \cdots & S_{2\nu-1} \end{bmatrix} \begin{bmatrix} \Lambda_\nu \\ \Lambda_{\nu-1} \\ \Lambda_{\nu-2} \\ \vdots \\ \Lambda_1 \end{bmatrix} = \begin{bmatrix} -S_{\nu+1} \\ -S_{\nu+2} \\ -S_{\nu+3} \\ \vdots \\ -S_{2\nu} \end{bmatrix}$$

The matrix is a *Hankel* matrix — constant on the backward diagonals. The algorithm we will describe thus points the way to a fast algorithm for solving Hankel systems of equations.

7 Application of the Berlekamp-Massey algorithm

In doing the decoding, we find the error locator polynomial using the B-M algorithm. (Show overhead).

There is a useful simplification for binary codes. In this case, every other discrepancy is equal to zero. There is thus never an update on these steps of the B-M algorithm, and they can be merged into the next step. This saves half the computation time.

8 Non-binary RS decoding

Having found the error-locator polynomial and its roots, there is still one more step for the non-binary RS decoder: we have to find the error values. We will find a way to determine the error values. Let us return to the syndrome,

$$S_j = \sum_{l=1}^{\nu} e_{i_l} X_l^j, \quad j = 1, 2, \dots, 2t.$$

If we know the error locators, one solution to the problem is to set up and solve a set of linear equations:

$$\begin{bmatrix} X_1 & X_2 & X_3 & \cdots & X_\nu \\ X_1^2 & X_2^2 & X_3^2 & \cdots & X_\nu^2 \\ \vdots & & & & \\ X_1^{2t} & X_2^{2t} & X_3^{2t} & \cdots & X_\nu^{2t} \end{bmatrix} \begin{bmatrix} e_{i_1} \\ e_{i_2} \\ \vdots \\ e_{i_\nu} \end{bmatrix} = \begin{bmatrix} S_1 \\ S_2 \\ \vdots \\ S_{2t} \end{bmatrix}$$

However, there is a method which is computationally easier and in addition provides us a key insight for another way of doing the decoding. This method is known as *Forney's algorithm*. (Forney was another really smart guy!)

We define the *syndrome polynomial* as

$$S(x) = S_1x + S_2x^2 + \cdots + S_{2t}x^{2t}.$$

(The book has some rubbish about infinite-degree syndrome polynomials; not only do they not exist, but they lead the wrong direction.)

Let us define a new polynomial known as the *error-evaluator polynomial* by

$$\Omega(x) = (1 + S(x))\Lambda(x) \pmod{x^{2t+1}}$$

Comment (as an aside): Some authors define

$$S(x) = S_1 + S_2x + \cdots + S_{2t}x^{2t-1},$$

in which case they define

$$\Omega(x) = S(x)\Lambda(x) \pmod{x^{2t}}$$

Provided that you keep everything straight, it doesn't matter which way you define things. We will follow our book's convention.

Fact:

$$(1 - x^{2t+1}) = (1 - x)(1 + x + x^2 + \dots + x^{2t}) = (1 - x) \sum_{j=0}^{2t} x^j$$

Observe:

$$\begin{aligned} S(x)\Lambda(x) \pmod{x^{2t+1}} &= \left(\sum_{j=1}^{2t} \sum_{l=1}^{\nu} e_{i_l} X_l^j x^j \right) \prod_{i=1}^{\nu} (1 - X_i x) \pmod{x^{2t+1}} \\ &= \sum_{l=1}^{\nu} e_{i_l} \sum_{j=1}^{2t} (X_l x)^j \prod_{i=1}^{\nu} (1 - X_i x) \pmod{x^{2t+1}} \\ &= \sum_{l=1}^{\nu} e_{i_l} (1 - X_l x) \sum_{j=1}^{2t} (X_l x)^j \prod_{i \neq l}^{\nu} (1 - X_i x) \pmod{x^{2t+1}} \end{aligned}$$

Now observe that

$$(1 - X_l x) \sum_{j=1}^{2t} (X_l x)^j = (X_l x) - (X_l x)^{2t+1}$$

from the fact before. Since $(X_l x)^{2t+1} \equiv 0 \pmod{x^{2t+1}}$ we have

$$S(x)\Lambda(x) = \sum_{l=1}^{\nu} e_{i_l} X_l x \prod_{i \neq l}^{\nu} (1 - X_i x) \pmod{x^{2t+1}}$$

We can therefore write

$$\Omega(x) = (1 + S(x))\Lambda(x) \pmod{x^{2t+1}} \tag{5}$$

This equation is known as the *key equation*.

We make the following observation (needed below):

1. The degree of $\Lambda(x)$ is always $\leq t$ (we can correct only up to t errors).
2. Removing the $S(x)$ part, the degree of $\Omega(x)$ is always $\leq t - 1$ (from the definition).

We now have the following for finding the error values:

Theorem 1 (*Forney's algorithm*)

$$e_{i_k} = -\frac{X_k \Omega(X_k^{-1})}{\Lambda'(X_k^{-1})}$$

where $\Lambda'(x)$ is the formal derivative of $\Lambda(x)$.

Proof We can write

$$\Lambda'(x) = \frac{d}{dx} \prod_{l=1}^{\nu} (1 - X_l x) = - \sum_{l=1}^{\nu} X_l \prod_{j \neq l}^{\nu} (1 - X_j x)$$

Then

$$\Lambda'(X_k^{-1}) = -X_k \prod_{j \neq k} (1 - X_j X_k^{-1}).$$

Now substitute X_k^{-1} in $\Omega(x)$:

$$\Omega(X_k^{-1}) = \Lambda(X_k^{-1}) + \sum_{l=1}^{\nu} e_{i_l} X_l X_k^{-1} \prod_{j \neq l} (1 - X_j X_k^{-1}).$$

Then since the error locator X_k is the reciprocal of a root of $\Lambda(x)$ we have $\Lambda(X_k^{-1}) = 0$. Also, every product has a zero in it, except the one having $j \neq k$. We obtain

$$\Omega(X_k^{-1}) = e_{i_k} \prod_{j \neq k} (1 - X_j X_k^{-1}) = -e_{i_k} \Lambda'(X_k^{-1}).$$

□

Example 1 Suppose $S(x) = (\alpha^6 + \alpha^3 x^2 + \alpha^4 x^3 + \alpha^3 x^4)$, and we find (say using the B-M algorithm) that

$$\Lambda(x) = 1 + \alpha^2 x + \alpha x^2.$$

in the code where $t = 2$. Then

$$\Omega(x) = (\alpha^6 + \alpha^3 x^2 + \alpha^4 x^3 + \alpha^3 x^4)(1 + \alpha^2 x + \alpha x^2) \pmod{x^5} = (1 + x + \alpha^3 x^2).$$

Then

$$\Lambda'(x) = \alpha^2 + 2\alpha x = \alpha^2.$$

So

$$e_{i_k} = -\frac{X_k(1 + X_k^{-1} + \alpha^3 X_k^{-2})}{\alpha^2} = \alpha^5 X_k + \alpha^5 + \alpha X_k^{-1}$$

The error locators (reciprocal roots of $\Lambda(x)$) are $X_1 = \alpha^3$ and $X_2 = \alpha^5$, so the errors are at $i_1 = 3$ and $i_2 = 5$. We find that

$$e_3 = \alpha^5 \alpha^3 + \alpha^5 + \alpha \alpha^4 = \alpha$$

and

$$e_5 = \alpha^5 \alpha^5 + \alpha^5 + \alpha \alpha^2 = \alpha^5.$$

The error polynomial is $e(x) = \alpha x^3 + \alpha^5 x^5$. □

9 Euclidean algorithm for BCH decoding

We return to the key equation:

$$\Omega(x) = (1 + S(x))\Lambda(x) \pmod{x^{2t+1}}$$

Recall what “mod” means:

$$a \equiv b \pmod{c}$$

means that

$$c|(a - b) \text{ or } c|(b - a).$$

which is to say that some multiple of c is equal to $b - a$:

$$cd = (b - a)$$

or

$$cd + a = b.$$

From key equation, we see that we have

$$\Theta(x)(x^{2t+1}) + \Lambda(x)[1 + S(x)] = \Omega(x)$$

for some $\Theta(x)$ whose value we don't really care about. In fact, all we really know to begin with is x^{2t+1} and $1 + S(x)$.

Now recall that the Euclidean algorithm returns, for a pair of elements (a,b) from a Euclidean domain, a pair of numbers x, y such that

$$ax + by = c,$$

where c is the GCD of a and b . In our case, we run the Euclidean algorithm (more precisely, the extended Euclidean algorithm) to obtain a sequence of polynomials $\Theta^{[k]}(x)$, $\Lambda^{[k]}(x)$ and $\Omega^{[k]}(x)$ satisfying

$$\Theta^{[k]}(x)x^{2t+1} + \Lambda^{[k]}(x)(1 + S(x)) = \Omega^{[k]}(x).$$

We step through the algorithm until $\Omega^{[k]}(x)$ has degree less than the degree of $\Lambda^{[k]}(x)$. (overhead)

In terms of computational efficiency, it appears that the B-M procedure is slightly better than the Euclidean algorithm, since the B-M method build longer polynomials, but the Euclidean algorithm deals with the whole $S(x)$. Probably they are very similar. Also, we get $\Omega(x)$ as a useful byproduct of the Euclidean algorithm method. (However, it is also possible to modify B-M so that Ω is also found simultaneously.)

10 Frequency-domain decoding

Let $\mathbf{r} = \mathbf{c} + \mathbf{e}$ be a received vector, and take the GF-FT:

$$\mathbf{R} = \mathbf{C} + \mathbf{E}.$$

The first (for a narrow-sense code) $2t$ coordinates of the transform are the syndromes. (Can we use this to do fast computation of the syndromes?) We have

$$E_j = \sum_{i=0}^{n-1} \alpha^{ij} e_i$$

Knowing the first $2t$ coordinates of E (which are the syndromes, we need some way of finding the remaining $n - 2t$ coordinates of E , after which we can find \mathbf{e} by inverse GF-FT.

For $\Lambda(x) = \prod_{i=1}^{\nu} (1 - X_i x)$, treating the coefficients as a spectrum, the inverse transform yields a vector $\boldsymbol{\lambda}$ which has zeros at the coordinates corresponding to the zeros of $\Lambda(x)$, so $\boldsymbol{\lambda}$ has a zero wherever \mathbf{e} is nonzero:

$$\lambda_i e_i = 0.$$

Translating the product back to the frequency domain, we have the convolution formula

$$\sum_{k=0}^{n-1} \Lambda_k E_{j-k} = 0, \quad j = 0, 1, \dots, n-1.$$

If we find Λ (using any of the techniques discussed before) then we can find the remaining coordinates of \mathbf{E} and hence \mathbf{e} .

11 Erasure decoding

An erasure is an error in which the location is known, but the value of the error is not. Sometimes we can examine a received signal and perceive that it is not within acceptable bounds and hence determine that some noise has affected the symbol. We then declare an erasure.

In terms of code performance we have the following. For a code with d_{\min} , suppose there is a single erasure. Over the $n-1$ unerased coordinates, the codewords are separated by a distance of at least $d_{\min} - 1$. (Think in terms of the minimum weight.) More generally, if there are f erased symbols, then the distance among the remaining digits is at least $d_{\min} - f$. Hence we can correct

$$t_e = \lfloor (d_{\min} - f - 1)/2 \rfloor$$

(the book has a typo). We can correct e errors and f erasures if $2e + f < d_{\min}$.

11.1 Binary erasure decoding

1. Place zeros in all erased coordinates and decode; label the result \mathbf{c}_0 .
2. Place ones in all erased coordinates and decode; label the result \mathbf{c}_1 .
3. Find which of \mathbf{c}_0 and \mathbf{c}_1 is closest to \mathbf{r} . This is the output code.

Suppose we have $(2e + f) < d_{\min}$ (so we can correct ok). Assign zero to f coordinates; we thereby generated e_0 errors, $e_0 < f$ so that the total number of errors is $(e_0 + e)$. Assign one to the f coordinates, we make e_1 errors, $e_1 < f$, and $e_0 + e_1 = f$. In this case we have $(e + e_1)$ errors. Since either e_0 or e_1 is $\leq f/2$, then either

or

$$2(e + e_1) \leq 2(e + f/2)$$

where $2(e + f/2) < d_{\min}$, so that one or the other must decode correctly.

11.2 Nonbinary erasures

Let the received word have ν errors and f erasures, with the errors at i_1, i_2, \dots, i_ν and the erasures at j_1, j_2, \dots, j_f . Error locators as before, X_1, X_2, \dots, X_ν , with $X_k = \alpha^{i_k}$. Introduce erasure locators

$$Y_1 = \alpha^{j_1} \quad Y_2 = \alpha^{j_2} \quad \dots \quad Y_f = \alpha^{j_f}$$

Goal: find locators X_k , values e_{i_k} at the error locations, and values at the erasures f_{j_k} .

Create erasure locator polynomial

$$\Gamma(x) = \prod_{l=1}^f (1 - Y_l x)$$

To get started, we need to compute syndromes, and we need a value at the erasure locations to compute it. We place in (for the moment) the value 0 at the erasure locations. Then

$$S_l = r(\alpha^l) \Big|_{\text{zeros at erasures}} = \sum_{k=1}^{\nu} e_{i_k} X_k^l + \sum_{k=1}^f f_{i_k} Y_k^l.$$

Then let

$$S(x) = \sum_{l=1}^{2t} S_l x^l$$

and create the *key equation*

$$\Lambda(x)\Gamma(x)[1 + S(x)] = \Omega(x) \pmod{x^{2t+1}}.$$

Let us now define

$$1 + \Xi(x) = \Gamma(x)[1 + S(x)] \pmod{x^{2t+1}}.$$

So the key equation becomes

$$\Lambda(x)[1 + \Xi(x)] = \Omega(x) \pmod{x^{2t+1}}.$$

Since we know $S(x)$ and $\Gamma(x)$, we can find $\Xi(x)$. Thus the problem boils down to being as before. Given $\Xi(x)$, we can apply either B-M or Euclid to find $\Lambda(x)$. Having found the error locators, we then need to find the error magnitudes and erasure magnitudes.

After knowing $\Lambda(x)$, we create

$$\Phi(x) = \Lambda(x)\Gamma(x).$$

Then, much as before, we can use a modified Forney's algorithm to find

$$e_{i_k} = -\frac{X_k \Omega(X_k^{-1})}{\Phi'(X_k^{-1})}$$

$$f_{i_k} = -\frac{Y_k \Omega(X_k^{-1})}{\Phi'(X_k^{-1})}$$