Finite fields and Linear codes

Suayb S. Arslan

"*Although this may seem a paradox, all exact science is dominated by the idea of approximation.*" R. Bertrand.

Version 0.1 – October 2013

Our previous discussion was on the general code constructions. Coding bounds that are developed previously apply to any code that has a block length of n symbols and information length of k symbols. However, we have not given any particular construction and/or mapping between message symbols to coded symbols as of yet. In otherwords, the question of practical construction of channel codes has not been addressed. In order for these codes to find application in real life, codes with good distance properties should be devised. In addition, low complexity designs are of significant interest to the electronic chip designers. In this section, we will try to concentrate on a subclass of codes, called linear codes. We will later narrow down our attention to even smaller subclasses of linear codes, called cyclic codes and cover their interesting properties that lead to practical code constructions/implementations. In order for reader to digest the material, the document will begin by giving background information about finite fields, linear spaces/subspaces through resorting to algebra fundamentals, before describing the elegant linear code constructions based on these mathematical principles.

1 FINITE FIELDS

Linear codes are conventionally defined over the elements of Galois Fields. A Galois Field with parameter q, denoted as $GF(q)$, is a finite set of elements on which two major operations, addition and multiplication are defined. In order to satisfy the following axioms of field, q must be a prime number of a power of a prime number.

- **Closure** $\forall a,b \in GF(q), a+b \in GF(q)$ and $\forall a,b \in GF(q), ab \in GF(q)$

- **Commutativity** $\forall a,b \in GF(q), a+b=b+a$ and $\forall a,b \in GF(q), ab=ba$
- **Associativity** $\forall a,b,c \in GF(q), (a+b)+c=a+(b+c)$ and $\forall a,b,c \in GF(q), (ab)c=a(bc)$
- **Identity** $\forall a \in GF(q), \exists e \in GF(q), a+e_0=a$ and $\forall a \in GF(q), \exists e_1 \in GF(q), ae_1=a$
- **Inverse** $\forall a \in GF(q), \exists (-a) \in GF(q), a+(-a)=e_0$ and $\forall a \in GF(q), \exists a^{-1} \in GF(q), a(a^{-1})=e_1$
- **Distributivity** $\forall a,b,c \in GF(q), a(b+c)=ab+ac$.

As can be noticed, these set of axioms implicitly define the subtraction and division as well over the elements of $GF(q)$. Since q is prime, $GF(p)$ is referred to as prime field which has the elements $\{0,1,2,\ldots,q-1\}$. Since for $a,b \in GF(q)$, $a+b \in GF(q)$ and $ab \in GF(q)$, we define $(a+b) \mod q$ and $ab \mod q$ in order to stay in the same field after addition and multiplication operations.

Definition 1: *A primitive element α of $GF(q)$ is an element such that every field element except zero can be expressed as a power of α.*

For example, 2 is a primitive element of $GF(5)$ under modular arithmetic operations. Given this definition, we can check that 3 is also a primitive element of the same field. In order to construct the extension field $GF(q^m)$, we need to resort to polynomial algebra. Addition and multiplication of polynomials over $GF(q)$ is defined as in the real number system except the computations for coefficients of the polynomials are performed according to corresponding definitions of addition and multiplication of $GF(q)$. An *irreducible polynomial* cannot be factorized to polynomials over $GF(q)$.

Definition 2: *Period of a polynomial $f(x)$ is defined as the solution to the following optimization problem*

$$per(f(x)) \triangleq \min_e f(x) | (x^e - 1) \tag{1.1}$$

where $a|b$ means a divides b without any reminder.

Definition 3: *A primitive polynomial that can generate the extension field $GF(q^m)$ is an irreducible polynomial of degree m (also known as prime polynomial in the literature) whose period is q^m-1.*

Let $p(x)=\sum_{i=0}^{m} p_i x^i$ be a primitive polynomial over $GF(q)$ with $p_m=1$, and let $\alpha \in GF(q)$ be a root of $p(x)$. The root of the primitive polynomial is known as the primitive element of the extension field $GF(q^m)$. We have $p(\alpha)=\sum_{i=0}^{m} p_i \alpha^i = 0$ which implies that $\alpha^m = -\sum_{i=0}^{m-1} p_i \alpha^i$. Therefore for any $q^m-1>s>m$, α^s can be expressed as a polynomial of degree less than m, i.e.,

$$\alpha^s = \sum_{i=0}^{m-1} c_i \alpha^i \tag{1.2}$$

where $c_i \in GF(q)$. This simply means that the elements of the extension field $GF(q^m) = \{0, \alpha^0, \alpha^1, \ldots, \alpha^{q^m-2}\}$ has polynomial representations of degree less than m. The coefficients of these polynomials define the vector representations of the field elements. Since $p(x)|x^{q^m-1} - 1$ and α is a root of $p(x)$, then $\alpha|x^{q^m-1} - 1$. This implies $\alpha^{q^m-1} = 1$. The following theorem establishes the relationship between the elements of the $GF(q)$ and the polynomial x^{q-1}.

Theorem 1: *All set of zeros of the polynomial x^{q-1} is given by the non-zero elements of $GF(q)$.*

PROOF: Let us denote the field elements as $GF(q) = \{0, \beta_1, \beta_2, \ldots, \beta_{q-1}\}$. For any non-zero element $\beta \in GF(q) - \{0\}$, we have a representation $\beta = \alpha^s$ for some integer s, where α is the primitive element of $GF(q)$. Then, we have

$$\beta^{q-1} = (\alpha^s)^{q-1} = (\alpha^{q-1})^s = 1^s = 1. \tag{1.3}$$

Hence, β is a zero of $x^{q-1} - 1$. Since we have $q-1$ non-zero elements of the field, and by the fundamental theorem of algebra, the polynomial must have $x^{q-1} - 1$ has the roots $\{\beta_1, \beta_2, \ldots, \beta_{q-1}\}$, i.e., $x^{q-1} - 1 = \prod_{i=1}^{q-1}(x - \beta_i)$. □

Let us construct the extension field $GF(2^3)$ using a primitive polynomial of the form $x^3 + x + 1$ over $GF(2)$. Let α be a root and therefore $\alpha^3 = \alpha + 1$. Using this relationship, the elements of $GF(2^3)$ can be represented by polynomial as shown below. Similarly, addition and multiplication tables can be computed. The construction of these tables are left as exercise.

Table 1.1: Elements of $GF(2^3)$ in polynomial and binary vector forms.

Power	Polynomial	Binary
α^0	1	(0,0,1)
α^1	α	(0,1,0)
α^2	α^2	(1,0,0)
α^3	$\alpha + 1$	(0,1,1)
α^4	$\alpha^2 + \alpha$	(1,1,0)
α^5	$\alpha^2 + \alpha + 1$	(1,1,1)
α^6	$\alpha^2 + 1$	(1,0,1)

From basic algebra on polynomials, remember that $x^n - 1$ can be factored into p irreducible polynomial components, i.e.,

$$x^n - 1 = f_1(x) f_2(x) \ldots f_p(x) \tag{1.4}$$

Also remember from Theorem 1, we have $x^{q^m-1} - 1 = \prod_{j=1}^{q^m-1}(x - \beta_j)$. If we set $n = q^m - 1$, we have

$$x^{q^m-1} - 1 = f_1(x) f_2(x) \ldots f_p(x) = \prod_{j=1}^{q^m-1}(x - \beta_j) \tag{1.5}$$

which implies that each polynomial $f_i(x)$ can be represented in $GF(q^m)$ as a product of a subset of linear terms. Note here that each β_i is a root of exactly one of the $f_i(x)$. This irreducible polynomial is called the minimal polynomial of β_i.

For instance let us consider our previous example and the extension field $GF(2^3)$. We have the following factorization

$$x^7 - 1 = (x-1)(x^3+x+1)(x^3+x^2+1) \tag{1.6}$$
$$= (x-\alpha^0)[(x-\alpha^1)(x-\alpha^2)(x-\alpha^4)][(x-\alpha^3)(x-\alpha^6)(x-\alpha^5)] \tag{1.7}$$

Minimal polynomial	Elements of GF(8) (polynomial form)	Elements of GF(8) (power form)
$x-1$	1	α^0
x^3+x+1	α, α^2 and $\alpha^2+\alpha$	α, α^2 and α^4
x^3+x^2+1	$\alpha+1$, α^2+1 and $\alpha^2+\alpha+1$	α^3, α^6 and α^5

Definition 4: *The elements of* $GF(8)$ *{*α, α^2, α^4*} are defined to be conjugates of eachother with respect to the base field* $GF(2)$. *They share the same minimal polynomial* x^3+x+1.

From this simple example, one can deduce that if $f(x)$ is the minimal polynomial of β, then it is also the minimal polynomial of the conjugate set $\{\beta, \beta^q, \beta^{q^2}, \ldots, \beta^{q^{r-1}}\}$, where r is the smallest integer such that $\beta^{q^r} = \beta$.Since the elements of the conjugate set are the roots of the minimal polynomial $f(x)$, we have the following expression

$$f(x) = (x-\beta)(x-\beta^q)(x-\beta^{q^2})\ldots(x-\beta^{q^{r-1}}) \tag{1.8}$$

Here the exponent of the conjugate set are called the cyclotomic cosets. In other words, the cyclotomic coset of the element β is given by $\{1, q, q^2, \ldots, q^{r-1}\}$. For our example, we have three cyclotomic cosets, namely $\{0\}, \{1,2,4\}$ and $\{3,5,6\}$. Apparent from our definition that elements in the same cyclotomic coset are the roots of the same minimal polynomial.

Exercise 1: Consider the extension field $GF(9)$ with the base field $GF(3)$. Let α be the primitive element of $GF(9)$. Find the conjugate set for α. What is the corresponding minimal polynomial?

Hint: Note that x^2+x+2 is an irreducible polynomial over $GF(3)$.

2 Linear Algebra Basics and Hamming Codes

A general codebook construction can be arbitrary. However, codes constructed that way may require hard work for defining the encoding and decoding structures to be useful in real

world applications. A subclass of codes is linear codes which has some basic properties that prove useful to define, understand and implement them efficiently. Linear codes are defined over alphabets which are finite fields i.e. $\Omega_q = \mathbb{F}_q$, finite field with q elements.

Let $\mathbf{x} = (x_0, x_1, \ldots, x_{k-1}) \in \mathbb{F}_q^k$ and $\mathbf{y} = (y_0, y_1, \ldots, y_{k-1}) \in \mathbb{F}_q^k$ be vectors of length k, where $x_i, y_i \in \mathbb{F}_q$ for $1 \leq i \leq k-1$. We define $\mathbf{x}+\mathbf{y}$ and $\mathbf{xy}$ as the componentwise addition and multiplications over $\mathbb{F}_q$, respectively. A scalar a is defined to be equivalent to a vector where each component is a. The set V is said to be a k-dimensional linear space if for all $\mathbf{x}, \mathbf{y} \in \mathbf{V}$, the finite field axioms are satisfied. A set $\mathbf{S} \subseteq \mathbf{V}$ is a linear subspace if it is closed under addition and scalar multiplication operations. For example, the set of all 3−bit tuples form a linear space $\mathbf{V}$ where the subset $\{000, 001, 010, 011\}$ is a linear subspace of $\mathbf{V}$.

A k- dimensional linear space can be specified by a set of basis vectors $A_V = \{\mathbf{v}_0, \ldots, \mathbf{v}_{k-1}\}$ such that any element of $\mathbf{V}$ can be defined in terms of these basis vectors. Basis vectors are not unique but for a set of vectors to define a basis set, the elements of the set must be mutually independent and the cardinality of the set must be equal to k, i.e., $|A_V| = k$. Since $\{\mathbf{v}_0, \ldots, \mathbf{v}_{k-1}\}$ are mutually independent,

$$a_0\mathbf{v}_0 + a_1\mathbf{v}_1 + \cdots + a_{k-1}\mathbf{v}_{k-1} = 0 \tag{2.1}$$

implies that $\{a_0 = 0, a_1 = 0, \ldots, a_{k-1} = 0\}$. For example, $\{001, 010, 100\}$ is a set of basis vectors for the space defined by all 3− bit tuples.

The null space of a k-dimensional linear space $\mathbf{V}$, denoted as $\mathbf{V}^c$, is defined by all the vectors $\mathbf{v}^c$ of length k such that

$$\forall \mathbf{v} \in \mathbf{V}, \mathbf{v}\mathbf{v}^c = 0 \quad \text{where} \quad \mathbf{v}^c = \{v_0^c, \ldots, v_{k-1}^c\}, v_i^c \in \mathbb{F}_q \tag{2.2}$$

Exercise 2: Show that $\mathbf{V}^c$ is a linear space.

Linear spaces, defined by the basis vectors, are conventionally represented by matrices in which the columns of the matrix are the basis vectors of the linear space. This definition is analogous to the linear codes and their generator matrices. A linear block code C of length n is defined to be a subspace of $\mathbb{F}_q^n$. Therefore, a linear code of length n and dimension k is denoted as (n, k) code and the encoder bijectively maps the elements of $\mathbb{F}_q^k$ to the elements of the subspace $C \subseteq \mathbb{F}_q^n$. Assuming that C is specified by the basis vectors $\{\mathbf{g}_0, \mathbf{g}_1, \ldots, \mathbf{g}_{k-1}\}$ each of length n, generator matrix $\mathbf{G}$ of a (n, k) linear block code C is defined by a $k \times n$ matrix over $\mathbb{F}_q$ whose rows are the basis vectors, in other words,

$$\mathbf{G} = \begin{pmatrix} \mathbf{g}_0 & \mathbf{g}_1 & 0 & \ldots & 0 & \mathbf{g}_{k-1} \end{pmatrix}_{n \times k}^T$$

where $(.)^T$ is the transpose operation.

Therefore, the linear code C is defined by a matrix multiplication $\mathbf{mG}$ where $\mathbf{m} = \{m_0, m_1, \ldots, m_{k-1}\}$, $m_i \in \mathbb{F}_q$. We could have alternatively defined $\mathbf{G}$ to be a $n \times k$ matrix whose columns are $\mathbf{g}_i, 0 \leq i \leq k-1$. Thus, the code C by is either the column or row space of $\mathbf{G}$. We will use

both definitions of the generator matrix interchangeably throughout. The null space of the code C, denoted as C^c is also a linear space with the set of basis functions $\{\mathbf{h}_0, \mathbf{h}_1, \ldots, \mathbf{h}_{n-k}\}$ where $\mathbf{h}_i$ is a vector of length n over $\mathbb{F}_q$. Similarly, we can define a $n-k \times n$ parity check matrix $\mathbf{H}$ using the basis vectors as the rows,

$$\mathbf{H} = \begin{pmatrix} \mathbf{h}_0 & \mathbf{h}_1 & 0 & \ldots & 0 & \mathbf{h}_{n-k-1} \end{pmatrix}^T_{n \times n-k}$$

Exercise 3: Show that $\mathbf{HG} = \mathbf{0}_{n-k \times k}$.

Theorem 2: *Let $\boldsymbol{H}$ be a parity check matrix of a linear code C. The minimum distance of the code C is d if H has d linearly dependent columns and any $1 \leq \nu \leq d-1$ column selections results in mutually independent set of vectors.*

PROOF: Left as an exercise.

Theorem 3: *If $\boldsymbol{G} = [\boldsymbol{I_k}|\boldsymbol{P}]$ is a generator matrix for a linear code C. Then, $\boldsymbol{H} = [-\boldsymbol{P^T}|\boldsymbol{I_{n-k}}]$.*

PROOF: Let $\mathbf{c} \in C$, then we have $\boldsymbol{Hc}^T = \boldsymbol{HmG}^T = \boldsymbol{HG}^T\mathbf{m}^T = (-\mathbf{P}^T + \mathbf{P})\mathbf{m}^T = 0$. □

2.1 Some basic linear codes

Let $\mathbb{F}_2$ be our base field, and (n,k) linear code will have n-bit long block length that is mapped from a k-bit long message block. One of the basic and early codes of practical importance was the parity check code which can detect one error in a block of n bits. Parity check code has the following parity check matrix,

$$\mathbf{H} = \begin{pmatrix} 1 & 1 & 1 & \ldots & 1 & 1 \end{pmatrix}_{1 \times n}$$

Using Theorem 3, we can obtain the generator matrix $\mathbf{G} = [\mathbf{I}_k|\mathbf{1}^T]$ were $\mathbf{1}$ is the all-one row vector. Even-parity check code is $(n, n-1)$ linear code with minimum distance 2. $\mathbf{G}$ generates a systematic codeword in which the last bit is given by $\sum_{i=0}^{k} m_i$ where $\mathbf{m} = \{m_0, \ldots, m_{k-1}\}$. For decoding such a code, first the syndromes are computed by multiplying the received word with the parity check matrix of the code. In otherwords, the decoder simply computes the sum of all the symbols of the received word $\mathbf{r}$, i.e., $s = \sum_{i=0}^{n-1} r_i$. If the sum (syndrome s) is zero, no error is detected, otherwise an odd number of errors are detected in $\mathbf{r}$.

Exercise 4: Think about the parity check matrix for odd-parity code.

Exercise 5: Show that $(n, n-1, 2)$ even-parity code has all the vectors in $\mathbb{F}_2^n$ of even hamming weight.

One of the earliest, well-known linear codes is Hamming code. Hamming codes are defined by their parity check matrices and are able to correct single bit error defined over $GF(2)$.

A conventional Hamming code assume a $r \times 2^r - 1$ parity check matrix where columns are binary representation of numbers $1, \ldots, 2^r - 1$. Therefore, Hamming code is a $(2^r - 1, 2^r - 1 - r, 3)$ code for $r \geq 2$ defined over $GF(2)$. Consider the following Hamming code example for $r = 3$:

$$\mathbf{H} = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}_{3\times 7}$$

If the received vector $\mathbf{r} = \mathbf{c} + \mathbf{e}$ is corrupted by a single error i.e., $\mathbf{e}$ has weight one, then the syndrome computation will result in any one of the columns of $\mathbf{H}$. Thus, the decoding operation is simple that once the syndrome computed, the value of the syndrome is the index at which the received word will have to be flipped for error correction.

Hamming codes can be extended to include one more parity check bit at the end of each valid codeword such that the parity check matrix will have the following form,

$$\mathbf{H} = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}_{4\times 8}$$

Exercise 6: Show that the extended Hamming code has minimum Hamming distance of 4 i.e., it is a $(2^r, 2^r - r - 1, 4)$ code.

Exercise 7: Let us consider the parity check matrix of the $(2^r - 1, 2^r - 1 - r, 3)$ Hamming code for any $r \geq 2$. We construct a new code by throwing away all columns of even weight of the parity check matrix. What are parameters of the new code? What is the minimum distance? Is there a relationship between this code with the extended Hamming code? Devise a decoding algorithm for this code (for example if a syndrome of even weight is encountered what does the decoder do?).

Exercise 8: Prove that Hamming codes are single-error correcting perfect codes. In other words, they achieve the Hamming bound (see previous class notes). Think about the error detection capability of the code.

Exercise 9: Remember that from Hamming bound, for any linear code defined over $GF(q)$ we have $q^n \geq q^k \sum_{i=0}^{t} \binom{n}{i} (q-1)^i$. Show that the following two codes achieve this bound and hence they are perfect codes. What are the minimum distance of these codes?

- (23,12) code defined over $GF(2)$.
- (11,6) code defined over $GF(3)$.

3 CYCLIC CODES

Cyclic codes are important subclass of general linear codes. the linear code C is qualified to be cyclic if it is invariant under cyclic shifts i.e., if $\mathbf{c} = (c_0, \ldots, c_{n-1}) \in C$ and $\mathbf{c}^s = (c_{n-1}, c_0, \ldots, c_{n-2}) \in C$. Cyclic codes need not be linear at all. In fact there are nonlinear code with cyclic property. However, historically this is not a preferred way of constructing good codes.

For any word of length n, $\mathbf{c} \in \Omega_q^n$ (and the shifted version), we can associate a polynomial of degree less than n,

$$c(x) = c_0 + c_1 x + \cdots + c_{n-1} x^{n-1} \tag{3.1}$$

$$c^s(x) = c_{n-1} + c_0 x + \cdots + c_{n-2} x^{n-1} \tag{3.2}$$

where we notice that $c(x)$ and $c^s(x)$ are equivalent in the ring of polynomials modulo $x^n - 1$, i.e.,

$$c^s(x) = xc(x) \mod (x^n - 1) \tag{3.3}$$

Through linearity argument and the fact that additional shifts do not take us out of the linear space spanned by the code C, for any polynomial $a(x)$, we have $a(x)c(x) \mod (x^n-1) \in C$. From Eqn. 1.4, we can choose a factor of $(x^n - 1)$, having degree r, that can generate a cyclic code C. In other words, let $q(x)$ be a polynomial over $GF(q)$ of degree less than $n - r$. Then, generator polynomial so chosen enables to claim $q(x)g(x) \in C$. Note that excluding the trivial minimal polynomials 1 and $x^n - 1$, there are $2^p - 2$ possible generator polynomials (different cyclic codes). However, only some of the generator polynomials give good codes with desirable distance and rate properties.

Definition 5: *If $g(x)$ is a generator polynomial of C, then the check polynomial of C, $h(x)$ satisfies $g(x)h(x) = x^n - 1$, i.e., $g(x)h(x) = 0 \mod (x^n - 1)$.*

Let us consider an extension field $GF(2^3)$ with the following factorization

$$x^7 + 1 = (x + 1)(x^3 + x + 1)(x^3 + x^2 + 1) \tag{3.4}$$

The generator polynomial 1 does not generate any code but all the elements in $GF(8)$ and $g(x) = x + 1$ is nothing but parity check code. Minimal polynomials such as $x^3 + x + 1$ or $x^3 + x^2 + 1$ generates $(7,3)$ Hamming codes.

Exercise 10: Prove that the $(8,4)$ extended binary Hamming code is not cyclic.

Once we have the generator polynomial $g(x) = \sum_{j=0}^{r} g_j x^j$ for a cyclic code, the construction of a generator matrix falls from the idea of convolution because multiplication of two

polynomial in fact corresponds to convolution operation. Therefore the non-systematic generator matrix is given by

$$\mathbf{G}^T = \begin{pmatrix} g_0 & g_1 & \dots & \dots & \dots & g_r & 0 & 0 & \dots & 0 \\ 0 & g_0 & g_1 & \dots & \dots & \dots & g_r & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & g_0 & g_1 & \dots & \dots & \dots & g_r \end{pmatrix}_{k \times n}$$

As we shall later see, cyclic codes have many different and efficient encoding/decoding procedures that can lead to easy hardware implementations. Almost all commonly used linear codes are cyclic codes.

4 Bose-Chaudhuri Hocquenghem (BCH) Codes

BCH codes are one of the good known linear cyclic block codes. BCH codes possess easy encoding and decoding algorithms and show multiple-error correcting capabilities. After establishing the necessary background on algebra, we are now ready to define BCH codes.

A BCH code can be defined by either a parity check matrix or a generator polynomial.

4.1 Design of good generator polynomials

Let us start with finding a good generator polynomial that will enable us to have good error detection/correction capabilities. Remember from previous section that, we have

$$x^{q^m-1} - 1 = f_1(x) f_2(x) \dots f_p(x) \tag{4.1}$$

and since $g(x) | x^{q^m-1} - 1$, then $g(x)$ must be a product of $f_i(x)$ i.e., set of prime polynomial over a given $GF(q)$. The main objective is to design the roots of the generator polynomial (let us say ν roots) and using the corresponding minimal polynomials of each root, we find the generator polynomial as follows,

$$g(x) = LCM\{f_1(x), \dots, f_\nu(x)\} \tag{4.2}$$

where $\{f_1(x), \dots, f_\nu(x)\}$ are the corresponding minimal polynomials and $LCM(.)$ is the least common multiple operation. Number of roots ν and the selection of these roots are related to the error-correction capability of the code and will be discussed next.

Consider a (n,k) BCH code. Let $a(x)$ be the message polynomial of order k and after encoding operation the codeword polynomial will be given by $c(x) = a(x)g(x)$. Suppose that the codeword polynomial is corrupted by an error polynomial $e(x)$ i.e., the received polynomial

is $r(x) = c(x) + e(x)$ where $e(x)$ has t non-zero coefficients at $i_1, i_2, \ldots, i_t$ locations. If we let $\{\omega_1, \ldots, \omega_v\} \in GF(q^m)$ be the roots of $g(x)$, then we must have

$$r(\omega_j) = c(\omega_j) + e(\omega_j) = a(\omega_j)g(\omega_j) + e(\omega_j) = e(\omega_j) = \sum_{i=0}^{n-1} e_i \omega_j^i = \sum_{s=1}^{t} e_{i_s} \omega_j^{i_s} \quad \text{for } j = 1, \ldots, v \tag{4.3}$$

Note that there are $2t$ unknowns, namely $\{i_1, \ldots, i_t, e_{i_1}, \ldots, e_{i_t}\}$ and v non-linear equations. For a unique solutio, we need at least $2t$ equations. Therefore, in order to design a t-error correcting (n,k) BCH code, we choose $v = 2t$ and $\omega_j = \alpha^{a+bj}$ in the extension field, where b is chosen such that $\gcd(n,b) = 1$. If $a = 1$, the BCH code is named narrow-sense. We will see how to solve the set of equation given in Eqn. 4.3 when we talk about algebraic decoding for BCH and Reed-Solomon codes. We note here that the choice of a might have an effect on the encoder/decoder complexity.

Exercise 11: Let us consider the following factorization over $GF(2)$

$$x^{15} + 1 = (x^4 + x + 1)(x^4 + x^3 + 1)(x^4 + x^3 + x^2 + x + 1)(x^2 + x + 1)(x + 1) \tag{4.4}$$

Choose an appropriate primitive polynomial to construct $GF(16)$. Give an example of a minimal polynomial that cannot be a primitive polynomial. Construct a single error ($t = 1$) correcting binary BCH code with a block length of 15. What is the generator polynomial? What are the code parameters k and D_{min}? Repeat this with $t = 2$.

When constructing an arbitrary cyclic code, there is no guarantee as to the resulting minimum distance. An exhaustive computer search is often used to find the minimum-weight codewords of a linear code and thereby the minimum distance. However the BCH codes, given a constraint on their generator polynomial, can achieve a certain target/design minimum distance.

Theorem 4: *Let C be a (n,k) cylic code over $GF(q^m)$. We choose m so that $GF(q^m)$ is the smallest extension field that contains the primitive element α and $\alpha^{n-1} \equiv 1$. Let $g(x)$ be some product of minimal polynomials corresponding to the set of powers of the primitive element $\{\alpha^a, \alpha^{a+1}, \ldots, \alpha^{a+2t-1}\}$ with $b = 1$. Therefore, we have $g(\alpha^a) = g(\alpha^{a+1}) = \cdots = g(\alpha^{a+2t-1}) = 0$ for some integer $b \geq 1$. The code C defined by the generator polynomial has a minimum distance $D_{min} \geq 2t + 1$. .*

Note that from theorem 2, we deduce that $c(\alpha^a) = c(\alpha^{a+1}) = \cdots = c(\alpha^{a+2t-1}) = 0$. Therefore, it is easy to see that a parity check matrix can be constructed for a BCH code based on this observation as follows,

$$H = \begin{pmatrix} 1 & \alpha^a & \alpha^{2a} & \ldots & \alpha^{(n-1)a} \\ 1 & \alpha^{a+1} & \alpha^{2(a+1)} & \ldots & \alpha^{(n-1)(a+1)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \alpha^{a+2t-2} & \alpha^{2(a+2t-2)} & \ldots & \alpha^{(n-1)(a+2t-2)} \\ 1 & \alpha^{a+2t-1} & \alpha^{2(a+2t-1)} & \ldots & \alpha^{(n-1)(a+2t-1)} \end{pmatrix}$$

Exercise 12: Show that a binary primitive narrow-sense BCH code over $GF(2^m)$ has the following parity check matrix,

$$H = \begin{pmatrix} 1 & \alpha & \alpha^2 & \dots & \alpha^{(2^m-2)} \\ 1 & \alpha^3 & \alpha^6 & \dots & \alpha^{3(2^m-2)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \alpha^{2t-1} & \alpha^{2(2t-1)} & \dots & \alpha^{(2^m-2)(2t-1)} \end{pmatrix}$$

A design procedure for $(n, k, D_{min} \geq 2t+1)$ BCH code can be summarized as follows,

- Select a primitive element $\alpha \in GF(q^m)$ so that m is minimal.
- Select $2t$ consecutive powers of α starting with α^a for some integer $a \geq 1$.
- Let $f_i(x)$ be the minimal polynomial of α^i where $a \leq i \leq a+2t-1$. Then choose $g(x) = LCM(f_a(x), \dots, f_{a+2t-1}(x))$ in $GF(q)$.

Most of the time, primitive binary BCH codes are used because cardinality of cyclotomic cosets (mod n) is generally smaller for primitive codes. This also means more efficient (better rate) codes can be constructed given the target minimum distance. In addition, large alphabets have better potential to have smaller cyclotomic cosets. This can be regarded as one of the motivations for Reed-Solomon codes as described next.

Let us construct a primitive narrow-sense binary 2-bit error correcting BCH code over $GF(2^4)$. Since it is primitive, $n = 15$. At this point, we do not know the message length nor the rate of the code. Let x^4+x+1 be the primitive polynomial that generates the $GF(2^4)$ with the root α, the primitive element of the extension field. Since the code is narrow-sense 2-bit error correcting, the generator polynomial has the roots $\alpha, \alpha^2, \alpha^3$ and α^4. We can realize the the minimal polynomial associated with the roots α, α^2 and α^4 is x^4+x+1, and the the minimal polynomial associated with the root α^3 is $x^4+x^3+x^2+x+1$. Therefore the $LCM(.)$ of the minimal polynomials of roots of the generator polynomials is given by

$$\begin{aligned} g(x) &= (x^4+x+1)(x^4+x^3+x^2+x+1) && (4.5) \\ &= x^8+x^7+x^6+x^4+1 && (4.6) \end{aligned}$$

In fact, the order of $g(x)$ is 8 suggesting that there are eight roots of this generator polynomials. These roots are given by $\alpha, \alpha^2, \alpha^3, \alpha^4, \alpha^6, \alpha^8, \alpha^9, \alpha^{12}$. Thus, the number of parity bits is 8. This generates a BCH code of $(15, 7, D_{min} \geq 5)$. In fact for these parameter selections, the code has exactly minimum distance 5. This result can be verified based on the following theorem.

Theorem 5: *A (n, k) primitive binary BCH code over $GF(2^m)$ and design distance $2t+1$ has exact minimum distance $2t+1$ if*

$$\sum_{i=0}^{t+1} \binom{n}{i} > 2^{mt} \qquad (4.7)$$

PROOF: The proof follows by recognizing that primitive binary BCH code has minimum distance d odd and using the sphere packing (Hamming) bound. We will generalize this theorem in problem section for general non-binary codes. □

Hence, the parity check matrix of $(15,7,5)$ binary BCH code will be given as,

$$H = \begin{pmatrix} 1 & \alpha & \alpha^2 & \dots & \alpha^{14} \\ 1 & \alpha^3 & \alpha^6 & \dots & \alpha^{42} \end{pmatrix} = \left(\begin{array}{cccccccccccccc} 1&0&0&0&1&0&0&1&1&0&1&0&1&1&1 \\ 0&1&0&0&1&1&0&1&0&1&1&1&1&0&0 \\ 0&0&1&0&0&1&1&0&1&0&1&1&1&1&0 \\ 0&0&0&1&0&0&1&1&0&1&0&1&1&1&1 \\ \hline 1&0&0&0&1&1&0&0&0&1&1&0&0&0&1 \\ 0&0&0&1&1&0&0&0&1&1&0&0&0&1&1 \\ 0&0&1&0&1&0&0&1&0&1&0&0&1&0&1 \\ 0&1&1&1&1&0&1&1&1&1&0&1&1&1&1 \end{array}\right)$$

The nice thing about the BCH codes is that they are cyclic codes and thus the encoding operation as well as the syndrome computations are quite easy and can be implemented using shift registers. Using our example, we can construct the BCH encoder using the following shift register,

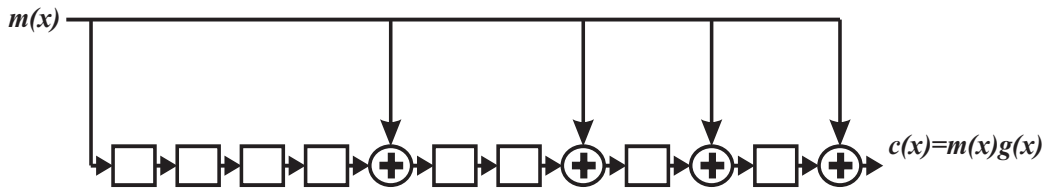

Figure 4.1: Non-systematic shift register encoder for the above example. Squares are simple memory elements such as flip flops.

4.2 Reed-Solomon Codes

A (n,k) RS code is a non-binary BCH code defined over $GF(q)$ where $n = q-1$ where q is either prime or power of prime. Remember that for a given primitive element α of the extension field $GF(q^m)$ the conjugate set consists of the set $\{\alpha^a, \alpha^{aq}, \dots \alpha^{aq^{r-1}}\}$ where r is the smallest integer such that $\alpha^{aq^r} = \alpha^a$, i.e., we have the cyclotomic set $\{a, aq, \dots, aq^r\}$. For RS codes we have $n = q-1$, so the cyclotomic set $\{a\}$ contains only one element because $aq^s = a \mod (q-1)$ for $0 \le s \le r-1$.Therefore, the minimal polynomial corresponding to element α^a is given by $(x-\alpha^a)$. Thus, the generator polynomial for t- symbol error correcting RS code is given by

$$g(x) = (x-\alpha^a)(x-\alpha^{a+1})\dots(x-\alpha^{a+2t-1}) \tag{4.8}$$

Let us assume the primitive polynomial x^3+x+1 generates the $GF(8)$ and let α be the primitive element. We construct the $t = 2$-symbol error correcting RS code using the following

narrow-sense generator polynomial,

$$g(x) = (x-\alpha)(x-\alpha^2)(x-\alpha^3)(x-\alpha^4) \quad (4.9)$$
$$= x^4 + \alpha^{13}x^3 + \alpha^6 x^2 + \alpha^3 x + \alpha^{10} \quad (4.10)$$

The order of the generator polynomial is 4 for this example. The primitive non binary BCH code (RS code) has length 7 symbols and each symbol may consist of 3 bits. This means that we constructed (7,3) RS code. The minimum distance of the constructed code ≥ 5 by design. But we know that from singleton bound we have an upper bound on the minimum distance, i.e., for any linear block code $d_{min} \leq n-k+1$ (refer to previous lectures). Therefore, minimum distance of this RS code is guaranteed to be 5. This is a special property of the RS codes which is generalized and stated in the following theorem,

Theorem 6: *An (n,k) Reed-Solomon code defined over $GF(q)$ has a minimum distance $n-k+1$.*

PROOF: Proof follows from our previous discussion for generalized (n,k) RS code using BCH bound and singleton bound.

An (n,k) code that satisfies the Singleton bound with equality is called Maximum Distance Separable (MDS) code. There are special properties with MDS codes. For example, the dual of an MDS code is also MDS. In addition, punctured/shortened MDS codes maintain the MDS property i.e., subcodes of an MDS code is also an MDS code. Finally, the weight distribution of the codewords of an MDS code is known. This tremendously help predict the performance of MDS codes and their functional characteristics.

The parity check matrix for the RS codes can be given based on the non-binary BCH code parity check matrix by recognizing that $\{c(\alpha), c(\alpha^2), \ldots, c(\alpha^{2t})\}$ are all zeros. Let $m(x)$ be a message polynomial of degree k and $g(x) = g_0 + g_1 x + \cdots + g_{2t-1}x^{2t-1} + x^{2t}$ be the generator polynomial of the code. The systematic encoding process for (n,k) RS code will then be given by (note $2t = n-k$ for RS codes)

$$c(x) = m(x)x^{n-k} - \left[m(x)x^{n-k} \mod g(x) \right] \quad (4.11)$$

The following shift register encoder generates the check bits $m(x)x^{n-k} \mod g(x)$ and appends it to the message bits $m(x)$ to generate the systematic codeword.

We have seen that by changing the channel alphabet over which the BCH is defined, we obtain codes with different properties such as RS codes. We now consider an example: different BCH codes over different decoder alphabets for comparison. Let us take our previous example (15,7,5) binary BCH code over the base field $GF(2)$. Let us assume α is the primitive polynomial of the extension field $GF(2^4)$. We present the BCH codes over the subfields $GF(2), GF(4)$ and $GF(16)$. A 2-symbol/bit correcting BCH code will have $\alpha, \alpha^2, \alpha^3$ and α^4 as the roots of the generator polynomial.

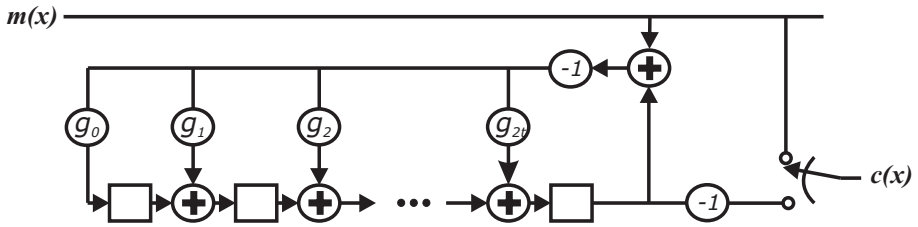

Figure 4.2: Systematic shift register encoder for RS codes. Squares are simple memory elements such as flip flops.

- BCH code over $GF(2)$ ($m = 4$) will have α and α^3 as the roots, the rest will be the conjugates. Thus, the order of the generator polynomial is $n-k = 8$, therefore we obtain $(15,7)$ BCH code with minimum distance 5. This code has a block length of 15 bits.

- BCH code over $GF(4)$ ($m = 2$) will have α, α^2 and α^3 as the roots, the rest will be the conjugates. Thus, the order of the generator polynomial is $n-k = 6$, therefore we obtain $(15,9)$ BCH code with minimum distance 5. This code has a block length of 30 bits.

- BCH code over $GF(16)$ ($m = 1$) will have $\alpha, \alpha^2, \alpha^3, \alpha^4$ as the roots, no conjugates exist. Since this BCH code is primitive, by definition, it is an RS code. The order of the generator polynomial is $n-k = 4$, therefore we obtain $(15,11)$ RS code with minimum distance 5. This code has a block length of 60 bits.

Finally, we remark about an important type of matrix called Vandermonde and establish the similarity between Vandermonde matrix and the parity check matrix of a general BCH code.

Definition 6: *A $\eta \times \eta$ Vandermonde matrix is a matrix of the form*

$$V_\eta = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ X_1 & X_2 & X_3 & \dots & X_\eta \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ X_1^{\eta-1} & X_2^{\eta-1} & X_3^{\eta-1} & \dots & X_\eta^{\eta-1} \end{bmatrix}$$

whose determinant is non-zero given X_i terms are distinct.

If we replace X_i with α^i, we can observe that this will be identical to H^T of the narrow-sense BCH code ($a = 1$) and thereby RS codes. Thus, we can argue that a (n,k) BCH code whose check matrix have $n-k \leq 2t$ rows, has minimum distance at least $2t+1$.

Observing the structure of the parity check matrix leads us to give an alternative definition of RS codes by constructing a non-systematic generator matrix. Let us have a message vector $\mathbf{m} = (m_0, m_1, \dots, m_{k-1})^T \in \Omega_q^k$, where the message polynomial is

$$m(x) = m_0 + m_1 x + m_2 + \cdots + m_{k-1} x^{k-1} \tag{4.12}$$

By far, we have shown how to construct systematic and non-systematic RS codes by multiplying the message polynomial with the generator polynomial of the code. Instead of multiplication, let us evaluate $m(x)$ at points $\{1, \alpha, \alpha^2, \ldots, \alpha^{n-1}\}$ to form the codeword vector $\mathbf{c} = (c_0, c_1, \ldots, c_{n-1})$ of a (n, k) RS code. We will show shortly, this operation indeed generates a RS code by looking at the check conditions $c(\alpha) = c(\alpha^2) = \cdots = c(\alpha^{n-k}) = 0$ which was based on our previous development of the general BCH codes and generator polynomials. However, let us first observe that evaluation of the polynomial $m(x)$ at the points $\{1, \alpha, \alpha^2, \ldots, \alpha^{n-1}\}$ is nothing but multiplying the message vector by the following generator matrix i.e., $\mathbf{c} = \mathbf{Gm}$ where

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

Our previous arguments were based on the parity check characterization of the code. Now let us prove that generator characterization through evaluations of the message polynomial at different locations is exactly the same as the parity check characterization. To show this, let us multiply the generator matrix so constructed with the parity check matrix of the RS code,

$$\underbrace{\begin{bmatrix} 1 & \alpha & \alpha^2 & \ldots & \alpha^{n-1} \\ 1 & \alpha^2 & (\alpha^2)^2 & \ldots & (\alpha^2)^{n-1} \\ 1 & \alpha^3 & (\alpha^3)^2 & \ldots & (\alpha^3)^{n-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \alpha^{n-k} & (\alpha^{n-k})^2 & \ldots & (\alpha^{n-k})^{n-1} \end{bmatrix}_{n-k \times n}}_{\mathbf{H}} \underbrace{\begin{bmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & \alpha & \alpha^2 & \ldots & \alpha^{k-1} \\ 1 & \alpha^2 & (\alpha^2)^2 & \ldots & (\alpha^2)^{k-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & \alpha^{n-1} & (\alpha^{n-1})^2 & \ldots & (\alpha^{n-1})^{k-1} \end{bmatrix}_{n \times k}}_{\mathbf{G}}$$

Let $\mathbf{H}_i$ be the i-th row of the parity check matrix where $1 \leq i \leq n-k$ and $\mathbf{G}_j$ be the j-th column of the generator matrix where $0 \leq j \leq k-1$. It is sufficient to show that $\mathbf{H}_i \mathbf{G}_j = 0 (\mod x^n - 1)$. Observe that

$$\mathbf{H}_i \mathbf{G}_j = \sum_{k=0}^{n-1} (\alpha^{i+j})^k = \frac{1 - (\alpha^{i+j})^n}{1 - \alpha} = \frac{1 - (\alpha^n)^{i+j}}{1 - \alpha} \tag{4.13}$$

But since $\alpha^n = 1 (\mod x^n - 1)$, the result follows. This observation leads to very simple codeword construction as it will enough to evaluate the message polynomial at n different locations to get a valid RS codeword.

Reed-Solomon codes found applications in many diverse fields of engineering since their first invention in 1960s [1]. Although these codes are old, they are still used in many commercial applications and products in the market. RS codes are optimal in the sense that they achieve the singleton bound. However, they are defined over large alphabets and basic unit of hardware is in bits. Therefore, some kind of transformation of RS symbols into bits

is needed in order to be able to generate binary codes. Yet, it is not optimal to construct a non-binary code if we are going to use the code in the binary domain simply because there are better codes constructed over the binary base fields such as binary BCH codes. The performance determinant of a given application is not only the way the code is constructed and used but also is the error characteristics over which the code is used to protect data. That is why RS codes are still popular in some applications which are mostly dominated by burst errors. Since the consecutive bit errors only effect few symbols in an RS code, the decoder will be able to recover the information whereas a binary code will fail due to too many bit errors.

Exercise 13: If a $(n/m, k/m)$ RS code is defined over $GF(2^m)$, each symbol will simply have m-bit representation. The resulting binary code will have parameters $(n, k, \geq n-k+1)$ because the minimum number of bit errors this RS code can correct (the worst case) is $n-k+1$ which follows from the singleton bound. Let d be the minimum distance of the code, other way of writing these parameters is $(n, n-(d-1)m, \geq d)$. For a t-bit error correcting BCH code, we argued that $n-k \leq mt$ and $d \geq 2t+1$. Thus, $t \leq (d-1)/2$ implies that

$$n - m\left(\frac{d-1}{2}\right) \leq n - mt \leq k \tag{4.14}$$

which leads to the binary BCH code with parameters $(n, \geq n - m\left(\frac{d-1}{2}\right), \geq d)$. As can be seen the binary BCH code guarantees the same distance (design) while ensuring a better code rate.

5 Algebraic Decoding of BCH Codes

Up to this point, we have analyzed code construction methodologies and talked about suitable encoders for BCH codes. In this section, we will present some of the well-known decoding algorithms devised in general for cyclic codes. These algorithms are therefore applicable to BCH as well as RS decoding. We will present a decoding procedure that can correct up to half the minimum distance of the code. However, there have been studies that is shown to improve performance by decoding beyond half the minimum distance. We will cover such algorithms later.

Suppose that the error polynomial $e(x)$ have $v \leq t$ errors in indexed locations $i_1, i_2, \ldots, i_v$ and let $r(x) = c(x) + e(x)$ be the received polynomial. The initial operation of the decoder is to multiply the received vector with the parity check matrix of the code $\mathbf{H}$. This is nothing but the evaluation of the received polynomial at the roots of the generator polynomial $\alpha, \alpha^2, \ldots, \alpha^{2t}$ (Note that we assume $a = 1$ in the rest of our discussion of this section. The general case $a \geq 1$ shall be left as an exercise). The result of this operation generates the syndromes as follows,

$$S_j = r(\alpha^j) = c(\alpha^j) + e(\alpha^j) = e(\alpha^j) = \sum_{s=1}^{v} e_{i_s}(\alpha^j)^{i_s} = \sum_{s=1}^{v} e_{i_s}(\alpha^{i_s})^j \quad 1 \leq j \leq 2t \tag{5.1}$$

For short hand notation, set $Y_s = e_{i_s}$ and $X_s = \alpha^{i_s}$. We have the following set of equations to solve,

$$\begin{aligned} S_1 &= Y_1X_1 + Y_2X_2 + \cdots + Y_\nu X_\nu && (5.2)\\ S_2 &= Y_1X_1^2 + Y_2X_2^2 + \cdots + Y_\nu X_\nu^2 && (5.3)\\ S_3 &= Y_1X_1^3 + Y_2X_2^3 + \cdots + Y_\nu X_\nu^3 && (5.4)\\ \vdots &= \vdots && (5.5)\\ S_{2t} &= Y_1X_1^{2t} + Y_2X_2^{2t} + \cdots + Y_\nu X_\nu^{2t} && (5.6) \end{aligned}$$

Given the computed syndromes, the objective of the decoder is to find $Y_1,\ldots,Y_\nu,X_1,\ldots,X_\nu$. Note that there are $2t$ equation and 2ν unknowns. In order for a unique solution $\nu \le t$ as assumed. The second step of the decoder is to compute, if possible, the error locations $X_1,\ldots,X_\nu$ by constructing a polynomial called "Error-Locator Polynomial" $\Lambda(x)$. This polynomial have the reciprocal of X_s as its roots, i.e., $\Lambda(X_1^{-1}) = \Lambda(X_2^{-1}) = \cdots = \Lambda(X_\nu^{-1}) = 0$ where

$$\begin{aligned} \Lambda(x) &= (1-xX_1)(1-xX_2)\ldots(1-xX_\nu) && (5.7)\\ &= \Lambda_\nu x^\nu + \Lambda_{\nu-1}x^{\nu-1} + \cdots + \Lambda_1 x + 1 && (5.8) \end{aligned}$$

The decoding procedure that is used to find $\Lambda(x)$ from $S_1,\ldots,S_{2t}$ is called Paterson-Gorenstein-Zierler (PGZ) decoder. Let us consider the following product,

$$\begin{aligned} Y_sX_s^{j+\nu}\Lambda(X_s^{-1}) &= Y_sX_s^{j+\nu}\left(\Lambda_\nu X_s^{-\nu} + \Lambda_{\nu-1}X_s^{-\nu+1} + \cdots + \Lambda_2X_s^{-2} + \Lambda_1X_s^{-1} + 1\right) && (5.9)\\ &= Y_s\left(\Lambda_\nu X_s^{j} + \Lambda_{\nu-1}X_s^{j+1} + \cdots + \Lambda_2X_s^{j+\nu-2} + \Lambda_1X_s^{j+\nu-1} + X_s^{j+\nu}\right) && (5.10)\\ &= 0. && (5.11) \end{aligned}$$

Now for a given j, let us sum the equations from $s=1$ to $s=\nu$. This gives us

$$0 = \sum_{s=1}^{\nu} Y_sX_s^{j+\nu}\Lambda(X_s^{-1}) = \Lambda_\nu\sum_{s=1}^{\nu} Y_sX_s^{j} + \Lambda_{\nu-1}\sum_{s=1}^{\nu} Y_sX_s^{j+1} + \cdots + \Lambda_1\sum_{s=1}^{\nu} Y_sX_s^{j+\nu-1} + \sum_{s=1}^{\nu} Y_sX_s^{j+\nu} \quad (5.12)$$

$$= \Lambda_\nu S_j + \Lambda_{\nu-1}S_{j+1} + \cdots + \Lambda_1 S_{j+\nu-1} + S_{j+\nu} \quad (5.13)$$

In order to end up with all the known syndromes, we must have $1 \le j \le 2t-\nu$. Thus, we have the following set of linear equations to solve,

$$\underbrace{\begin{bmatrix} S_1 & S_2 & S_3 & \ldots & S_\nu \\ S_2 & S_3 & S_4 & \ldots & S_{\nu+1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ S_\nu & S_{\nu+1} & S_{\nu+2} & \ldots & S_{2\nu-1} \end{bmatrix}}_{\mathbf{S}_{2\nu}} \begin{bmatrix} \Lambda_\nu \\ \Lambda_{\nu-1} \\ \vdots \\ \Lambda_1 \end{bmatrix} = \begin{bmatrix} -S_{\nu+1} \\ -S_{\nu+2} \\ \vdots \\ -S_{2\nu} \end{bmatrix}$$

We note that the syndrome matrix can be decomposed to Vandermonde components as follows,

$$\mathbf{S}_{2\nu} = V_\nu \mathbf{D} V_\nu^T \tag{5.14}$$

where

$$\mathbf{D} = \begin{bmatrix} Y_1X_1 & 0 & 0 & \dots & 0 \\ 0 & Y_2X_2 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & Y_\nu X_\nu \end{bmatrix}$$

If $\nu < t$ errors occur, then $\mathbf{S}_{2t}$ is non-singular matrix because $X_{\nu+1}, \dots, X_t, Y_{\nu+1}, \dots, Y_t$ are zeros and therefore $\det(\mathbf{D}) = 0$. The latter follows from the basic fact of linear algebra $\det(\mathbf{S}_{2\nu}) = \det(\mathbf{S}_{2\nu}) = \det(V_\nu)^2 \det(\mathbf{S}_{2\nu})$ and Vandermonde matrices are singular for distinct entries.

This observation provides a way to determine the number of errors occurred in the codeword as long as $\nu \leq t$. In other words, we find largest ν such that $\det(\mathbf{S}_{2\nu})$ is nonzero. Next, we compute $\mathbf{S}_{2\nu}^{-1}$ based on the syndrome values. Thus, the coefficients of the error locator polynomial $\Lambda(x)$ are found using matrix multiplication. The roots i.e., the set $\{X_1, \dots, X_\nu\}$ of $\Lambda(x)$ can be found by trial and error, a process known as *Chien Search*. Once we determine the set $\{X_1, \dots, X_\nu\}$, we can use the first ν set of syndrome equations to compute $\{Y_1, \dots, Y_\nu\}$ because if ν errors occurred, $\{X_1, \dots, X_\nu\}$ will be distinct and therefore the following matrix will have non-zero determinant (Why?).

$$\begin{bmatrix} X_1 & X_2 & X_3 & \dots & X_\nu \\ X_1^2 & X_2^2 & X_3^2 & \dots & X_\nu^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ X_1^\nu & X_2^\nu & X_3^{\eta-1} & \dots & X_\nu^\nu \end{bmatrix}$$

Note that if we decode a binary code, we do not need to compute the error values $\{Y_1, \dots, Y_\nu\}$. This leads to complexity reductions in the decoder implementation.

5.1 Efficient Decoding Algorithms

Successful application of an error correcting code can only be realized if they possess efficient encoding and decoding procedures. Although the rate and minimum distance of the code are parameters based on which a performance estimation can be made, the efficiency and implementation complexity are the other crucial items that would have an effect on the final judgement of the designer.

Previously, we have shown a way to compute the coefficients of the error locator polynomial $\Lambda(x)$. However, matrix inversion is costly procedure. There are efficient ways to compute $\Lambda_v, \dots \Lambda_1$. One way is given by Berlekamp-Massey (BM) Algorithm based on a iterative method. It sets $\Lambda(x) = 1$ initially, and gradually improve the result based on a number of iterations and syndromes S_j such that

$$S_{j+v} + \Lambda_1 S_{j+v-1} + \cdots + \Lambda_{v-1} S_{j+1} + \Lambda_v S_j = 0 \quad \Rightarrow \quad S_{j+v} = -\sum_{i=1}^{v} \Lambda_i S_{j-i+v} \tag{5.15}$$

$$S_l = -\sum_{i=1}^{v} \Lambda_i S_{l-i} \tag{5.16}$$

in which we make the change of variables $l = j + v$.

The BM algorithm generates the LFSR that produces the entire syndrome sequence $\{S_1, \dots, S_{2t}\}$ by successively modifying an existing LFSR to produce increasingly longer sequences. Let us denote k be the iteration index and L_k be the length of the LFSR generated on iteration k. We initialize error locator polynomial as $\Lambda^{(-1)}(x) = 1$ and at the k-th iteration we have,

$$\Lambda^{(k)}(x) = 1 + \Lambda_1^{(k)} x + \Lambda_2^{(k)} x^2 + \dots \Lambda_{L_k}^{(k)} x^{L_k} \tag{5.17}$$

Suppose after $k-1$ iterations, we end up with $\Lambda^{(k-1)}(x)$. On iteration k we first compute

$$\hat{S}_k = -\sum_{i=1}^{L_k - 1} \Lambda_i^{(k-1)} S_{k-i} \tag{5.18}$$

The discrepancy is given by $d_k = S_k - \hat{S}_k = S_k + \sum_{i=1}^{L_k-1} \Lambda_i^{(k-1)} S_{k-i}$. If there is a non-zero discrepancy, the algorithm regenerates a polynomial using the iteration,

$$\Lambda^{(k)}(x) = \Lambda^{(k-1)}(x) + d_{k-1}^{-1} d_k x \Lambda^{(k-2)}(x) \tag{5.19}$$

The iterations continue till the LFSR produces the whole set of syndromes. The final answer will be the error locator polynomial.

More about BM algorithm's derivations of iterations and the details of this method can be found in [2]. BM algorithm is a very efficient way to compute the error locator polynomial particularly for binary codes in which some of the update steps can be merged into single step. If the decoded code is non-binary, we still need to compute the error values. This can be accomplished by *Forney's algorithm* which we will be described next. We start with the following definition and *Lemma*.

Definition 7: *Syndrome polynomial is a polynomial representation of the computed syndrome values as follows,*

$$S(x) = S_1 + S_2 x + \cdots + S_{2t} x^{2t-1} \tag{5.20}$$

Lemma 1: *We have the following fact*

$$(1-x^{2t}) = (1-x)(1+x+x^2+\cdots+x^{2t-1}) = (1-x)\sum_{j=0}^{2t-1} x^j \tag{5.21}$$

Now we define the error-evaluator polynomial $\phi(x) = S(x)\Lambda(x)(\mod x^{2t})$. We observe,

$$\begin{aligned}
\phi(x) &= \left(\sum_{j=1}^{2t} S_j x^{j-1}\right)\prod_{i=1}^{v}(1-X_i x) \quad (\mod x^{2t}) && (5.22)\\
&= \left(\sum_{j=1}^{2t}\sum_{s=1}^{v} Y_s X_s^j x^{j-1}\right)\prod_{i=1}^{v}(1-X_i x) \quad (\mod x^{2t}) && (5.23)\\
&= \sum_{s=1}^{v} Y_s X_s \sum_{j=1}^{2t}(xX_s)^{j-1}\prod_{i=1}^{v}(1-X_i x) \quad (\mod x^{2t}) && (5.24)\\
&= \sum_{s=1}^{v} Y_s X_s \underbrace{(1-xX_s)\sum_{j=0}^{2t-1}(xX_s)^j}_{(1-(xX_s)^{2t}) \text{ by } \textit{Lemma 1}}\prod_{i\neq s}^{v}(1-X_i x) \quad (\mod x^{2t}) && (5.25)\\
&= \sum_{s=1}^{v} Y_s X_s \prod_{i\neq s}^{v}(1-X_i x) \quad (\mod x^{2t}) && (5.26)
\end{aligned}$$

where we used the fact that $(xX_s)^{2t} \equiv 0(\mod x^{2t})$. Now we have the necessary tools to state the following theorem.

Theorem 7: *(Forney's algorithm)*

$$Y_s = -\frac{\phi(X_s^{-1})}{\Lambda'(X_s^{-1})} = -\frac{S(X_s^{-1})\Lambda(X_s^{-1})}{\Lambda'(X_s^{-1})} \tag{5.27}$$

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

PROOF: Note that the derivative of $\Lambda(.)$ can be found as

$$\Lambda'(x) = \frac{d}{dx}\prod_{i=1}^{v}(1-X_i x) = -\sum_{i=1}^{v} X_i \prod_{j\neq i}(1-X_j x) \tag{5.28}$$

Therefore, we have

$$\Lambda'(X_k^{-1}) = -X_k \prod_{j\neq k}(1-X_j X_k^{-1}) \tag{5.29}$$

Let us substitute X_k^{-1} in $\phi(x)$ found in Eqn. 5.26 and have

$$\phi(X_k^{-1}) = \sum_{s=1}^{v} Y_s X_s \prod_{i \neq s}^{v} (1 - X_i X_k^{-1}) \tag{5.30}$$

$$= Y_k X_k \prod_{i \neq k}^{v} (1 - X_i X_k^{-1}) \tag{5.31}$$

$$= -Y_k \Lambda'(X_k^{-1}) \tag{5.32}$$

which completes the proof. □.

Forney's algorithm leads to an important equation called "key equation" now on, i.e., $\phi(x) = S(x)\Lambda(x)(\mod x^{2t})$. We shall see that there is another efficient way for the computation of $\Lambda(x)$ by solving the "key equation" through Extended Euclidean algorithm.

5.1.1 The key equation and extended Euclidean algorithm

The extended Euclidan algorithm is an extension to the original Euclidian algorithm that is used to find the greatest common divisor (*GCD*) for two numbers a and b or two polynomials $a(x)$ and $b(x)$. This extended version also finds two integers or polynomials $\delta(x)$ and $\omega(x)$ such that

$$\delta(x)a(x) + \omega(x)b(x) = GCD(a(x), b(x)) \tag{5.33}$$

Let us remember the original Euclidian algorithm by way of an example. Suppose that $a = 363$ and $b = 57$. Then, we have the following successive division operations and corresponding remainders. In each step, we carry the divisor of the previous step as the divident and the remainder as the new divisor for the current step. The algorithm halts when the remainder hits zero. The last divisor value of the algorithm is the greatest common divisor.

Divident	Divisor	Quotient	Remainder
-	-	-	$r_1 = 363$
-	-	-	$r_2 = 57$
363	57	$q_3 = 6$	$r_3 = 21$
57	21	$q_4 = 2$	$r_4 = 15$
21	15	$q_5 = 1$	$r_5 = 6$
15	6	$q_6 = 2$	$r_6 = 3$
6	**3** = *GCD*(363,57)	$q_7 = 2$	$r_7 = 0$

From this table, we observe that $r_{i-2} = r_i + q_i r_{i-1}$ for $i > 2$. Let us express the remainder in each step in terms of a and b, i.e., $r_i = a\delta_i + b\omega_i$. The reason for defining this relationship is that at the last step before the remainder becomes zero, $r_6 = GCD(363,57)$ in our example.

Note that for $i = 1$ and $i = 2$, we have $r_1 = 363 = a$ implies $(\delta_1, \omega_1) = (1,0)$ and $r_2 = 57 = b$ implies $(\delta_2, \omega_2) = (0,1)$. Using this definition, let us observe the following set of equations by substitution

$$r_i = r_{i-2} + q_i r_{i-1} \tag{5.34}$$
$$= (a\delta_{i-2} + b\omega_{i-2}) - q_i(a\delta_{i-1} + b\omega_{i-1}) \tag{5.35}$$
$$= a(\delta_{i-2} - q_i\delta_{i-1}) + b(\omega_{i-2} - q_i\omega_{i-1}) \tag{5.36}$$

Thus, the coefficients for $i > 2$ can be found using the following recursive equations,

$$\delta_i = \delta_{i-2} - q_i\delta_{i-1} \tag{5.37}$$
$$\omega_i = \omega_{i-2} - q_i\omega_{i-1} \tag{5.38}$$

Therefore this algorithm simultaneously solves δ_i, ω_i and r_i, which at the last iteration equals to $GCD(a,b)$. Similar reasoning can be used to extend this idea to polynomials. After giving a background refresher for the extended Euclidian algorithm, let us consider our key equation again in this context.

$$\theta(x)x^{2t} + \Lambda(x)S(x) = \phi(x)(\mod x^{2t}) \tag{5.39}$$

where $\theta(x)$ is some arbitrary polynomial multiplied by x^{2t} to satisfy the above equation in modulo x^{2t}. Now let us have $a(x) = x^{2t}$ and $b(x) = S(x)$ in our context of previously defined extended Euclidian algorithm. Let also $\delta_k = \theta^{[k]}(x)$ and $\omega_k = \Lambda^{[k]}(x)$ for $k > 2$ with the initial conditions $\phi^{[1]}(x) = x^{2t}$ and $\phi^{[2]}(x) = S(x)$. Then, we step through the algorithm to obtain sequence of polynomials $\theta^{[k]}(x), \Lambda^{[k]}(x), \phi^{[2]}(x)$ satisfying

$$\theta^{[k]}(x)x^{2t} + \Lambda^{[k]}(x)S(x) = \phi^{[k]}(x)(\mod x^{2t}) \tag{5.40}$$

where

$$\Lambda^{[k]}(x) = \Lambda^{[k-2]}(x) - Q^{[k]}(x)\Lambda^{[k-1]}(x) \tag{5.41}$$
$$\theta^{[k]}(x) = \theta^{[k-2]}(x) - Q^{[k]}(x)\theta^{[k-1]}(x) \tag{5.42}$$

and $Q^{[k]}(x)$ is the quotient of the division $\phi^{[k-2]}(x)/\phi^{[k-1]}(x)$. As can be seen, as we step through the algorithm the order of $\phi^{[k]}(x)$ decreases whereas the order of $\Lambda^{[k]}(x)$ increases. We cease the algorithm when $\phi^{[k]}(x)$ has degree less than the degree of $\Lambda^{[k]}(x)$.

5.1.2 Finding the roots of $\Lambda(x)$

Previously, we have shown efficient methods for computing the coefficients of $\Lambda(x)$, i.e., $\{\Lambda_v, \Lambda_{v-1}, \ldots, \Lambda_1\}$. However, the error locations are the reciprocals of the roots of this polynomial. In other words, an extra work must be done to compute the roots of the error locator polynomial $\Lambda(x)$. In theory, we can evaluate $\Lambda(x)$ at $x = \beta$ for each $\beta \in GF(q)$. Those elements

of the field that results in null shall be the roots of the polynomial. From our introduction to finite fields, we observed that each element of the field can be expressed as some power of the primitive element α. We have the following computations for an element α^i for $0 \leq i \leq q-3$,

$$\Lambda(\alpha^i) = \Lambda_\nu(\alpha^i)^\nu + \Lambda_{\nu-1}(\alpha^i)^{\nu-1} + \cdots + \Lambda_2(\alpha^i)^2 + \Lambda_1(\alpha^i) + 1 \tag{5.43}$$

Now, let us evaluate it at the next element in the field α^{i+1}

$$\Lambda(\alpha^{i+1}) = \Lambda_\nu(\alpha^i)^\nu\alpha^\nu + \Lambda_{\nu-1}(\alpha^i)^{\nu-1}\alpha^{\nu-1} + \cdots + \Lambda_2(\alpha^i)^2\alpha^2 + \Lambda_1(\alpha^i)\alpha^1 + 1 \tag{5.44}$$

We immediately see the relationship

$$\Lambda_{j,i+1} = \Lambda_{j,i}\alpha^j \text{ where } \Lambda_{j,i} = \Lambda_j(\alpha^j)^i \text{ for } 1 \leq j \leq \nu. \tag{5.45}$$

and we argue that the computations can simply be iterated based on the previous evaluation result. When implemented in hardware, this approach significantly reduces the complexity, as all multiplications consist of one variable and one constant, rather than two variables as in the brute-force approach. This observation was due to **Chien** who devised this search algorithm for efficiency in 1964 [3].

5.1.3 Galois Field Frequency Domain Decoding

We represent the polynomial $v(x) = v_1 + v_2x + \cdots + v_nx^{n-1}$ by the coefficient vector as $\mathbf{v} = (v_1, v_1, \ldots, v_n)$ and its GF-discrete fourier transform (GF-DFT) as $\widehat{\mathbf{v}} = (\widehat{v}_1, \widehat{v}_2, \ldots, \widehat{v}_n)$, defined by

$$\widehat{v}_j = v(\alpha^j) = \sum_{i=1}^{n} v_i\alpha^{j(i-1)}, \quad \text{for } j = 1, 2, \ldots, n \tag{5.46}$$

It is worth noting that DFT is injective mapping and its inverse (GF-IDFT) can be used to recover $\mathbf{v}$ from $\widehat{\mathbf{v}}$, which is defined by

$$v_i = \frac{1}{n}v(\alpha^{-i}) = \frac{1}{n}\sum_{j=1}^{n} \widehat{v}_j\alpha^{-j(i-1)}, \quad \text{for } i = 1, 2, \ldots, n \tag{5.47}$$

Remember that our polynomial formulation $r(x) = c(x) + e(x)$ can be written in a vector form as $\mathbf{r} = \mathbf{c} + \mathbf{e}$. Using the above definition and taking the GF-DFT of both sides, we obtain the transform domain equation $\mathbf{R} = \mathbf{C} + \mathbf{E}$. Here note that the first $2t$ elements of $\mathbf{R}$ (and therefore of $\mathbf{E} = (E_1, E_2, \ldots, E_n)$) are given by the set of syndromes,

$$E_1 = S_1 = r(\alpha), E_2 = S_2 = r(\alpha^2), \ldots, E_{2t} = S_{2t} = r(\alpha^{2t}) \tag{5.48}$$

If we knew the other $n-2t$ of the entries of $\mathbf{E}$, we could have taken the GF-IDFT to obtain $e(x)$. The way to obtain this missing information is to use the error locator polynomial coefficients $\mathbf{\Lambda}$. if we take the n-point inverse transform, we have

$$\lambda_i = \frac{1}{n}\Lambda(\alpha^{-i}) = \frac{1}{n}\sum_{j=1}^{\nu}\Lambda_{j-1}\alpha^{-i(j-1)} \tag{5.49}$$

At the error locations $\{i_s : s = 1,2,\ldots,\nu\}$, we have

$$\Lambda(\alpha^{-i_1}) = \Lambda(\alpha^{-i_2}) = \cdots = \Lambda(\alpha^{-i_\nu}) = 0. \tag{5.50}$$

We observe that at the zeros of $\Lambda(x)$, the error polynomial $e(x)$ has non-zero values. Therefore, $\lambda_i e_i = 0$, for $i = 1,2,\ldots,n$ i.e., $\mathbf{\Lambda e}^T = 0$. Multiplication operation in the time domain is a convolution operation in the transform domain. We have,

$$\sum_{k=0}^{\nu}\Lambda_k E_{m-k} = 0, \quad \text{for } m = \nu+1, \nu+2, \ldots, n+\nu \tag{5.51}$$

where $\Lambda_0 = 1$ by definition. Since $E_1, E_2, \ldots, E_{2t}$ are already known, using Eqn. (5.51) we can compute E_{2t+1} by letting $m = 2t+1$,

$$\Lambda_0 E_{2t+1} + \sum_{k=1}^{\nu}\Lambda_k E_{2t+1-k} = 0 \Rightarrow E_{2t+1} = -\sum_{k=1}^{\nu}\Lambda_k E_{2t+1-k} = -\sum_{k=1}^{\nu}\Lambda_k S_{2t+1-k} \tag{5.52}$$

Similarly, the rest of the transform error coefficients ($m > 2t+1$) are calculated recursively as follows,

$$E_m = -\sum_{k=1}^{\nu}\Lambda_k E_{m-k} \tag{5.53}$$

As can be noticed, we also calculated E_{n+1}, E_{n+2},…,$E_{n+\nu}$ in order to check the following equalities to hold. If at least one of them does not hold, the decoder declares a failure [6].

$$E_1 = E_{n+1}, \;\; E_2 = E_{n+2}, \;\; \ldots, \;\; E_\nu = E_{n+\nu} \tag{5.54}$$

Final step of decoding is to take the GF-IDFT of $\mathbf{E}$, the error values will be given by error polynomial coefficients computed as follows,

$$e_i = \frac{1}{n}\sum_{j=1}^{n} E_j \alpha^{-j(i-1)}, \quad \text{for } i = 1,2,\ldots,n \tag{5.55}$$

5.1.4 ERROR & ERASURE DECODING

An erasure in decoding terminology is an error, location of which is known with some probability. On the other hand, the value of the error is certainly not known. For example, channel can provide a soft value of the transmitted symbol due to noise effect. If the soft information is not within the acceptable bounds, we can declare an erasure is found.

Example: If the received word has ν errors and l erasures, we can show that if $2\nu + l < d_{min}$, the decoding will be successful, where d_{min} is the minimum distance of the code.

Let us assume that the received word has ν errors and l erasures with error locators $X_1, X_2, \ldots, X_\nu$ where $X_s = \alpha^{i_s}$ as previously defined. In addition, we have erasure locators $E_1, E_2, \ldots, E_l$ defined by $E_s = \alpha^{j_s}$ where we have erasures at locations $j_1, j_2, \ldots, j_l$. Since the erasure locations are provided to the decoder, we can construct the erasure locator polynomial,

$$\Gamma(x) = \prod_{s=1}^{l} (1 - E_s x) \tag{5.56}$$

Let us compute the syndromes S_j, but now with the presence of erasures. A natural way is to assume that the symbol that is erased to have 0 value after "erasure". In that case, the received word (or the received polynomial $r(x)$) will have zeros in erasure locations $j_1, j_2, \ldots, j_l$. For this to happen, erasure values in those locations must be equal to codeword symbols in the exact same locations i.e., $l_{i_s} = c_{i_s}$. In this case, the syndromes will be given by

$$S_j = r(\alpha^j) = \sum_{s=1}^{\nu} e_{i_s} X_s^j + \sum_{s=1}^{l} l_{i_s} E_s^j \qquad 1 \le j \le 2t \tag{5.57}$$

Now let our syndrome polynomial be

$$S(x) = \sum_{j=1}^{2t} S_j x^{j-1} \tag{5.58}$$

Therefore our key equation becomes,

$$\Lambda(x)\Gamma(x)S(x) = \phi(x) \quad (\mod x^{2t}) \tag{5.59}$$

If we update the definition of syndrome polynomial as $\overline{S}(x) = \Gamma(x)S(x)(\mod x^{2t})$. Since we know $\Gamma(x)$ and $S(x)$, this computation should not be hard. After this definition, it should apparent that the problem is reduced down to the one previously discussed; The key equation to be solved is

$$\Lambda(x)\overline{S}(x) = \phi(x) \quad (\mod x^{2t}) \tag{5.60}$$

We can either apply B-M algorithm or extended Euclidian algorithm to find $\Lambda(x)$. In a number of reported results B-M algorithm is shown to be little more efficient than the extended Euclidian algorithm, although many applications use it because it computes the polynomials simultaneously and for other hardware implementation reasons. After computing $\Lambda(x)$, we generate the polynomial $\Psi(x) = \Lambda(x)\Gamma(x)$, and either use Forney's algorithm or frequency domain decoding to efficiently compute the error and erasure values. Using Forney's algorithm for example, we have

$$e_{i_s} = -\frac{\phi(X_s^{-1})}{\Psi'(X_s^{-1})}, \quad l_{i_s} = -\frac{\phi(E_s^{-1})}{\Psi'(E_s^{-1})}. \tag{5.61}$$

where $\Psi'(x)$ is the ordinary derivative of $\Psi(x)$.

5.2 A WORKING EXAMPLE

Before quitting this section, let us give a working example to show how BM, Chien search, extended Euclidean and Forney's algorithms work all together. Let us assume we use $(7,3)$ RS code defined over $GF(8)$ (using a primitive polynomial x^3+x+1) with the generator polynomial $g(x) = (x-\alpha)(x-\alpha^2)(x-\alpha^3)(x-\alpha^4)$. Suppose we received $r(x) = \alpha^3 x + \alpha^4 x^3$. This code is a two symbol error correcting code (as indicated before) and thus we have the following syndromes:

$$S_1 = r(\alpha) = \alpha^4+1 = \alpha^5, S_2 = r(\alpha^2) = \alpha^5+\alpha^3 = \alpha^2, S_3 = r(\alpha^3) = \alpha^6+\alpha^6 = 0, S_4 = r(\alpha^4) = 1+\alpha^2 = \alpha^6 \tag{5.62}$$

Let us now show how BM algorithm works on the computed syndromes to compute the error locator polynomial. We first set $\Lambda^{(-1)}(x) = 1$ and $\Lambda^{(0)}(x) = 1$. The first discrepancy $d_0 = 1+0$. The following steps are taken to determine $\Lambda(x)$,

- $k = 1$: $S_1 = \alpha^5$, $d_1 = S_1 + 0 = \alpha^5$. $\Lambda^{(1)}(x) = \Lambda^{(0)}(x) + 1^{-1}\alpha^5 x\Lambda^{(-1)}(x) = 1 + \alpha^5 x$

- $k = 2$: $S_2 = \alpha^2$, $d_2 = S_2 + \Lambda_1^{(1)} S_1 = \alpha^2 + \alpha^3 = \alpha^5$. $\Lambda^{(2)}(x) = \Lambda^{(1)}(x) + (\alpha^5)^{-1}\alpha^5 x\Lambda^{(0)}(x) = 1 + \alpha^5 x + x = 1 + \alpha^4 x$.

- $k = 3$: $S_3 = 0$, $d_3 = S_3 + \Lambda_1^{(2)} S_2 + \Lambda_2^{(2)} S_1 = 0 + \alpha^6 + 0$. $\Lambda^{(3)}(x) = \Lambda^{(2)}(x) + (\alpha^5)^{-1}\alpha^6 x\Lambda^{(1)}(x) = 1 + \alpha^4 x + \alpha x(1+\alpha^5 x) = 1 + \alpha^2 x + \alpha^6 x^2$.

- $k = 4$: $S_4 = \alpha^6$, $d_4 = S_4 + \Lambda_1^{(3)} S_3 + \Lambda_2^{(3)} S_2 + \Lambda_3^{(3)} S_1 = \alpha^6 + \alpha = \alpha^5$. $\Lambda^{(4)}(x) = \Lambda^{(3)}(x) + (\alpha^6)^{-1}\alpha^5 x\Lambda^{(2)}(x) = 1 + \alpha^2 x + \alpha^6 x^2 + \alpha^6 x(1+\alpha^4 x) = 1 + x + \alpha^4 x^2$.

Since now the all syndromes can be produced by the LFSR defined by the polynomial $\Lambda(x) = 1+x+\alpha^4 x^2$, BM algorithm stops iterating. Using Chien search, we can find the roots of

this polynomial by iteratively evaluating it at each element of the field. We find that $\Lambda(\alpha^6) = 0$ and $\Lambda(\alpha^4) = 0$. Reciprocals of these roots give us the error locations $\alpha^{-6} = \alpha$ and $\alpha^{-4} = \alpha^3$. Let us construct the syndrome polynomial,

$$S(x) = S_1 + S_2 x + S_3 x^2 + S_4 x^3 = \alpha^5 + \alpha^2 x + \alpha^6 x^3 \tag{5.63}$$

together which the key equation can be computed as follows,

$$\begin{aligned} \phi(x) = S(x)\Lambda(x) &= (\alpha^5 + \alpha^2 x + \alpha^6 x^3)(1 + x + \alpha^4 x^2) \mod (x^4) \\ &= \alpha^5 + \alpha^2 x + \alpha^6 x^3 + \alpha^5 x + \alpha^2 x^2 + \alpha^6 x^4 + \alpha^2 x^2 + \alpha^6 x^3 + \alpha^3 x^5 \mod (x^4) \\ &= \alpha^5 + \alpha^3 x + \alpha^6 x^4 + \alpha^3 x^5 \mod (x^4) \\ &= \alpha^5 + \alpha^3 x \end{aligned} \tag{5.64}$$

Alternatively, we could have used extended Euclidian algorithm to solve for $\Lambda(x)$ and $\phi(x)$ all together. As indicated for $t = 2$, we set $\phi^{[1]}(x) = x^4$ and $\phi^{[2]}(x) = S(x) = \alpha^5 + \alpha^2 x + \alpha^6 x^3$. Based on this initialization, we also implicitly initialized $\Lambda^{[1]}(x) = 0$, $\Lambda^{[2]}(x) = 1$, $\theta^{[1]}(x) = 1$ and $\theta^{[2]}(x) = 0$ to satisfy the key equation in the first two steps of the algorithm: $\theta^{[k]}(x)x^{2t} + \Lambda^{[k]}(x)S(x) = \phi^{[k]}(x)(\mod x^{2t})$. The rest of the steps of the algorithm ($k > 3$) can be summarized as,

- $k = 3$: $Q^{[3]}(x) = \phi^{[1]}(x)/\phi^{[2]}(x) = x^4/(\alpha^6 x^3 + \alpha^2 x + \alpha^5) = \alpha x$, we have

$$\Lambda^{[3]}(x) = \Lambda^{[1]}(x) - Q^{[3]}(x)\Lambda^{[2]}(x) = 0 + \alpha x = \alpha x \tag{5.65}$$

$$\theta^{[3]}(x) = \theta^{[1]}(x) - Q^{[3]}(x)\theta^{[2]}(x) = 1 + 0 = 1 \tag{5.66}$$

$$\phi^{[3]}(x) = x^4 \mod (\alpha^5 + \alpha^2 x + \alpha^6 x^3) = \alpha^3 x^2 + \alpha^6 x \tag{5.67}$$

- $k = 4$: $Q^{[4]}(x) = \phi^{[2]}(x)/\phi^{[3]}(x) = (\alpha^6 x^3 + \alpha^2 x + \alpha^5)/(\alpha^3 x^2 + \alpha^6 x) = \alpha^3 x + \alpha^6$, we have

$$\Lambda^{[4]}(x) = \Lambda^{[2]}(x) - Q^{[4]}(x)\Lambda^{[3]}(x) = 1 + (\alpha^3 x + \alpha^6)(\alpha x) = 1 + x + \alpha^4 x^2 \tag{5.68}$$

$$\theta^{[4]}(x) = \theta^{[2]}(x) - Q^{[4]}(x)\theta^{[3]}(x) = 0 + \alpha^3 x + \alpha^6 = \alpha^3 x + \alpha^6 \tag{5.69}$$

$$\phi^{[4]}(x) = \alpha^5 + \alpha^2 x + \alpha^6 x^3 \mod (\alpha^3 x^2 + \alpha^6 x) = \alpha^3 x + \alpha^5 \tag{5.70}$$

Since the degree of $\Lambda^{[4]}(x) \leq t = 2$, the algorithm stops and the outputs are $\Lambda(x) = 1 + x + \alpha^4 x^2$ and $\phi(x) = \alpha^3 x + \alpha^5 x$ as found by the BM algorithm and polynomial multiplication we performed after we determine $\Lambda(x)$. Some hardware manufacturers prefer extended euclidian algorithm over the BM because of complexity reductions, less iterations and simultaneous calculation of $\Lambda(x)$ and $\phi(x)$.

Next, we check if $\deg\{\phi(x)\} < \deg\{\Psi(x)\} = \deg\{\Lambda(x)\}$. Since in our example we have $\deg\{\phi(x)\} = 1 < \deg\{\Lambda(x)\} = 2$, we continue finding the error values. Otherwise the decoder declares a “decoding failure". Final step of decoding is to find the error values.

Using Forney's algorithm: Since $\Lambda'(x) = 1$, we compute the error values as follows,

$$e_2 = \phi(\alpha^6) = \alpha^5 + \alpha^3\alpha^6 = \alpha^3 \tag{5.71}$$
$$e_4 = \phi(\alpha^4) = \alpha^5 + \alpha^3\alpha^4 = \alpha^4 \tag{5.72}$$

Using Frequrency domain decoding: We already know $E_1 = \alpha^5$, $E_2 = \alpha^2$,$E_3 = 0$ and $E_4 = \alpha^6$. The rest of the values can be computed as follows,

$$E_5 = -\Lambda_1 S_4 - \Lambda_2 S_3 = \alpha^6 + \alpha^4.0 = \alpha^6 \tag{5.73}$$
$$E_6 = -\Lambda_1 S_5 - \Lambda_2 S_4 = \alpha^6 + \alpha^4.\alpha^6 = \alpha^4 \tag{5.74}$$
$$E_7 = -\Lambda_1 S_6 - \Lambda_2 S_5 = \alpha^4 + \alpha^4.\alpha^6 = \alpha^6 \tag{5.75}$$
$$E_8 = -\Lambda_1 S_7 - \Lambda_2 S_6 = \alpha^6 + \alpha^4.\alpha^4 = \alpha^5 \tag{5.76}$$
$$E_9 = -\Lambda_1 S_8 - \Lambda_2 S_7 = \alpha^5 + \alpha^4.\alpha^6 = \alpha^2 \tag{5.77}$$

Thus, $\mathbf{E} = [\alpha^5\ \alpha^2\ 0\ \alpha^6\ \alpha^6\ \alpha^4\ \alpha^6\ \alpha^5\ \alpha^2]$. Since the periodicity is present, the decoding is successful. The error polynomial coefficients can be found GF-IDFT. They can be computed to be $\mathbf{e} = [0\ \alpha^3\ 0\ \alpha^4\ 0\ 0\ 0\ 0\ 0]$.

Therefore the error polynomial is given by $e(x) = e_1 + e_2x + \cdots + e_nx^{n-1} = \alpha^3x + \alpha^4x^3$ and the decoded codeword is $c(x) = r(x) + e(x) = \alpha^3x + \alpha^4x^3 + \alpha^3x + \alpha^4x^3 = 0$. Our final note about BM algorithm is that there might be major simplifications to this algorithm as for binary codes, at the even iterations of the algorithm the gaps can be shown to be zero and hence one can skip those steps to have computational savings.

5.3 Computational complexity of the decoding algorithms

For an (n,k,d) RS code, direct computation of syndromes requires $O(dn) = O(n^2)$ arithmetic field operations. Faster algorithms can reduce this down to $O(n\log^2 n\log\log n)$. The roots of $\Lambda(x)$ can be found by Chien search with time complexity around $O(\nu n)$ arithmetic field operations. The extended euclidian algorithm requires $O(\nu d) = O(\nu n)$ and in cases where this algorithm can be accelerated requires $O(n\log^2 n\log\log n)$ arithmetic operations. With novel advanced futures of simultaneous polynomial evaluations at n points, Forney's algorithm possess around the same time complexity of extended Euclidian algorithm. Finally, accelerated BM algorithm through recursions are reported to require $O(n\log^2 n\log\log n)$ arithmetic field operations. The details about these figures can be found in [7] and the references therein.

For a $(n,k,2t+1)$ RS code decoding consisting of syndrome computation, Key equation solver (BM or an Extended Euclidian algorithm), Chien search and Forney's algorithm, we can report computational complexity for each in terms of there basic operations: 1) Multiplications, 2)Additions and 3) Inversions. Such is summarized in Table 5.1 based on the references [8] and [9]. For example it must be quite easy to see that using a Horner's method (See Problem 1) a syndrome computation require us to do $(n-1)$ additions and multiplications and

Operation	Multiplications	Additions	Inversions
Syndrome Computation	$2t(n-1)$	$2t(n-1)$	0
Key Equation Solver	$4t(2t+1)$	$2t(2t+1)$	0
Chien Search	$n(t-1)$	nt	0
Forney's formula	$2t^2$	$t(2t-1)$	t
Total	$3nt+10t^2-n+6t$	$3nt+6t^2-t$	t

Table 5.1: Computational complexity of decoding operations for BCH codes/RS codes.

since we have $2t$ syndromes, the total amount if additions and multiplications is $2t(n-1)$ as predicted by the table.

It might be of interest to express these figures in terms of a basic operation such as binary XOR. Let us make few assumptions in order to express the building block of the decoding operation in terms of XORs. We assume multiplications and inversions have the same complexity in $GF(2^m)$. We further assume one multiplication is equivalent to $2m$ additions. One $GF(2^m)$ addition operation is equivalent to m independent $GF(2)$ additions and without loss of generality XOR and AND gates gave the same complexity. Therefore, the overall complexity in terms of XOR-only operations is given by

$$C = 2m^2(N_{multiplications} + N_{inversions}) + mN_{additions} \tag{5.78}$$

where $N_{multiplications}$, $N_{inversions}$ and $N_{additions}$ denote the number of multiplications, inversions and additions, respectively. Using the results of Table 5.1 and Eqn. (5.78), one can express the overall decoding complexity in terms of rough binary XOR operations.

Such computational complexity figures can be extended to hardware. However, this will be function of the hardware architecture and implementation details of the constituent functional blocks of the decoder. For example, [9] considered hypersystolic architecture. Other architectures can give different results.

6 An Efficient RS Code: Cauchy Reed-Solomon Codes

In various coding applications such as data storage, RS codes are used for erasure correction only. However, due to encoding and decoding complexity of these codes, low complexity alternatives are considered. As of yet, we do not know any finite-length linear codes with linear time encoding and decoding while having the MDS property defined over $GF(2)$. One option might be to give a little MDS performance away and obtain codes that allows simple encoding and decoding algorithms.

Another alternative is to maintain the MDS property and simplify the Galois field additions and multiplications by using simple XOR operator. Note that the generator matrix **G**

for RS codes are of the Vandermonde matrix form. A special property about the generator matrix is that every $k \times k$ submatrix of $\mathbf{G}$ is a Vandermonde matrix and is non-singular i.e., invertible. Alternative class of matrices with that property is Cauchy matrices which shall be discussed later. The advantage of Cauchy matrices over Vandermonde matrices is that their determinant and/or inverses are easier (in terms of computation complexity) to find [5].

6.1 BINARY MATRIX/VECTOR REPRESENTATIONS OF FINITE FIELD ELEMENTS

From Table 1.1, it is apparent that every element of the field $GF(q^m)$ has a column binary vector representation of length m bits. We denote $\phi(\beta)$ as column vector representation of the field element β. For example $\phi(\alpha) = [1\ 0\ 0]^T$.

The following construction also shows that such a trivial isomorphism between the finite field elements and the m-bit tuple binary vectors can be extended to matrix representations of the same field elements.

Definition 8: *For any element $\beta \in GF(2^m)$ which is defined by the primitive polynomial $p(X)$, let $\psi(\beta)$ be an $m \times m$ square matrix whose i-th element is the coefficient vector of the polynomial given by $X^{i-1}\beta \mod p(X)$.*

Let us consider the same example from Table 1.1, and let $p(X) = X^3 + X + 1$ be the primitive polynomial that generates the extension field $GF(2^3)$. Apparently $0 \in GF(2^3)$ has all-zero matrix representation. We can also compute the matrix representations of the rest of the elements as follows:

$$\alpha^0 \Rightarrow \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \alpha^1 \Rightarrow \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \alpha^2 \Rightarrow \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}, \alpha^3 \Rightarrow \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix} \tag{6.1}$$

$$, \alpha^4 \Rightarrow \begin{bmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}, \alpha^5 \Rightarrow \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix}, \alpha^6 \Rightarrow \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \tag{6.2}$$

The main reason for matrix representations is its suitability for defining Galois field additions and multiplications in binary domain i.e., vector/matrix additions and multiplications. The following theorem characterizes that.

Theorem 8: *The matrix representation $\psi(.)$ characterizes an isomorphism from $GF(2^m)$ to $\psi(GF(2^m))$. Particularly,*

- *$\psi(0)$ is all-zero matrix and $\psi(1)$ is the identity matrix.*
- *$\psi(.)$ is injective.*
- *For $\beta_1 \in GF(2^m)$ and $\beta_2 \in GF(2^m)$, we have $\psi(\beta_1 + \beta_2) = \psi(\beta_1) + \psi(\beta_2)$*

- *For $\beta_1 \in GF(2^m)$ and $\beta_2 \in GF(2^m)$, we have $\psi(\beta_1\beta_2) = \psi(\beta_1)\psi(\beta_2)$*

PROOF: Proof can be found in [5].

In a practical application, coefficient vectors of field elements are stored in memory and a sliding window of size 3×3 can be used to generate the matrix representations. The details are left as exercise. In general, the following operations are the least costly to perform Galois field operations,

- Addition: Let $\beta_1 \in GF(2^m)$ and $\beta_2 \in GF(2^m)$, $\phi(\beta_1) + \phi(\beta_2) = \phi(\beta)$.
- Multiplication: Let $\beta_1 \in GF(2^m)$ and $\beta_2 \in GF(2^m)$, $\psi(\beta_1)\phi(\beta_2) = \phi(\beta_1\beta_2)$.

6.2 CAUCHY MATRICES

For erasure-resilient coding, generator matrix being in systematic form is of particular interest. One important reason is that if the failures are located on parity symbols or packets, the data will be readily available which require no decoding operation. That can enhance the access speed to the data.

Theorem 9: *Let $\boldsymbol{C}$ be a $n-k \times k$ matrix defined over $GF(q^m)$. The matrix $[\boldsymbol{I}_k|\boldsymbol{C}]^T$ shall be a valid generator matrix of a systematic MDS code if and only if every square submatrix of $\boldsymbol{C}$ is non-singular.*

PROOF: Proof can be found in [4].

One solution to what Theorem 9 suggests is to use Vandermonde type matrices which generates a type of Reed-solomon codes. However, when erasure-resilient designs are concerned, there has been shown to exist better alternatives such as Cauchy matrices which satisfies the conditions of Theorem 9.

A $n-k \times k$ Cauchy matrix can be defined by defining two disjoint sets $S = \{s_1, \ldots, s_{n-k}\}$ and $R = \{r_1, \ldots, r_k\}$ where s_i and r_j are distinct elements of $GF(2^m)$. The (i, j)-th entry of the Cauchy matrix is given by $1/(s_i + r_j)$. The determinant of a Cauchy matrix can be shown to be non-zero. Since every square sub-matrix of a Cauchy matrix is another Cauchy matrix, the matrix $\mathbf{C}$ of Theorem 9 can be chosen to be a Cauchy matrix. Furthermore, any $k \times k$ sub-matrix of $[\mathbf{I}_k|\mathbf{C}]^T$ is invertible. If $\mathbf{C}$ is chosen to be Cauchy type, then it is shown that the inversion can be done in $O(n^2)$ time instead of $O(n^3)$ for a general Vandermonde matrix inversion. For details, see [5].

7 SOFT DECISION DECODING FOR LINEAR ALGEBRAIC CODES

This section is an advanced topic and will only be summarized. Interested readers shall be directed to the referenced articles for more details. One of the very well known soft decision

algebraic decoding technique was proposed by Forney back in 1966 in the name Generalized Minimum Distance (GMD) decoding which was based on multiple use of hard decision decoding techniques. Later Chase (1972) proposed a method based on flipping some of the least reliable symbols and multiple use of hard decision decoder. Drawback of these mathods was the exponentially increasing complexity.

8 PROBLEMS

Problem 1: Apparent from the text is that all the algorithms presented consist of basic finite field polynomial arithmetic. In this question we would like to evaluate a polynomial at a point. If the polynomial $A(x) = \sum_{i=0}^{n-1} a_i x^i$ is to be evaluated at point x_0, what is the minimum number of additions and multiplications? (Hint: Rewrite $A(x_0) = a_0 + x_0(a_1 + x_0(a_2 + \cdots + x a_{n-1} \ldots))$.)

Problem 2: Construct a non-binary Hamming code over $\mathbb{F}_q$ with parameter r by considering all $(q^m - 1)$ $q-$ ary tuples. Show that the parity check matrix of the non-binary Hamming code is of size $r \times (q^m - 1)/(q-1)$ with 1-symbol of r bits can be corrected. In otherwords, this code can correct r-bit errors maximum.

Problem 3: Consider a (n, k, d) cyclic code with generator polynomial $g(x)$ defined over a field with primitive element α. Assuming that α is the n-th root of unity, show that the parity check polynomial should satisfy $h(x)g(x) = x^n - 1$. Furthermore, if $g(x) = LCM\{f_1(x), \ldots, f_{d-1}(x)\}$ where $f_j(x)$ is an irreducible factor of $x^n - 1$, prove that $h(x) = GCD\{f_1(x), \ldots, f_{d-1}(x)\}$.

Problem 4: Consider the following factorization of the polynomial $X^{23} + 1$ in $GF(2)$,

$$(1 + X)(1 + X^2 + X^4 + X^5 + X^6 + X^{10} + X^{11})(1 + X + X^5 + X^6 + X^7 + X^9 + X^{11}) \tag{8.1}$$

Construct a 3-bit error correcting binary BCH code. Show that minimum distance of the code is 7. Show that the code is perfect. If we extend the code by adding parity at the end of each valid codeword, what will be the minimum distance? Is the code still perfect? Give the parity check matrix of the resultant code.

Problem 5: In this example we will show that we can find analytical result for k, the dimension of a BCH code given that it is primitive and narrow-sense. Let us remember the definition of a primitive, narrow-sense BCH code of length $n = q^m - 1$ defined over $GF(q)$. With the design distance γ, the code has the following generator polynomial of the form

$$g(x) = \prod_{s \in Z} (x - \alpha^s), \quad \text{with } Z = C_1 \cup C_2 \cup \cdots \cup C_{\gamma - 1} \tag{8.2}$$

where $C_x = \{x q^k \mod (q^m - 1) | k \in \mathbb{Z}\}$ are the cyclomatic cosets of x modulo $q^m - 1$. We have shown that a t-error correcting BCH code (a design distance of $2t + 1$) can be constructed with the parameters $(q^m - 1, k, \geq 2t + 1)$ over $GF(q)$.

1.1) Show that the number of cosets are $\gamma - 1$.

1.2) Show that the cardinality of the coset C_x is m if $1 \le x \le q^{\lceil m/2 \rceil}$.

1.3) Now let us exclude all repeated cyclomatic cosets. Show that the number of redundant cosets are $\lfloor (\gamma-1)/q \rfloor$.

1.4) Finally prove that the dimension of the code $k = q^m - 1 - m\lceil (\gamma-1)(1-1/q) \rceil$

Thus, we have proven the theorem that a primitive, narrow-sense BCH code over $GF(q)$ with the design distance γ in the range $\le \gamma \le q^{\lceil m/2 \rceil} + 1$ has the dimension $k = q^m - 1 - m\lceil (\gamma - 1)(1-1/q) \rceil$.

Problem 6: In this problem, we generalize the result of *Theorem 3* for arbitrary q. If C is primitive, narrow sense BCH code of length $q^m - 1$ over $GF(q)$ with the design distance γ in the range $\le \gamma \le q^{\lceil m/2 \rceil} + 1$ has a minimum distance γ or $\gamma + 1$ if

$$Vol_q(q^m - 1, \lfloor (\gamma+1)/2 \rfloor) = \sum_{i=0}^{\lfloor (\gamma+1)/2 \rfloor} \binom{q^m - 1}{i} (q-1)^i > q^{m\lceil (\gamma-1)(1-1/q) \rceil} \tag{8.3}$$

Furthermore, if $\gamma \equiv 0 \mod q$, then the minimum distance is exactly $\gamma + 1$. Let us consider the asymptotical approximation for the hamming sphere in the above equation by letting $\lambda = \lfloor (\gamma+1)/2 \rfloor / (q^m - 1)$ and show that

$$\mathbf{f}(m) = \frac{q^m h_q(\lambda)}{m} > \lceil (\gamma-1)(1-1/q) \rceil \tag{8.4}$$

$$\Rightarrow m = \mathbf{f}^{-1}\left(\lceil (\gamma-1)(1-1/q) \rceil \right) \tag{8.5}$$

If $0 \le \lambda \le 1 - 1/q$ where $h_q(.)$ is the entropy function. This expression is simply saying that for large block lengths and larger alphabets satisfying the above equation, design distance of the BCH code becomes the actual minimum distance of the code. For example, RS codes, being a special case of BCH codes, is an example to this.

Problem 7: As already mentioned in the text, the closed form expressions of Section 5 applies to the form of a generator polynomial $g(x) = (x - \alpha^a)(x - \alpha^{a+1}) \ldots (x - \alpha^{a+2t-1})$ where $a = 1$. Prove the following extended results for general $a \ge 1$.

$$Y_s = -X_s^{-(a-1)} \frac{\phi(X_s^{-1})}{\Lambda'(X_s^{-1})}, \quad \text{for } s = 1, 2, \ldots, \nu \quad \text{(Forney's algorithm)} \tag{8.6}$$

$$e_i = \frac{\alpha^{-(a-1)i}}{n} \sum_{j=1}^{n} E_j \alpha^{-j(i-1)}, \quad \text{for } i = 1, 2, \ldots, n \quad \text{(Frequency domain decoding)} \tag{8.7}$$

Problem 8: Considering the encoding operation of a Cauchy Reed-Solomon Code, we note that the number of 1s in the binary representation of the generator matrix determines

the number of XOR operations i.e., the encoding complexity. Let us define $S = \{0, \alpha^1\}$ and $R = \{\alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6\}$, we have the following cauchy matrix:

$$\mathbf{C} = \begin{bmatrix} 1/(\alpha^2) & 1/(\alpha^3) & 1/(\alpha^4) & 1/(\alpha^5) & 1/(\alpha^6) \\ 1/(\alpha+\alpha^2) & 1/(\alpha+\alpha^3) & 1/(\alpha+\alpha^4) & 1/(\alpha+\alpha^5) & 1/(\alpha+\alpha^6) \end{bmatrix} \tag{8.8}$$

$$= \begin{bmatrix} \alpha^5 & \alpha^4 & \alpha^3 & \alpha^2 & \alpha \\ \alpha^3 & 1 & \alpha^5 & \alpha & \alpha^2 \end{bmatrix} \tag{8.9}$$

which has a binary representation as follows,

$$\mathbf{C} = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

Note that this binary matrix had 54 ones. The interesting question is are there better cauchy generator matrices for a given m (less number of ones i.e., that can lead to lower complexity encoding operation) and the extension field $GF(2^m)$. Does the solution change if we change the primitive polynomial that defines the extension field?

REFERENCES

[1] Irving S. Reed and Gustave Solomon. Polynomial codes over certain Finite Fields. J. Soc. Indust. Appl. Math., 8(2):300–304, 1960.

[2] Richard E. Blahut, “Algebraic codes for data transmission," (2nd ed.), Cambridge. Univ. Press., 2003.

[3] R. T. Chien, “Cyclic Decoding Procedures for the Bose-Chaudhuri-Hocquenghem Codes", *IEEE Transactions on Information Theory*, IT-10 (4): 357Ű363, ISSN 0018-9448 Oct. 1964.

[4] F. J. MacWilliams, N. J. A. Sloane, “The Theory of Error-Correcting Codes," North-Holland, New York, 1977.

[5] J. Blömer, M. Kalfane, M. Karpinski, R. Karp, M. Luby, D. Zuckerman, “An XOR-Based Erasure-Resilient Coding Scheme," ICSI Technical Report No. TR-95-048, August 1995.

[6] R. J. McEliece, “The Decoding of Reed-Solomon Codes," TDA Technical Progress Report 42-95.

[7] Ron M. Roth, “Introduction to Coding Theory," Cambridge. Univ. Press., 2006.

[8] D. Mandelbaum, “On decoding of Reed–Solomon codes," *IEEE Trans. Inf. Theory*, vol. 17, no. 6, pp. 707Ű712, Nov. 1971.

[9] N. Chen and Z. A Yan, “Complexity analysis of reed-solomon decoding over GF(2^m) without using syndromes," *EURASIP Journal on Wireless Communications and Networking,* vol. 2008.