WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link
Page information
Wikidata item
Cite this page

Print/export
Create a book
Download as PDF
Printable version

Languages
Čeština
Deutsch
Français
Bahasa Indonesia
Nederlands
日本語
Português
Русский
Українська
Tiếng Việt
中文
Edit links

BCH code

From Wikipedia, the free encyclopedia
(Redirected from BCH Code)

In coding theory, the **BCH codes** form a class of cyclic error-correcting codes that are constructed using finite fields. BCH codes were invented in 1959 by French mathematician Alexis Hocquenghem, and independently in 1960 by Raj Bose and D. K. Ray-Chaudhuri.[1][2][3] The acronym *BCH* comprises the initials of these inventors' names.

One of the key features of BCH codes is that during code design, there is a precise control over the number of symbol errors correctable by the code. In particular, it is possible to design binary BCH codes that can correct multiple bit errors. Another advantage of BCH codes is the ease with which they can be decoded, namely, via an algebraic method known as syndrome decoding. This simplifies the design of the decoder for these codes, using small low-power electronic hardware.

BCH codes are used in applications such as satellite communications,[4] compact disc players, DVDs, disk drives, solid-state drives[5] and two-dimensional bar codes.

Contents [hide]

Definition and illustration [edit]

Primitive narrow-sense BCH codes [edit]

Given a prime power q and positive integers m and d with $d \le q^m - 1$, a primitive narrow-sense BCH code over the finite field $\mathrm{GF}(q)$ with code length $n = q^m - 1$ and distance at least d is constructed by the following method.

Let α be a primitive element of $\mathrm{GF}(q^m)$. For any positive integer i, let $m_i(x)$ be the minimal polynomial of α^i over $\mathrm{GF}(q)$. The generator polynomial of the BCH code is defined as the least common multiple $g(x) = \mathrm{lcm}(m_1(x),\ldots,m_{d-1}(x))$. It can be seen that $g(x)$ is a polynomial with coefficients in $\mathrm{GF}(q)$ and divides $x^n - 1$. Therefore, the polynomial code defined by $g(x)$ is a cyclic code.

Example [edit]

Let q=2 and m=4 (therefore n=15). We will consider different values of d. There is a primitive root α in $\mathrm{GF}(16)$ satisfying

$$\alpha^4 + \alpha + 1 = 0 \tag{1}$$

its minimal polynomial over $\mathrm{GF}(2)$ is

$$m_1(x) = x^4 + x + 1.$$

The minimal polynomials of the first seven powers of α are

$$m_1(x) = m_2(x) = m_4(x) = x^4 + x + 1,$$

$$m_3(x) = m_6(x) = x^4 + x^3 + x^2 + x + 1,$$
$$m_5(x) = x^2 + x + 1,$$
$$m_7(x) = x^4 + x^3 + 1.$$

The BCH code with $d = 2, 3$ has generator polynomial

$$g(x) = m_1(x) = x^4 + x + 1.$$

It has minimal Hamming distance at least 3 and corrects up to one error. Since the generator polynomial is of degree 4, this code has 11 data bits and 4 checksum bits.

The BCH code with $d = 4, 5$ has generator polynomial

$$g(x) = \mathrm{lcm}(m_1(x), m_3(x)) = (x^4+x+1)(x^4+x^3+x^2+x+1) = x^8+x^7+x^6+x^4+1.$$

It has minimal Hamming distance at least 5 and corrects up to two errors. Since the generator polynomial is of degree 8, this code has 7 data bits and 8 checksum bits.

The BCH code with $d = 8$ and higher has generator polynomial

$$\begin{aligned} g(x) &= \mathrm{lcm}(m_1(x), m_3(x), m_5(x), m_7(x)) \\ &= (x^4 + x + 1)(x^4 + x^3 + x^2 + x + 1)(x^2 + x + 1)(x^4 + x^3 + 1) \\ &= x^{14} + x^{13} + x^{12} + \cdots + x^2 + x + 1. \end{aligned}$$

This code has minimal Hamming distance 15 and corrects 7 errors. It has 1 data bit and 14 checksum bits. In fact, this code has only two codewords: 000000000000000 and 111111111111111.

General BCH codes [edit]

General BCH codes differ from primitive narrow-sense BCH codes in two respects.

First, the requirement that α be a primitive element of $\mathrm{GF}(q^m)$ can be relaxed. By relaxing this requirement, the code length changes from $q^m - 1$ to $\mathrm{ord}(\alpha),$ the order of the element $\alpha.$

Second, the consecutive roots of the generator polynomial may run from $\alpha^c, \ldots, \alpha^{c+d-2}$ instead of $\alpha, \ldots, \alpha^{d-1}.$

Definition. Fix a finite field $GF(q),$ where q is a prime power. Choose positive integers m, n, d, c such that $2 \leq d \leq n,$ $\gcd(n, q) = 1,$ and m is the multiplicative order of q modulo $n.$

As before, let α be a primitive nth root of unity in $GF(q^m),$ and let $m_i(x)$ be the minimal polynomial over $GF(q)$ of α^i for all $i.$ The generator polynomial of the BCH code is defined as the least common multiple

$$g(x) = \mathrm{lcm}(m_c(x), \ldots, m_{c+d-2}(x)).$$

Note: if $n = q^m - 1$ as in the simplified definition, then $\gcd(n, q)$ is automatically 1, and the order of q modulo n is automatically $m.$ Therefore, the simplified definition is indeed a special case of the general one.

Special cases [edit]

- A BCH code with $c = 1$ is called a *narrow-sense BCH code*.
- A BCH code with $n = q^m - 1$ is called *primitive*.

The generator polynomial $g(x)$ of a BCH code has coefficients from $\mathrm{GF}(q).$ In general, a cyclic code over $\mathrm{GF}(q^p)$ with $g(x)$ as the generator polynomial is called a BCH code over $\mathrm{GF}(q^p).$ The BCH code over $\mathrm{GF}(q^m)$ with $g(x)$ as the generator polynomial is called a Reed–Solomon code. In other words, a Reed–Solomon code is a BCH code where the decoder alphabet is the same as the channel alphabet.[6]

Properties [edit]

1. The generator polynomial of a BCH code has degree at most $(d-1)m.$ Moreover, if $q = 2$ and $c = 1,$ the generator polynomial has degree at most $dm/2.$

Proof: each minimal polynomial $m_i(x)$ has degree at most $m.$

Therefore, the least common multiple of $d - 1$ of them has degree at most $(d-1)m.$ Moreover, if $q = 2,$ then $m_i(x) = m_{2i}(x)$ for all $i.$ Therefore, $g(x)$ is the least common multiple of at most $d/2$ minimal polynomials $m_i(x)$ for odd indices $i,$ each of degree at most $m.$

2. A BCH code has minimal Hamming distance at least $d.$ Proof: Suppose that $p(x)$ is a code word with fewer than d non-zero terms. Then

$$p(x) = b_1 x^{k_1} + \cdots + b_{d-1} x^{k_{d-1}}, \text{ where } k_1 < k_2 < \cdots < k_{d-1}.$$

Recall that $\alpha^c, \ldots, \alpha^{c+d-2}$ are roots of $g(x),$ hence of $p(x).$ This implies that $b_1, \ldots, b_{d-1}$ satisfy the following equations, for $i = c, \ldots, c + d - 2:$

$$p(\alpha^i) = b_1 \alpha^{ik_1} + b_2 \alpha^{ik_2} + \cdots + b_{d-1} \alpha^{ik_{d-1}} = 0.$$

In matrix form, we have

$$\begin{bmatrix} \alpha^{ck_1} & \alpha^{ck_2} & \cdots & \alpha^{ck_{d-1}} \\ \alpha^{(c+1)k_1} & \alpha^{(c+1)k_2} & \cdots & \alpha^{(c+1)k_{d-1}} \\ \vdots & \vdots & & \vdots \\ \alpha^{(c+d-2)k_1} & \alpha^{(c+d-2)k_2} & \cdots & \alpha^{(c+d-2)k_{d-1}} \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_{d-1} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

The determinant of this matrix equals

$$\left(\prod_{i=1}^{d-1} \alpha^{ck_i}\right) \det \begin{pmatrix} 1 & 1 & \cdots & 1 \\ \alpha^{k_1} & \alpha^{k_2} & \cdots & \alpha^{k_{d-1}} \\ \vdots & \vdots & & \vdots \\ \alpha^{(d-2)k_1} & \alpha^{(d-2)k_2} & \cdots & \alpha^{(d-2)k_{d-1}} \end{pmatrix} = \left(\prod_{i=1}^{d-1} \alpha^{ck_i}\right) \det(V).$$

The matrix V is seen to be a Vandermonde matrix, and its determinant is

$$\det(V) = \prod_{1 \le i < j \le d-1} (\alpha^{k_j} - \alpha^{k_i}),$$

which is non-zero. It therefore follows that $b_1, \ldots, b_{d-1} = 0,$ hence $p(x) = 0.$

3. A BCH code is cyclic.

Proof: A polynomial code of length n is cyclic if and only if its generator polynomial divides $x^n - 1.$ Since $g(x)$ is the minimal polynomial with roots $\alpha^c, \ldots, \alpha^{c+d-2},$ it suffices to check that each of $\alpha^c, \ldots, \alpha^{c+d-2}$ is a root of $x^n - 1.$ This follows immediately from the fact that α is, by definition, an nth root of unity.

Encoding [edit]

This section is empty. You can help by adding to it. *(March 2013)*

Decoding [edit]

There are many algorithms for decoding BCH codes. The most common ones follow this general outline:

1. Calculate the syndromes *s_j* for the received vector
2. Determine the number of errors *t* and the error locator polynomial *Λ(x)* from the syndromes
3. Calculate the roots of the error location polynomial to find the error locations *X_i*
4. Calculate the error values *Y_i* at those error locations
5. Correct the errors

During some of these steps, the decoding algorithm may determine that the received vector has too many errors and cannot be corrected. For example, if an appropriate value of *t* is not found, then the correction would fail. In a truncated (not primitive) code, an error location may be out of range. If the received vector has more errors than the code can correct, the decoder may unknowingly produce an apparently valid message that is not the one that was sent.

Calculate the syndromes [edit]

The received vector R is the sum of the correct codeword C and an unknown error vector $E.$ The syndrome values are formed by considering R as a polynomial and evaluating it at $\alpha^c, \ldots, \alpha^{c+d-2}.$ Thus the syndromes are[7]

$$s_j = R(\alpha^j) = C(\alpha^j) + E(\alpha^j)$$

for $j = c$ to $c + d - 2.$ Since α^j are the zeros of $g(x),$ of which $C(x)$ is a multiple, $C(\alpha^j) = 0.$ Examining the syndrome values thus isolates the error vector so one can begin to solve for it.

If there is no error, $s_j = 0$ for all $j.$ If the syndromes are all zero, then the decoding is done.

Calculate the error location polynomial [edit]

If there are nonzero syndromes, then there are errors. The decoder needs to figure out how many errors and the location of those errors.

If there is a single error, write this as $E(x) = e\,x^i,$ where i is the location of the error and e is its magnitude. Then the first two syndromes are

$$\begin{aligned} s_c &= e\,\alpha^{c\,i} \\ s_{c+1} &= e\,\alpha^{(c+1)\,i} = \alpha^i s_c \end{aligned}$$

so together they allow us to calculate e and provide some information about i (completely determining it in the case of Reed–Solomon codes).

If there are two or more errors,

$$E(x) = e_1 x^{i_1} + e_2 x^{i_2} + \cdots$$

It is not immediately obvious how to begin solving the resulting syndromes for the unknowns e_k and $i_k.$ First step is finding locator polynomial

$\Lambda(x) = \prod_{j=1}^{t}(x\alpha^{i_j} - 1)$ compatible with computed syndromes and with minimal possible $t.$

Two popular algorithms for this task are:

1. Peterson–Gorenstein–Zierler algorithm
2. Berlekamp–Massey algorithm

Peterson–Gorenstein–Zierler algorithm [edit]

Peterson's algorithm is the step 2 of the generalized BCH decoding procedure. Peterson's algorithm is used to calculate the error locator polynomial coefficients $\lambda_1, \lambda_2, \dots, \lambda_v$ of a polynomial

$$\Lambda(x) = 1 + \lambda_1 x + \lambda_2 x^2 + \cdots + \lambda_v x^v.$$

Now the procedure of the Peterson–Gorenstein–Zierler algorithm.[8] Expect we have at least 2*t* syndromes $s_c,\ldots,s_{c+2t-1}$. Let $v = t$.

- Start by generating the $S_{v\times v}$ matrix with elements that are syndrome values

$$S_{v\times v} = \begin{bmatrix} s_c & s_{c+1} & \dots & s_{c+v-1} \\ s_{c+1} & s_{c+2} & \dots & s_{c+v} \\ \vdots & \vdots & \ddots & \vdots \\ s_{c+v-1} & s_{c+v} & \dots & s_{c+2v-2} \end{bmatrix}.$$

- Generate a $c_{v\times 1}$ vector with elements

$$C_{v\times 1} = \begin{bmatrix} s_{c+v} \\ s_{c+v+1} \\ \vdots \\ s_{c+2v-1} \end{bmatrix}.$$

- Let Λ denote the unknown polynomial coefficients, which are given by

$$\Lambda_{v\times 1} = \begin{bmatrix} \lambda_v \\ \lambda_{v-1} \\ \vdots \\ \lambda_1 \end{bmatrix}.$$

- Form the matrix equation

$$S_{v\times v}\Lambda_{v\times 1} = -C_{v\times 1}.$$

- If the determinant of matrix $S_{v\times v}$ is nonzero, then we can actually find an inverse of this matrix and solve for the values of unknown Λ values.
- If $\det(S_{v\times v}) = 0,$ then follow

```
if $v = 0$
then
      declare an empty error locator polynomial
      stop Peterson procedure.
end
set $v \leftarrow v - 1$
continue from the beginning of Peterson's decoding by making smaller $S_{v\times v}$
```

- After you have values of Λ, you have with you the error locator polynomial.
- Stop Peterson procedure.

Factor error locator polynomial [edit]

Now that you have the $\Lambda(x)$ polynomial, its roots can be found in the form $\Lambda(x) = (\alpha^{i_1}x - 1)(\alpha^{i_2}x - 1)\cdots(\alpha^{i_v}x - 1)$ by brute force for example using the Chien search algorithm. The exponential powers of the primitive element α will yield the positions where errors occur in the received word; hence the name 'error locator' polynomial.

The zeros of Λ(*x*) are α^{-i_1}, ..., α^{-i_v}.

Calculate error values [edit]

Once the error locations are known, the next step is to determine the error values at those locations. The error values are then used to correct the received values at those locations to recover the original codeword.

For the case of binary BCH, (with all characters readable) this is trivial; just flip the bits for the received word at these positions, and we have the corrected code word. In the more general case, the error weights e_j can be determined by solving the linear system

$$s_c = e_1\alpha^{c\,i_1} + e_2\alpha^{c\,i_2} + \cdots$$
$$s_{c+1} = e_1\alpha^{(c+1)\,i_1} + e_2\alpha^{(c+1)\,i_2} + \cdots$$
$$\vdots$$

Forney algorithm [edit]

However, there is a more efficient method known as the Forney algorithm.

Let $S(x) = s_c + s_{c+1}x + s_{c+2}x^2 + \cdots + s_{c+d-2}x^{d-2}$.

Let $v \leq d-1, \lambda_0 \neq 0,$ and $\Lambda(x) = \sum_{i=0}^{v} \lambda_i x^i = \lambda_0 \cdot \prod_{k=0}^{v} (\alpha^{-i_k}x - 1)$.

Let $\Omega(x) = S(x)\,\Lambda(x) \pmod{x^{d-1}}$ be the error evaluator polynomial[9]

Let $\Lambda'(x) = \Sigma_{i=1}^{v} i \cdot \lambda_i x^{i-1}$, where $i \cdot x$ denotes here $\sum_{k=1}^{i} x$ rather than multiplying in the field.

Than if syndromes could be explained by an error word, which could be nonzero only on positions i_k, then error values are

$$e_k = -\frac{\alpha^{i_k}\Omega(\alpha^{-i_k})}{\alpha^{c\cdot i_k}\Lambda'(\alpha^{-i_k})}.$$

For narrow-sense BCH codes, *c* = 1, so the expression simplifies to:

$$e_k = -\frac{\Omega(\alpha^{-i_k})}{\Lambda'(\alpha^{-i_k})}.$$

Explanation of Forney algorithm computation [edit]

It is based on Lagrange interpolation and techniques of generating functions.

Look at $S(x)\Lambda(x)$. Let for simplicity $\lambda_k = 0$ for $k > v$, and $s_k = 0$ for $k > c+d-2$.

Then $S(x)\Lambda(x) = \sum_{j=0}^{\infty}\sum_{i=0}^{j} s_{j-i+1}\lambda_i x^j$.

$$S(x) = \sum_{i=0}^{d-2}\sum_{j=1}^{v} e_j\alpha^{(c+i)\cdot i_j}x^i = \sum_{j=1}^{v} e_j\alpha^{ci_j}\sum_{i=0}^{d-2}(\alpha^{i_j})^i x^i = \sum_{j=1}^{v} e_j\alpha^{ci_j}\frac{(x\alpha^{i_j})^{d-1}-1}{x\alpha^{i_j}-1}.$$
$$S(x)\Lambda(x) = S(x)\lambda_0\prod_{\ell=1}^{v}(\alpha^{i_\ell}x-1) = \lambda_0\sum_{j=1}^{v} e_j\alpha^{ci_j}\frac{(x\alpha^{i_j})^{d-1}-1}{x\alpha^{i_j}-1}\prod_{\ell=1}^{v}(\alpha^{i_\ell}x-1).$$

We could gain form of polynomial:

$$S(x)\Lambda(x) = \lambda_0\sum_{j=1}^{v} e_j\alpha^{ci_j}((x\alpha^{i_j})^{d-1}-1)\prod_{\ell\in\{1,\dots,v\}\setminus\{j\}}(\alpha^{i_\ell}x-1).$$

We want to compute unknowns e_j, and we could simplify the context by removing the $(x\alpha^{i_j})^{d-1}$ terms. This leads to the error evaluator polynomial

$$\Omega(x) = S(x)\,\Lambda(x) \pmod{x^{d-1}}.$$

Thanks to $v \leq d-1$ we have

$$\Omega(x) = -\lambda_0\sum_{j=1}^{v} e_j\alpha^{ci_j}\prod_{\ell\in\{1,\dots,v\}\setminus\{j\}}(\alpha^{i_\ell}x-1).$$

Look at $\Omega(\alpha^{-i_k})$. Thanks to Λ (the Lagrange interpolation trick) the sum degenerates to only one summand

$$\Omega(\alpha^{-i_k}) = -\lambda_0 e_k\alpha^{c\cdot i_k}\prod_{\ell\in\{1,\dots,v\}\setminus\{k\}}(\alpha^{i_\ell}\alpha^{-i_k}-1).$$

To get e_k we just should get rid of the product. We could compute the product directly from already computed roots α^{-i_j} of Λ, but we could use simpler form.

As formal derivative $\Lambda'(x) = \lambda_0\sum_{j=1}^{v}\alpha^{i_j}\prod_{\ell\in\{1,\dots,v\}\setminus\{j\}}(\alpha^{i_\ell}x-1)$, we get again only one summand in

$$\Lambda'(\alpha^{-i_k}) = \lambda_0\alpha^{i_k}\prod_{\ell\in\{1,\dots,v\}\setminus\{k\}}(\alpha^{i_\ell}\alpha^{-i_k}-1).$$

So finally

$$e_k = -\frac{\alpha^{i_k}\Omega(\alpha^{-i_k})}{\alpha^{c\cdot i_k}\Lambda'(\alpha^{-i_k})}.$$

This formula is advantageous when one computes the formal derivative of Λ form its $\Lambda(x) = \sum_{i=1}^{v}\lambda_i x^i$ form, gaining

$$\Lambda'(x) = \Sigma_{i=1}^{v} i \cdot \lambda_i x^{i-1},$$

where $i \cdot x$ denotes here $\sum_{k=1}^{i} x$ rather than multiplying in the field.

Decoding based on extended Euclidean algorithm [edit]

The process of finding both the polynomial Λ and the error values could be based on the Extended Euclidean algorithm. Correction of unreadable characters could be incorporated to the algorithm easily as well.

Let $k_1, \ldots, k_k$ be positions of unreadable characters. One creates polynomial localising these positions $\Gamma(x) = \prod_{i=1}^{k} (x\alpha^{k_i} - 1).$ Set values on unreadable positions to 0 and compute the syndromes.

As we have already defined for the Forney formula let $S(x) = \sum_{i=0}^{d-2} s_{c+i} x^i.$

Let us run extended Euclidean algorithm for locating least common divisor of polynomials $S(x)\Gamma(x)$ and x^{d-1}. The goal is not to find the least common divisor, but a polynomial $r(x)$ of degree at most $\lfloor (d+k-3)/2 \rfloor$ and polynomials $a(x), b(x)$ such that $r(x) = a(x)S(x)\Gamma(x) + b(x)x^{d-1}$. Low degree of $r(x)$ guarantees, that $a(x)$ would satisfy extended (by Γ) defining conditions for $\Lambda.$

Defining $\Xi(x) = a(x)\Gamma(x)$ and using Ξ on the place of $\Lambda(x)$ in the Fourney formula will give us error values.

The main advantage of the algorithm is that it meanwhile computes $\Omega(x) = S(x)\Xi(x) \bmod x^{d-1} = r(x)$ required in the Forney formula.

Explanation of the decoding process [edit]

The goal is to find a codeword which differs from the received word minimally as possible on readable positions. When expressing the received word as a sum of nearest codeword and error word, we are trying to find error word with minimal number of non-zeros on readable positions. Syndrom s_i restricts error word by condition $s_i = \sum_{j=0}^{n-1} e_j \alpha^{ij}$. We could write these conditions separately or we could create polynomial $S(x) = \sum_{i=0}^{d-2} s_{c+i} x^i$ and compare coefficients near powers 0 to $d-2.$

$$S(x) \overset{\{0,\ldots,d-2\}}{=} E(x) = \sum_{i=0}^{d-2} \sum_{j=0}^{n-1} e_j \alpha^{ij} \alpha^{cj} x^i.$$

Suppose there is unreadable letter on position $k_1,$ we could replace set of syndromes $\{s_c, \ldots, s_{c+d-2}\}$ by set of syndromes $\{t_c, \ldots, t_{c+d-3}\}$ defined by equation $t_i = \alpha^{k_1} s_i - s_{i+1}.$ Suppose for an error word all restrictions by original set $\{s_c, \ldots, s_{c+d-2}\}$ of syndromes hold, than $t_i = \alpha^{k_1} s_i - s_{i+1} = \alpha^{k_1} \sum_{j=0}^{n-1} e_j \alpha^{ij} - \sum_{j=0}^{n-1} e_j \alpha^j \alpha^{ij} = \sum_{j=0}^{n-1} e_j (\alpha^{k_1} - \alpha^j) \alpha^{ij}.$ New set of syndromes restricts error vector $f_j = e_j(\alpha^{k_1} - \alpha^j)$ the same way the original set of syndromes restricted the error vector $e_j.$ Note, that except the coordinate $k_1,$ where $f_{k_1} = 0,$ an f_j is zero, iff e_j is zero. For the goal of locating error positions we could change the set of syndromes in the similar way to reflect all unreadable characters. This shortens the set of syndromes by $k.$

In polynomial formulation, the replacement of syndromes set $\{s_c, \ldots, s_{c+d-2}\}$ by syndromes set $\{t_c, \ldots, t_{c+d-3}\}$ leads to $T(x) = \sum_{i=0}^{d-3} t_{c+i} x^i = \alpha^{k_1} \sum_{i=0}^{d-3} s_{c+i} x^i - \sum_{i=1}^{d-2} s_{c+i} x^{i-1}.$ Therefore $xT(x) \overset{\{1,\ldots,d-2\}}{=} (x\alpha^{k_1} - 1)S(x).$

After replacement of $S(x)$ by $S(x)\Gamma(x)$, one would require equation for coefficients near powers $k, \ldots, d-2.$

One could consider looking for error positions from the point of view of eliminating influence of given positions similarly as for unreadable characters. If we found v positions such that eliminating their influence leads to obtaining set of syndromes consisting of all zeros, than there exists error vector with errors only on these coordinates. If $\Lambda(x)$ denotes the polynomial eliminating the influence of these coordinates, we obtain $S(x)\Gamma(x)\Lambda(x) \overset{\{k+v,\ldots,d-2\}}{=} 0.$

In Euclidean algorithm, we try to correct at most $(d-1-k)/2$ errors (on readable positions), because with bigger error count there could be more codewords in the same distance from the received word. Therefore, for $\Lambda(x)$ we are looking for, the equation must hold for coefficients near powers starting from $k + \lfloor (d-1-k)/2 \rfloor.$

In Forney formula, $\Lambda(x)$ could be multiplied by a scalar giving the same result.

It could happen that the Euclidean algorithm finds $\Lambda(x)$ of degree higher than $(d-1-k)/2,$ having number of different roots equal to its degree, where the Fourney formula would be able to correct errors in all its roots, anyways correcting such many errors could be risky (especially with no other restrictions on received word). Usually after getting $\Lambda(x)$ of higher degree, we decide not to correct the errors. Correction could fail in the case $\Lambda(x)$ has roots with higher multiplicity or the number of roots is smaller than its degree. Fail could be detected as well by Forney formula returning error outside the transmitted alphabet.

Correct the errors [edit]

Using the error values and error location, correct the errors and form a corrected code vector by subtracting error values at error locations.

Decoding examples [edit]

Decoding of binary code without unreadable characters [edit]

Consider a BCH code in GF(2^4) with $d = 7$ and $g(x) = x^{10} + x^8 + x^5 + x^4 + x^2 + x + 1$. (This is used in QR codes.) Let the message to be transmitted be [1 1 0 1 1], or in polynomial notation, $M(x) = x^4 + x^3 + x + 1.$ The "checksum" symbols are calculated by dividing $x^{10}M(x)$ by $g(x)$ and taking the remainder, resulting in $x^9 + x^4 + x^2$ or [1 0 0 0 0 1 0 1 0 0]. These are appended to the message, so the transmitted codeword is [1 1 0 1 1 1 0 0 0 0 1 0 1 0 0].

Now, imagine that there are two bit-errors in the transmission, so the received codeword is [1 0 0 1 1 1 0 0 0 1 1 0 1 0 0]. In polynomial notation:

$$R(x) = C(x) + x^{13} + x^5 = x^{14} + x^{11} + x^{10} + x^9 + x^5 + x^4 + x^2$$

In order to correct the errors, first calculate the syndromes. Taking $\alpha = 0010,$ we have $s_1 = R(\alpha^1) = 1011, s_2 = 1001,$ $s_3 = 1011, s_4 = 1101, s_5 = 0001,$ and $s_6 = 1001.$ Next, apply the Peterson procedure by row-reducing the following augmented matrix.

$$[S_{3\times3}|C_{3\times1}] = \begin{bmatrix} s_1 & s_2 & s_3 & s_4 \\ s_2 & s_3 & s_4 & s_5 \\ s_3 & s_4 & s_5 & s_6 \end{bmatrix} = \begin{bmatrix} 1011 & 1001 & 1011 & 1101 \\ 1001 & 1011 & 1101 & 0001 \\ 1011 & 1101 & 0001 & 1001 \end{bmatrix} \Rightarrow \begin{bmatrix} 0001 & 0000 & 1000 & 0111 \\ 0000 & 0001 & 1011 & 0001 \\ 0000 & 0000 & 0000 & 0000 \end{bmatrix}$$

Due to the zero row, $S_{3\times3}$ is singular, which is no surprise since only two errors were introduced into the codeword. However, the upper-left corner of the matrix is identical to $[S_{2\times2} \mid C_{2\times1}]$, which gives rise to the solution $\lambda_2 = 1000, \lambda_1 = 1011.$ The resulting error locator polynomial is $\Lambda(x) = 1000x^2 + 1011x + 0001,$ which has zeros at $0100 = \alpha^{-13}$ and $0111 = \alpha^{-5}.$ The exponents of α correspond to the error locations. There is no need to calculate the error values in this example, as the only possible value is 1.

Decoding with unreadable characters [edit]

Suppose the same scenario, but the received word has two unreadable characters [1 0 0 ? 1 1 ? 0 0 1 1 0 1 0 0]. We replace the unreadable characters by zeros while creating the polynom reflecting their positions $\Gamma(x) = (\alpha^8 x - 1)(\alpha^{11} x - 1).$ We compute the syndromes $s_1 = \alpha^{-7}, s_2 = \alpha^1, s_3 = \alpha^4, s_4 = \alpha^2, s_5 = \alpha^5,$ and $s_6 = \alpha^{-7}.$ (Using log notation which is independent on GF(2^4) isomorphisms. For computation checking we can use the same representation for addition as was used in previous example. Hexadecimal description of the powers of α are consecutively 1,2,4,8,3,6,C,B,5,A,7,E,F,D,9 with the addition based on bitwise xor.)

Let us make syndrome polynomial $S(x) = \alpha^{-7} + \alpha^1 x + \alpha^4 x^2 + \alpha^2 x^3 + \alpha^5 x^4 + \alpha^{-7} x^5,$ compute $S(x)\Gamma(x) = \alpha^{-7} + \alpha^4 x + \alpha^{-1} x^2 + \alpha^6 x^3 + \alpha^{-1} x^4 + \alpha^5 x^5 + \alpha^7 x^6 + \alpha^{-3} x^7.$

Run the extended Euclidean algorithm:

$$\begin{pmatrix} S(x)\Gamma(x) \\ x^6 \end{pmatrix} = \begin{pmatrix} \alpha^{-7} + \alpha^4 x + \alpha^{-1} x^2 + \alpha^6 x^3 + \alpha^{-1} x^4 + \alpha^5 x^5 + \alpha^7 x^6 + \alpha^{-3} x^7 \\ x^6 \end{pmatrix}$$

$$= \begin{pmatrix} \alpha^7 + \alpha^{-3} x & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x^6 \\ \alpha^{-7} + \alpha^4 x + \alpha^{-1} x^2 + \alpha^6 x^3 + \alpha^{-1} x^4 + \alpha^5 x^5 + (\alpha^7 + \alpha^7) x^6 + (\alpha^{-3} + \alpha^{-3}) x^7 \end{pmatrix}$$

$$= \begin{pmatrix} \alpha^7 + \alpha^{-3} x & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \alpha^4 + \alpha^{-5} x & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \alpha^{-7} + \alpha^4 x + \alpha^{-1} x^2 + \alpha^6 x^3 + \alpha^{-1} x^4 + \alpha^5 x^5 \\ \alpha^{-3} + (\alpha^{-7} + \alpha^3) x + (\alpha^3 + \alpha^{-1}) x^2 + \\ (\alpha^{-5} + \alpha^{-6}) x^3 + (\alpha^3 + \alpha^1) x^4 + (\alpha^{-6} + \alpha^{-6}) x^5 + (\alpha^0 + 1) x^6 \end{pmatrix}$$

$$= \begin{pmatrix} (1 + \alpha^{-4}) + (\alpha^1 + \alpha^2) x + \alpha^7 x^2 & \alpha^7 + \alpha^{-3} x \\ \alpha^4 + \alpha^{-5} x & 1 \end{pmatrix} \begin{pmatrix} \alpha^{-7} + \alpha^4 x + \alpha^{-1} x^2 + \alpha^6 x^3 + \alpha^{-1} x^4 + \alpha^5 x^5 \\ \alpha^{-3} + \alpha^{-2} x + \alpha^0 x^2 + \alpha^{-2} x^3 + \alpha^{-6} x^4 \end{pmatrix}$$

$$= \begin{pmatrix} \alpha^{-3} + \alpha^5 x + \alpha^7 x^2 & \alpha^7 + \alpha^{-3} x \\ \alpha^4 + \alpha^{-5} x & 1 \end{pmatrix} \begin{pmatrix} \alpha^{-5} + \alpha^{-4} x & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \alpha^{-3} + \alpha^{-2} x + \alpha^0 x^2 + \alpha^{-2} x^3 + \alpha^{-6} x^4 \\ (\alpha^7 + \alpha^{-7}) + (\alpha^{-7} + \alpha^{-7} + \alpha^4) x + \\ (\alpha^{-5} + \alpha^{-6} + \alpha^{-1}) x^2 + \\ (\alpha^{-7} + \alpha^{-4} + \alpha^6) x^3 + \\ (\alpha^4 + \alpha^{-6} + \alpha^{-1}) x^4 + (\alpha^5 + \alpha^5) x^5 \end{pmatrix}$$

$$= \begin{pmatrix} \alpha^7 x + \alpha^5 x^2 + \alpha^3 x^3 & \alpha^{-3} + \alpha^5 x + \alpha^7 x^2 \\ \alpha^3 + \alpha^{-5} x + \alpha^6 x^2 & \alpha^4 + \alpha^{-5} x \end{pmatrix} \begin{pmatrix} \alpha^{-3} + \alpha^{-2} x + \alpha^0 x^2 + \alpha^{-2} x^3 + \alpha^{-6} x^4 \\ \alpha^{-4} + \alpha^4 x + \alpha^2 x^2 + \alpha^{-5} x^3 \end{pmatrix}.$$

We have reached polynomial of degree at most 3, and as

$$\begin{pmatrix} -(\alpha^4 + \alpha^{-5} x) & \alpha^{-3} + \alpha^5 x + \alpha^7 x^2 \\ \alpha^3 + \alpha^{-5} x + \alpha^6 x^2 & -(\alpha^7 x + \alpha^5 x^2 + \alpha^3 x^3) \end{pmatrix} \begin{pmatrix} \alpha^7 x + \alpha^5 x^2 + \alpha^3 x^3 & \alpha^{-3} + \alpha^5 x + \alpha^7 x^2 \\ \alpha^3 + \alpha^{-5} x + \alpha^6 x^2 & \alpha^4 + \alpha^{-5} x \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix},$$

we get $\begin{pmatrix} -(\alpha^4 + \alpha^{-5} x) & \alpha^{-3} + \alpha^5 x + \alpha^7 x^2 \\ \alpha^3 + \alpha^{-5} x + \alpha^6 x^2 & -(\alpha^7 x + \alpha^5 x^2 + \alpha^3 x^3) \end{pmatrix} \begin{pmatrix} S(x)\Gamma(x) \\ x^6 \end{pmatrix} = \begin{pmatrix} \alpha^{-3} + \alpha^{-2} x + \alpha^0 x^2 + \\ \alpha^{-2} x^3 + \alpha^{-6} x^4 \\ \alpha^{-4} + \alpha^4 x + \alpha^2 x^2 + \\ \alpha^{-5} x^3 \end{pmatrix}.$

Therefore $S(x)\Gamma(x)(\alpha^3 + \alpha^{-5} x + \alpha^6 x^2) - (\alpha^7 x + \alpha^5 x^2 + \alpha^3 x^3) x^6 = \alpha^{-4} + \alpha^4 x + \alpha^2 x^2 + \alpha^{-5} x^3.$

Let $\Lambda(x) = \alpha^3 + \alpha^{-5} x + \alpha^6 x^2.$ Don't worry that $\lambda_0 \neq 1.$ Find by brute force a root of $\Lambda.$ The roots are $\alpha^2,$ and α^{10} (after finding for example α^2 we can divide Λ by corresponding monom $(x - \alpha^2)$ and the root of resulting monom could be found easily).

Let $\Xi(x) = \Gamma(x)\Lambda(x) = \alpha^3 + \alpha^4 x^2 + \alpha^2 x^3 + \alpha^{-5} x^4$, and let $\Omega(x) = S(x)\Xi(x) \bmod x^6 = \alpha^{-4} + \alpha^4 x + \alpha^2 x^2 + \alpha^{-5} x^3$. Let us look for error values using formula $e_j = -\Omega(\alpha^{-i_j})/\Xi'(\alpha^{-i_j})$, where α^{-i_j} are roots of $\Xi(x)$. $\Xi'(x) = \alpha^2 x^2$. We get
$e_1 = -\Omega(\alpha^4)/\Xi'(\alpha^4) = (\alpha^{-4} + \alpha^{-7} + \alpha^{-5} + \alpha^7)/\alpha^{-5} = \alpha^{-5}/\alpha^{-5} = 1$,
$e_2 = -\Omega(\alpha^7)/\Xi'(\alpha^7) = (\alpha^{-4} + \alpha^{-4} + \alpha^1 + \alpha^1)/\alpha^1 = 0$,
$e_3 = -\Omega(\alpha^{10})/\Xi'(\alpha^{10}) = (\alpha^{-4} + \alpha^{-1} + \alpha^7 + \alpha^{-5})/\alpha^7 = \alpha^7/\alpha^7 = 1$,
$e_4 = -\Omega(\alpha^2)/\Xi'(\alpha^2) = (\alpha^{-4} + \alpha^6 + \alpha^6 + \alpha^1)/\alpha^6 = \alpha^6/\alpha^6 = 1$. Fact, that $e_3 = e_4 = 1$, should not be surprising.

Corrected code is therefore [1 1 0 1 1 1 0 0 0 0 1 0 1 0 0].

Decoding with unreadable characters with a small number of errors [edit]

Let us show the algorithm behaviour for the case with small number of errors. Let the received word is [1 0 0 ? 1 1 ? 0 0 0 1 0 1 0 0].

Again, replace the unreadable characters by zeros while creating the polynom reflecting their positions $\Gamma(x) = (\alpha^8 x - 1)(\alpha^{11} x - 1)$. Compute the syndromes $s_1 = \alpha^4, s_2 = \alpha^{-7}, s_3 = \alpha^1, s_4 = \alpha^1, s_5 = \alpha^0$, and $s_6 = \alpha^2$. Create syndrom polynommial $S(x) = \alpha^4 + \alpha^{-7} x + \alpha^1 x^2 + \alpha^1 x^3 + \alpha^0 x^4 + \alpha^2 x^5$, and $S(x)\Gamma(x) = \alpha^4 + \alpha^7 x + \alpha^5 x^2 + \alpha^3 x^3 + \alpha^1 x^4 + \alpha^{-1} x^5 + \alpha^{-1} x^6 + \alpha^6 x^7$. Let us run the extended Euclidean algorithm:

$$\begin{pmatrix} S(x)\Gamma(x) \\ x^6 \end{pmatrix} = \begin{pmatrix} \alpha^4 + \alpha^7 x + \alpha^5 x^2 + \alpha^3 x^3 + \alpha^1 x^4 + \alpha^{-1} x^5 + \alpha^{-1} x^6 + \alpha^6 x^7 \\ x^6 \end{pmatrix}$$
$$= \begin{pmatrix} \alpha^{-1} + \alpha^6 x & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x^6 \\ \alpha^4 + \alpha^7 x + \alpha^5 x^2 + \alpha^3 x^3 + \alpha^1 x^4 + \alpha^{-1} x^5 + (\alpha^{-1} + \alpha^{-1}) x^6 + (\alpha^6 + \alpha^6) x^7 \end{pmatrix}$$
$$= \begin{pmatrix} \alpha^{-1} + \alpha^6 x & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \alpha^3 + \alpha^1 x & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \alpha^4 + \alpha^7 x + \alpha^5 x^2 + \alpha^3 x^3 + \alpha^1 x^4 + \alpha^{-1} x^5 \\ \alpha^7 + (\alpha^{-5} + \alpha^5) x + (\alpha^{-7} + \alpha^{-7}) x^2 + (\alpha^6 + \alpha^6) x^3 + \\ (\alpha^4 + \alpha^4) x^4 + (\alpha^2 + \alpha^2) x^5 + (\alpha^0 + 1) x^6 \end{pmatrix}$$
$$= \begin{pmatrix} (1 + \alpha^2) + (\alpha^0 + \alpha^{-6}) x + \alpha^7 x^2 & \alpha^{-1} + \alpha^6 x \\ \alpha^3 + \alpha^1 x & 1 \end{pmatrix} \begin{pmatrix} \alpha^4 + \alpha^7 x + \alpha^5 x^2 + \alpha^3 x^3 + \alpha^1 x^4 + \alpha^{-1} x^5 \\ \alpha^7 + \alpha^0 x \end{pmatrix}.$$

We have reached polynomial of degree at most 3, and as

$$\begin{pmatrix} -(1) & \alpha^{-1} + \alpha^6 x \\ \alpha^3 + \alpha^1 x & -(\alpha^{-7} + \alpha^7 x + \alpha^7 x^2) \end{pmatrix} \begin{pmatrix} \alpha^{-7} + \alpha^7 x + \alpha^7 x^2 & \alpha^{-1} + \alpha^6 x \\ \alpha^3 + \alpha^1 x & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix},$$

we get

$$\begin{pmatrix} -(1) & \alpha^{-1} + \alpha^6 x \\ \alpha^3 + \alpha^1 x & -(\alpha^{-7} + \alpha^7 x + \alpha^7 x^2) \end{pmatrix} \begin{pmatrix} S(x)\Gamma(x) \\ x^6 \end{pmatrix} = \begin{pmatrix} \alpha^4 + \alpha^7 x + \alpha^5 x^2 + \alpha^3 x^3 + \alpha^1 x^4 + \alpha^{-1} x^5 \\ \alpha^7 + \alpha^0 x \end{pmatrix}.$$

Therefore,

$$S(x)\Gamma(x)(\alpha^3 + \alpha^1 x) - (\alpha^{-7} + \alpha^7 x + \alpha^7 x^2) x^6 = \alpha^7 + \alpha^0 x.$$

Let $\Lambda(x) = \alpha^3 + \alpha^1 x$. Don't worry that $\lambda_0 \neq 1$. The root of $\Lambda(x)$ is α^{3-1}.

Let $\Xi(x) = \Gamma(x)\Lambda(x) = \alpha^3 + \alpha^{-7} x + \alpha^{-4} x^2 + \alpha^5 x^3$, and $\Omega(x) = S(x)\Xi(x) \mathrm{mod} x^6 = \alpha^7 + \alpha^0 x$. Let us look for error values using formula $e_j = -\Omega(\alpha^{-i_j})/\Xi'(\alpha^{-i_j})$, where α^{-i_j} are roots of polynomial $\Xi(x)$. $\Xi'(x) = \alpha^{-7} + \alpha^5 x^2$. We get $e_1 = -\Omega(\alpha^4)/\Xi'(\alpha^4) = (\alpha^7 + \alpha^4)/(\alpha^{-7} + \alpha^{-2}) = \alpha^3/\alpha^3 = 1$,
$e_2 = -\Omega(\alpha^7)/\Xi'(\alpha^7) = (\alpha^7 + \alpha^7)/(\alpha^{-7} + \alpha^4) = 0/\alpha^5 = 0$,
$e_3 = -\Omega(\alpha^2)/\Xi'(\alpha^2) = (\alpha^7 + \alpha^2)/(\alpha^{-7} + \alpha^{-6}) = \alpha^{-3}/\alpha^{-3} = 1$. The fact that $e_3 = 1$ should not be surprising.

Corrected code is therefore [1 1 0 1 1 1 0 0 0 0 1 0 1 0 0].

Citations [edit]

1. ^ Reed & Chen 1999, p. 189
2. ^ Hocquenghem 1959
3. ^ Bose & Ray-Chaudhuri 1960
4. ^ "Phobos Lander Coding System: Software and Analysis" (PDF). Retrieved 25 February 2012.
5. ^ "Sandforce SF-2500/2600 Product Brief". Retrieved 25 February 2012.
6. ^ Gill n.d., p. 3
7. ^ Lidl & Pilz 1999, p. 229
8. ^ Gorenstein, Peterson & Zierler 1960
9. ^ Gill n.d., p. 47

References [edit]

Primary sources [edit]

- Hocquenghem, A. (September 1959), "Codes correcteurs d'erreurs", *Chiffres* (in French) (Paris) **2**: 147–156
- Bose, R. C.; Ray-Chaudhuri, D. K. (March 1960), "On A Class of Error Correcting Binary Group Codes", *Information and Control* **3** (1): 68–79, doi:10.1016/s0019-9958(60)90287-4, ISSN 0890-5401

Secondary sources [edit]

- Gill, John (n.d.), *EE387 Notes #7, Handout #28* (PDF), Stanford University, pp. 42–45, retrieved April 21, 2010[*dead link*] Course notes are apparently being redone for 2012: http://www.stanford.edu/class/ee387/
- Gorenstein, Daniel; Peterson, W. Wesley; Zierler, Neal (1960), "Two-Error Correcting Bose-Chaudhuri Codes are Quasi-Perfect", *Information and Control* **3** (3): 291–294, doi:10.1016/s0019-9958(60)90877-9
- Lidl, Rudolf; Pilz, Günter (1999), *Applied Abstract Algebra* (2nd ed.), John Wiley
- Reed, Irving S.; Chen, Xuemin (1999), *Error-Control Coding for Data Networks*, Boston, MA: Kluwer Academic Publishers, ISBN 0-7923-8528-4

Further reading [edit]

- Gilbert, W. J.; Nicholson, W. K. (2004), *Modern Algebra with Applications* (2nd ed.), John Wiley
- Lin, S.; Costello, D. (2004), *Error Control Coding: Fundamentals and Applications*, Englewood Cliffs, NJ: Prentice-Hall
- MacWilliams, F. J.; Sloane, N. J. A. (1977), *The Theory of Error-Correcting Codes*, New York, NY: North-Holland Publishing Company
- Rudra, Atri, *CSE 545, Error Correcting Codes: Combinatorics, Algorithms and Applications*, University at Buffalo, retrieved April 21, 2010

Categories: Error detection and correction | Finite fields | Coding theory