Introduction into the Error-Correcting Codes

Oleg Fomenko

February 2025

1 Introduction

Error-correcting codes (ECC) are a group of protocols that wrap the input data extending it with an excessive information that allows recovery of the original data even if the received message contains errors. Such methods are widely used in the network protocols and data storage approaches. Moreover, they found many applications in the modern cryptographic protocols (for example, Reed-Solomon ECC has been applied in the FRI protocol, which is widely used in ZK-STARKs). Additionally, each code allows the determination of what errors exactly appeared in the received message (sometimes, even if the message can not be decoded).

ECC protocols can be divided in two groups: **block codes** and **convolutional codes**. The block codes split information into the blocks of fixed pre-defined size and apply encoding to each block separately, while the convolutional codes work over the input data stream of an arbitrary size. Modern practical block codes allow encoding by polynomial complexity algorithms depending on the block size. Also, most of classic block codes leverage the properties of the finite fields.

In this article we primarily describe the structure of some classic block ECCs, including Walsh-Hadamard code and Reed-Solomon code. Its aim is to give the reader an understanding of how they work and the mathematical primitives they rely on.

2 Preliminaries

First of all let's define the main property of each ECC that directly determines its structure:

Definition 2.1 *For the arbitrary binary strings of fixed length $x, y \in \{0,1\}^n$ we define an **absolute Hamming distance** as the number of elements where our binary strings do not match: $\Delta(x,y) = |\{i : x_i \neq y_i\}|$. We also define the **normalized Hamming distance** as the absolute Hamming distance divided by the length of the string: $\Delta(x,y) = \frac{1}{n}|\{i : x_i \neq y_i\}|$.*

The normalized Hamming distance becomes very useful because of its independence on the codeword length, while the absolute Hamming distance is useful for understanding the practicalities of specific implementation.

Next, we can define what exactly a block ECC is. Basically, it operates over some field ($\mathbb{GF}(2)$ for example) encoding information using some algorithm that differs for each code.

Definition 2.2 *According to the [AB09] for each $\delta \in [0,1]$ we call a function $E : \{0,1\}^n \to \{0,1\}^m$ an **error-correcting code** with distance δ if for each two input strings $x \neq y \in \{0,1\}^n$ the distance between their images is more or equal δ: $\Delta(E(x), E(y)) \geq \delta$. We call $Im(E)$ the set of **codewords** of corresponding code.*

Thus, each block ECC that takes an input word of size n and outputs a codeword of size m can theoretically deal with $t = m - n$ errors. A block ECC also utilizes a relation between its distance and its theoretical correcting capability: $t < \lfloor \frac{\delta - 1}{2} \rfloor$, where δ is an absolute distance. This relation exists because two arbitrary codewords can be decoded if they contain at most t errors. Therefore, the distance between these codewords must be at least $2t$ to enable unique decoding (if it is less then two different codewords with errors can be represented as the same message, so the pre-image cannot be found correctly). While the ECC distance is d, it leads to the relation above (check Figure 1).

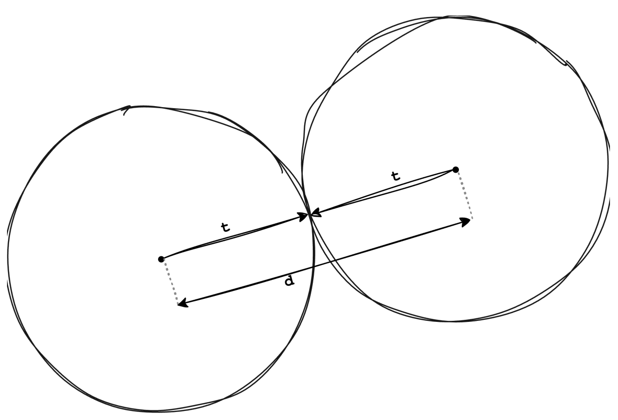

Figure 1: Relation between distance and the number of possible errors

3 Walsh-Hadamard code

The Hadamard code [MS77], named after French mathematician Jacques Hadamard, is an error-correcting block code designed for detecting and correcting errors in message transmissions over highly noisy or unreliable channels. In 1971, NASA utilized this code to send images of Mars from the Mariner 9 space probe back to Earth. This code is also referred to as the Walsh code, Walsh family, or Walsh–Hadamard code, in honor of American mathematician Joseph Leonard Walsh.

Definition 3.1 (Walsh-Hadamard code) *For the binary strings $x, y \in \{0,1\}^n$ let's define $\langle x, y\rangle = \sum_{i=0}^{n-1} x_i y_i \mod 2$. The Walsh-Hadamard code is a function $\mathsf{WH} : \{0,1\}^n \to \{0,1\}^{2^n}$ that maps each input of size n into the string $z \in \{0,1\}^{2^n}$ where $z_y = \langle x, y\rangle$ for $\forall y \in \{0,1\}^n$.*

Encoding. For example, for the input size $n = 2$ we have the following set of the possible codewords:

$$\{(0,0), (0,1), (1,0), (1,1)\}$$

Then, for each input codeword the Walsh-Hadamard code will be:

$$\mathsf{WH}(x) = (\langle x, (0,0)\rangle, \langle x, (0,1)\rangle, \langle x, (1,0)\rangle, \langle x, (1,1)\rangle)$$

For example, taking $x = (0,1)$ we have:

$$\mathsf{WH}((0,1)) = \begin{bmatrix} \langle (0,1), (0,0)\rangle \\ \langle (0,1), (0,1)\rangle \\ \langle (0,1), (1,0)\rangle \\ \langle (0,1), (1,1)\rangle \end{bmatrix} = (1,0,0,1)$$

Decoding. To decode a received codeword, we use the following technique: for each possible input word $x \in \{0,1\}^n$ we calculate the list of $C_x(y) = (-1)^{\langle x,y\rangle}$ for each $y \in \{0,1\}^n$. We also represent our received codeword Y as list of $(-1)^{Y_i}$. For each possible word $u \in \{0,1\}^n$ we calculate $S(u) = \sum_{i=0}^{2^n-1} (-1)^{Y_i} \cdot C_u(i)$. This sum represents the similarity between the word u and the encoded word. If both words have the same sign at position i, the sum increases; otherwise, it decreases. The decoded word is the one with the highest sum.

To define the distance for the Walsh-Hadamard ECC we first need to prove an additional lemma:

Lemma 3.1 (Random subsum principle) *For two binary strings $u \neq v \in \{0,1\}^n$ the* $\Pr[\langle u, x\rangle \neq \langle v, x\rangle] = \frac{1}{2}$ *for the random binary string $x \in \{0,1\}^n$.*

Proof. Note, that $\langle u, x\rangle \neq \langle v, x\rangle$ works if and only if $1 = \langle u, x\rangle + \langle v, x\rangle = \langle (u+v), x\rangle$, where $+$ is an addition by modulo 2 (aka `XOR`). Then, we can rewrite this as $\langle (u+v), x\rangle = \sum_{(u+v)_i \neq 0} x_i \mod 2$. While $u + v \neq 0^n$ from the initial assumption and x is a uniformly random string, the $\sum_{(u+v)_i \neq 0} x_i \mod 2 = 1$ with probability $\frac{1}{2}$. So, finally, $\langle u, x\rangle \neq \langle v, x\rangle$ with probability $\frac{1}{2}$. $\triangle$

Lemma 3.2 *The Walsh-Hadamard code is an error-correcting code with distance* $\frac{1}{2}$.

Proof. Taking two codewords $f(x_1)$ and $f(x_2)$ the absolute distance between them is equal to the number of $y \in \{0,1\}^n$ where $\langle x_1, y\rangle \neq \langle x_2, y\rangle$. We know that this happens in the half of the cases, so for a codeword of size 2^n the distance will be $\Delta(f(x_1), f(x_2)) = \frac{2^{n-1}}{2^n} = \frac{1}{2}$. $\triangle$

This code made a significant impact on the coding theory, mathematics, and theoretical computer science. However, it is impractical for modern applications due to the exponential growth in the size of the codewords.

4 Reed-Solomon code

In information and coding theory, Reed–Solomon ECCs are a group of block error-correcting codes developed by Irving S. Reed and Gustave Solomon in 1960. They are widely used across various applications, including consumer technologies like MiniDiscs, CDs, DVDs, Blu-ray discs, QR codes, and Data Matrix. They also play a crucial role in data transmission systems such as DSL and WiMAX, broadcast technologies like satellite communications, DVB, and ATSC, as well as storage solutions such as RAID6. Essentially, the difference between the variants Reed-Solomon code lies in the assumptions on the basis of which their encoding and decoding algorithms are determined.

Firstly, let's start from the definition of the input data that differs from binary:

Definition 4.1 *For some alphabet* Σ *elements* $x, y \in \Sigma^n$ *we define* $\Delta(x,y) = \frac{1}{n}|\{i : x_i \neq y_i\}|$.

Now we can describe error-correcting codes for the alphabets that differ from binary. Also, let's describe one subclass of the block ECCs to which the Reed-Solomon code belong:

Definition 4.2 (Linear code) *Linear code is a code where the set of codewords forms a linear space:*

– *For each two codewords* c_1, c_2*, the* $c_1 + c_2$ *is also a codeword.*

– *For the codeword* c *and constant* α *the* αc *is also a codeword.*

Theorem 4.1 *Let the **weight** of a codeword be the number of positions where its value is non-zero. Then, the distance of the linear error-correcting code is equal to the minimum possible weight of non-zero codeword.*

Proof. For two codewords c_1, c_2, the $c_1 - c_2$ is also a codeword. Note, that $\Delta(c_1, c_2) = w(c_1 - c_2)$, where $w(x) = |\{i : x_i \neq 0\}|$ is a weight function and subtraction is done by modulo 2 over binary elements. Thus, the distance of the linear error-correcting code will be equal to the minimum possible distance between two non-equal codewords that is also equal to the minimum possible weight of the non-zero codeword:

$$\delta = \min_{c_1 \neq c_2} \Delta(c_1, c_2) = \min_{c_1 \neq c_2} w(c_1 - c_2) = \min_{c \neq 0} w(c)$$

$\triangle$ Note, that the Walsh-Hadamard code is also a linear code, so we can prove its distance using this approach as well.

Definition 4.3 (Reed-Solomon code) *For a field* $\mathbb{F}$ *and numbers* $0 < n \leq m < |\mathbb{F}|$ *the **Reed-Solomon code** is a function* $\mathbb{F}^n \to \mathbb{F}^m$ *that takes* $n-1$ *degree polynomial* $A(x) \in \mathbb{F}[x]$ *and outputs its evaluation over* m *points* $f_0, ..., f_{m-1} \in \mathbb{F}$.

Lemma 4.2 *The Reed-Solomon code has distance of* $1 - \frac{n-1}{m}$

Proof. The word is a polynomial of degree less then n so it can have no more then $n-1$ roots. The codeword is the evaluation of this polynomial at $m > n$ distinct points, meaning it must contain at least $m-(n-1)$ non-zero values (since at most $n-1$ points can be the roots of our word polynomial). Since the Reed-Solomon code is a linear code, we can use its property to define its distance as $\delta = \frac{m-n+1}{n} = 1 - \frac{n-1}{m}$. $\triangle$

By following the "Unisolvence theorem" we know that to recover $n-1$ degree polynomial we need at least n points for interpolation. Because the Reed-Solomon code performs polynomial evaluation over $m > n$ points we can still interpolate (or decode) the polynomial even if some points are corrupted during data transmission.

Theorem 4.3 *There exists a polynomial-time (depending on the codeword size) algorithm with the following properties:*

- `Input`*: a list of pairs* $(a_i, b_i)\big|_0^{m-1}$ *of elements in* $\mathbb{F}$ *such that for* $\hat{t} > \frac{m}{2} + \frac{n}{2}$ *of them* $G(a_i) = b_i$ *for some unique polynomial* $G(x)$ *with* $\deg G(x) = n - 1$.
- `Output`*: a polynomial* $G(x)$ *with* $\deg G(x) = n - 1$.

Proof. The assumption $\hat{t} > \frac{m}{2} + \frac{n}{2}$ can be transformed into the corresponding constraint on the number of possible errors t:

$$t = m - \hat{t} < m - \frac{m}{2} - \frac{n}{2} = \frac{m}{2} - \frac{n}{2}$$

Let's also evaluate a necessary constraint that will be useful in the algorithm below:

$$t < \frac{m}{2} - \frac{n}{2}$$
$$2t < m - n$$
$$m > 2t + n$$

So, we know the lower bound of the codeword size depending on the word and number of errors: $m \geq 2t + n + 1$.

Algorithm 1 Berlekamp-Welch decoding [BW86]

So, $G(a_i) = b_i$ for at least $\hat{t}$ of m pairs. We also know that $\deg G(x) = n - 1$ and $t < m - n$. It means that we already can recover polynomial but we firstly have to deal with an errors.

Let's put an **error polynomial** $E(X)$ a polynomial which has roots at the error points.

So, $\deg E(x) = t < \frac{m}{2} - \frac{n}{2}$.

Our algorithm is based on the following equation:

$$C(a_i) = G(a_i) \cdot E(a_i)\big|_{i=0}^{m-1}$$

where $C(x)$ is an arbitrary polynomial that we are going to find.

Note, that $\deg C(x) = \deg G(x) + \deg E(x) = n - 1 + t$

Then, by solving the equation $C(a_i) = b_i \cdot E(a_i)\big|_{i=0}^{m-1}$ we can find $C(x)$ and $E(x)$.

This can be considered as a set of m linear equations where we have unknown $n + t$ coefficients of $C(x)$ and $t + 1$ coefficient of $E(x)$, so it can be solved via linear algebra while $m \geq 2t + n + 1$.

Finally, we put $G(x) = \frac{C(x)}{E(x)}$

The last equation follows from the observation that polynomial $C(x) - G(x) \cdot E(x)$ is zero in $\hat{t}$ points while it has degree $n + t - 1$. It is easy to prove that the polynomial degree less then the number of it's roots, so $C(x) - G(x) \cdot E(x)$ is zero for each x. That is why from $C(x) = E(x)G(x)$ follows that $C(x)$ is divisible on $E(x)$. $\triangle$

4.0.1 Reed-Solomon(255,223) implementation

The example implementation on SageMath aims to help reader to understand how the encoding and decoding look like. The Reed-Solomon code with $m = 255, n = 223$ is widely used in communication and storage systems, including QR codes, CDs, DVDs, and deep-space communication. It operates over $\mathbb{GF}(256)$ (the finite field of 256 elements) and encodes data as polynomials evaluated at different field elements. Following our theorem, the number of errors is constrained as $t < \frac{m}{2} - \frac{n}{2}$ which means that $t < 16$. For our implementation we take $t = 15$.

Let's start from the basic parameters definition:

```
F = GF(256)
R.<x> = PolynomialRing(F) # define x as a GF(256) element
m = 255 # A codeword size
n = 223 # A word size
t = 15 # The number of possible errors
distance = (m - n + 1)/m
```

The `distance` then is equal to $\frac{11}{85}$. Let's sample a random message that will be an information polynomial of degree $n = 223$ (this messsage can be obtained by the interpolation of some useful data):

```
word = sum(F.random_element() * x^i for i in range(n))
```

We start encoding from defining the evaluation elements that will be all non-zero elements of $\mathbb{GF}(256)$. Then, our codeword will be the $m = 255$ evaluations of our information polynomial.

```
# All non zero elements in F. Order(F) - 1 = 255
f = [f_i for f_i in F if f_i != 0]
assert len(f) == m
codeword = [word(f_i) for f_i in f]
assert len(codeword) == m
```

To decode the codeword we should solve the system of m equations $C(x_i) - b_i \cdot E(x_i) = 0$. Note, that this system is homogeneous, so the trivial solution (all zeros) is always a solution. To find a non-zero solution we impose a normalization condition, such as setting the leading coefficient of $E(x)$ to 1. This turns our original equation into the $C(x_i) - b_i \cdot E(x_i) = b_i \cdot x_i^{deg_E}$.

```
deg_C = 237 # deg = n - 1 + t
deg_E = 15 # deg = t

M = Matrix(F, m, deg_C + deg_E + 1)
for i, (f_i, b_i) in enumerate(zip(f, codeword)):
    # Fill in coefficients for C(x)
    for j in range(deg_C + 1):
        M[i, j] = f_i^j
    # Fill in coefficients for E(x), multiplied by -b_i
    for j in range(deg_E):
        M[i, deg_C + 1 + j] = -b_i * (f_i^j)


rhs = vector(F, [b_i * (f_i^deg_E) for f_i, b_i in zip(f, codeword)])
# Solve the linear system over F
solution = M.solve_right(rhs)
assert len(solution) == deg_C + deg_E + 1
```

Finally, we can recover the original information polynomial $G(x) = \frac{C(x)}{E(x)}$. We also should not forget to add the normalized leading coefficient of $E(x)$.

```
C = sum(solution[i] * x^i for i in range(deg_C + 1))
E = sum(solution[deg_C + 1 + i] * x^i for i in range(deg_E))
E += x ^ deg_E
G = C/E
assert G.numerator() == word
```

To discrover the error correcting properties of the presented code you can add the following snippet after the codeword calculation. It changes first t elements of codeword to the random $\mathbb{GF}(256)$ elements. If you try to change more elements then the linear system solving procedure will fail.

```
for i in range(t):
    codeword[i] = F.random_element()
```

5 Convolutional codes

A convolutional code is an ECC that utilizes the following properties:

- For each k input bits it generates $n > k$ output bits;
- The transformation also depends on the m previous bits;
- The ECC function is linear: for two inputs x, y, corresponding ECC's outputs X, Y and scalars a, b the following holds:
$$ax + by = aX + bY$$

For example, the **linear-feedback shift register** (LFSR) ECC [Gol67] for each k input bits provides n output bits while remembering m previous bits to use during the encoding process.

Definition 5.1 (Convolutional rate) *We call a **code rate** as the ratio between the input and output sizes:*
$$R = \frac{k}{n}$$

For example a rate $\frac{1}{3}$ means that for each input bit the code will generate 3 output bits. Usually, LFSR operates with rates $\frac{1}{2}$ or $\frac{1}{3}$. Each LFSR code defines the output function per each output bit that defines how it will be calculated. Usually, this function uses `XOR` over the pre-defined positions of the previously read values to generate the output (initially they are all zeros). So, during each iteration of the LFSR, it reads the next bit (or k bits) of the input (which is equivalent to shifting the whole sequence to the right, which allows to add incoming bits from the left), updates the state of the already read bits (utilizing the first-in-first-out over the queue of size m) and calculates n output bits.

To decode the output, convolutional codes utilize different algorithms (for example the Sequential Decoding [Woz57] or Viterbi algorithm [Vit67]), including, but not limited to, heuristic and probabilistic algorithms. All of them differ in computational complexity and correctness of the result.

References

[Woz57] John M. Wozencraft. *Sequential Decoding for Reliable Communication.* Tech. rep. Technical Report 325. Research Laboratory of Electronics, MIT, 1957.

[Gol67] Solomon W. Golomb. *Shift Register Sequences.* Holden-Day, 1967.

[Vit67] Andrew J. Viterbi. "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm". In: *IEEE Transactions on Information Theory* 13.2 (1967), pp. 260–269.

[MS77] F. J. MacWilliams and N. J. A. Sloane. *The Theory of Error-Correcting Codes.* North-Holland Publishing Company, 1977.

[BW86] Elwyn R. Berlekamp and Lloyd R. Welch. "Error Correction for Algebraic Block Codes". Patent US 4,633,470. Dec. 1986.

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach.* Cambridge University Press, 2009. ISBN: 978-0-521-42426-4. URL: `https://theory.cs.princeton.edu/complexity/book.pdf`.