Finite Fields and Error-Correcting Codes

Xavier Servot *

April 2019

1 Introduction

1.1 Motivations

We will assume here that the reader is familiar with basic channel coding and information theory, a document about it can be found at `xavierservot.com`

We we send a message, say a string of bits, this message might be altered during transmission. This is because it can travel through noisy channels, it can be stored on a corrupted storage system and so on and so forth.

In particular, the message can either be partially deleted like in: $01 \rightarrow ?1$ i.e. the 0 is deleted during transmission, or it can either be partially replaced like in: $01 \rightarrow 11$, where the 0 is replaced by a one. The former is called an *erasure channel* and the latter is called an *error channel*. In an erasure channel, we know at which location the message has been erased, i.e. 01 does not become 1 but ?1 where ? can be interpreted as undefined.

We can assign an *erasure weight* or an *error weight* noted p that measures the number of erasures or errors of a message.

So to deal with this, it looks like the decoder needs a way to retrieve what encoded message corresponds to an altered message, to then decode it. It might not be possible. For instance, in the case of a message of two bits 11. The receiver gets 01: even if he knows an error has occurred, he cannot know if the original was 00 or 11: both hypothesis are equally likely.

Nonetheless, the intuition of comparing the received message to the list of possible encoded messages is a good one. We will therefore need to define

*EPFL

some sort of *distance* between codewords to determine if two codewords are close or not.

In an error channel, and with the binary alphabet for codewords, we might for example consider the *hamming distance*, which computes in how many bits two codewords differ. You can see how such a measure can be useful in an error channel that flips a bit: retrieving the original codeword comes down to finding the codeword with hamming distance 1 from the received message.

1.2 In summary: the edge cases in decoding

With transmission in an error channel, we can expect three cases outside of the standard straightforward decoding.

- **Channel-error detection:** The decoder can detect an error, but is unable to correct it. In that case, it may ask for the transmitter to resend the code.
- **Channel-error correction:** The decoder can detect an error and correct it.
- **Channel-error decoding error:** It may decode the message wrong if it thinks it can correct, but it actually can't do it right.
- **Channel-error detection error:** A message could be wrong yet correspond to another valid codeword. For example, let's say $\mathcal{C} = \{00, 01\}$. If the transmitter sends 00 and in the channel the code becomes 01, it will be interpreted as valid by the decoder.

and for erasure channels, we know that if an error is present it is detected for sure. We have theses cases:

- **Channel-erasure correction:** The decoder can fill the erased position.
- **Channel-erasure detection:** It may ask for retransmission
- **Channel-erasure decoding error:** It fills blank positions with incorrect symbols.

1.3 Notations, definitions

1.3.1 Block codes

In this document, we only study *block codes*. We can note a block code by (n, k), which means the code is of constant length n, and the original messages are of length k.

In other words, each k source symbols is substituted by n channels symbols. This is done over the same alphabet.

1.3.2 Terminology

- $\mathcal{C}$ is the set of codewords for a given coding map.
- $\mathcal{A}$ is the alphabet
- A codeword takes its values in $\mathcal{A}^n$, and can be noted $(c_1, \ldots, c_n)$
- Information theory tells us that each codeword carries $k = \log_D(|\mathcal{C}|)$ information, for a D-ary alphabet.
- The *rate* is k/n.

1.3.3 Distance

Let X be a set. A *distance* is an application

$$d\colon X \times X \to [0; +\infty)$$

such that $\forall x, y, z \in X$

- $d(x, z) \leq d(x, y) + d(y, z)$ (Triangular inequality)
- $d(x, y) = d(y, x)$ (Symmetry)
- $d(x, x) = 0$ (Separation i)
- $d(x, y) = 0 \Rightarrow x = y$ (Separation ii)

A *metric space* is a set that has a distance. In our case, the metric space we are working on is $\{0, 1\}^n$, and we are going to focus on a distance called

Hamming distance. Let $x = (x_1, \ldots, x_n), y = (y_1, \ldots, y_n) \in \{0,1\}^n$It can be formally defined as

$$d(x,y) = \sum_{i=1}^{n} |x_i - y_i|$$

The three last properties of a distance are easily proven and we only need to prove the triangular inequality to prove that the Hamming Distance is actually a distance:

$$\begin{aligned} d(x,z) &= \sum_{i=1}^{n} |x_i - z_i| \\ &= \sum_{i=1}^{n} |x_i - y_i + y_i - z_i| \\ &\leq \sum_{i=1}^{n} |x_i - y_i| + |y_i - z_i| \\ &= d(x,y) + d(y,z) \end{aligned}$$

1.4 The role of distance in decoding

1.4.1 Minimum distance decoder

The setup is composed of an encoder, an error channel and a decoder. There is a special class of decoder called *minimum distance decoders*, and this is how they proceed:

i. If the channel code received corresponds to an existing code, it can decode and it's done. Otherwise, it proceeds to the following steps

ii. The channel decides to interpret the code c as the code it is the closest with in the set of possible codes, i.e.

$$\overline{c} = \arg\min_{c_p \in \mathcal{C}} d(c, c_p)$$

is what it chooses to interpret as the right code.

1.4.2 Error detection and $d_{\min}$

Definition 1.1 (Minimum-distance)**.** Let $\mathcal{C}$ be the set of codewords. We define the minimum distance as

$$d_{\min} = \min_{x,y \in \mathcal{C}, x \neq y} d(x, y)$$

using these notations, we can express our first interesting theorem, which has a pretty trivial proof.

Theorem 1.1 (Error-channel detection)**.** In an error channel, if a channel code has error weights p if

i. $p < d_{\min}$, then the error can be detected by a minimum distance decoder.

ii. $p \geq d_{\min}$, then some codewords can be transformed into other existing codewords and the error might not be detected by the minimum distance decoder.

Theorem 1.2 (Erasure-channel correction)**.** In an erasure channel, $p < d_{\min}$ if and only if the blank positions can be filled by the minimum-distance decoder for all erased codes

Proof. Let's say we have $p < d_{\min}$. Let's say we have c as the transmitted code and x as the partially erased code. In particular, $d(x, c) = p$. Ideally, we'd want our decoder to interpret c as x. For that matter, if $\exists y \in \mathcal{C} \setminus \{x\}$ such that c can be filled to y.

Therefore, we know that these two codes x, y differ at at most p positions. Thus $d(x, y) \leq p < d_{\min}$, which is absurd, because it is impossible unless $x = y$.

Conversely, let's say we can fill all the blank positions. By contradiction, let's say $p \geq d_{\min}$. Then let x, y such that $d(x, y) = d_{\min}$. Then, if we erase $d_{\min}$ components in c, the output of the channel code x, we already cannot know if c corresponds to x or y, and we still have erasures left. □

Theorem 1.3 (Error-channel correction)**.** In an error-channel, $p < \frac{d_{\min}}{2}$ if and only if the minimum distance decoder can correctly decode.

Proof. Let's say we have $p < \frac{d_{\min}}{2}$. Let c be the transmitted code and x the code with errors. We want to prove that the guess of the minimum distance decoder is c. Let c' be the guess. Then we have $d(x, c') \leq p$ and

$$\begin{aligned} d(c, c') &\leq d(c, x) + d(x, c') \\ &\leq 2p \\ &= d_{\min} \end{aligned}$$

i.e. $c = c'$

Conversely, we can consider only the case $d_{\min} > p \geq \frac{d_{\min}}{2}$ as $p = d_{\min}$ implies a channel code output overlap by construction. Let's say the minimum distance decoder can correctly decode.

Let c, c' such that $d(c, c') = d_{\min}$. Let y such that of the $d_{\min}$ positions where c and c' differ, y takes the first p from c and the rest of them from c'. That means

$$d(c, y) = p$$

and

$$d(c', y) = d_{\min} - p \geq 2p - p = p$$

i.e. c' is at least as close as c to y. □

1.5 Overcoming the computational hurdle of $d_{\min}$

1.5.1 Motivation

With a brute force algorithm, $d_{\min}$ could be computed in $\frac{1}{2}|\mathcal{C}|(|\mathcal{C}| - 1)$ i.e. $O(|\mathcal{C}|^2)$.

Now the codes used in a CD has around 10^{300} codewords, which should render the computation of $d_{\min}$ unfeasable using a brut force algorithm.

Other methods should then be considered, like computing it analytically.

1.5.2 An upper bound for $d_{\min}$

Theorem 1.4 (Singleton's bound)**.** Let $\mathcal{C}$ be an (n, k) block code. Then

$$d_{\min} - 1 \leq n - k$$

Proof. We consider the mapping $\psi \colon \mathcal{C} \to \mathcal{A}^{n-d_{\min}+1}$ which reduces the dimension of the codewords. We know that the difference in dimension between

the domain and the co-domain is $d_{\min} - 1$, which means that by construction ψ is injective, otherwise it would suffice for a change of weight $d_{\min} - 1$ to lead to another codeword, which is absurd.

As ψ is injective, the cardinality of the domain cannot be greater than the cardinality of the co-domain i.e.

$$|\mathcal{C}| \leq |\mathcal{A}^{n-d_{\min}+1}| \Leftrightarrow k \leq n - d_{\min} + 1$$

□

Definition 1.2. If the code $\mathcal{C}$ is such that

$$d_{\min} - 1 = n - k$$

then it is called a *maximum distance separable* code.

2 Finite fields

2.1 Motivations

Encoding, decoding and computing $d_{\min}$ become easier if the code is in a vector space, which are defined over finite fields.

2.2 Definition

Let $\mathbb{F}$ be a set and let's define the following mappings

$$+ : \begin{array}{l} \mathbb{F} \times \mathbb{F} \to \mathbb{F} \\ (a, b) \mapsto a + b \end{array}$$

$$\cdot : \begin{array}{l} \mathbb{F} \times \mathbb{F} \to \mathbb{F} \\ (a, b) \mapsto a \cdot b \end{array}$$

Let $\mathbb{F}_* = \mathbb{F} \setminus \{e_{(\mathbb{F},+)}\}$. $\mathbb{F}$ is a *ring* if $(\mathbb{F}, +)$ is an abelian group and $(\mathbb{F}_*, \cdot)$ is a semi-group, and if $\cdot$ is distributive over $+$.

$\mathbb{F}$ is a *commutative ring* if $(\mathbb{F}, +)$ is an abelian group and $(\mathbb{F}_*, \cdot)$ is a commutative semi-group, and if $\cdot$ is distributive over $+$.

A ring $\mathbb{F}$ has *unity* if the semi-group $(\mathbb{F}_*, \cdot)$ has an identity.

$\mathbb{F}$ is a *field* if $(\mathbb{F}, +)$ and $(\mathbb{F}_*, \cdot)$ are abelian groups, and if $\cdot$ is distributive over $+$.

We are going to call the mapping $+$ and addition and the mapping $\cdot$ a multiplication. We are going to note 0 for the additive identity $e_{(\mathbb{F},+)}$ and 1 for the multiplicative identity $e_{(\mathbb{F},\cdot)}$.

Now the reason that we don't require $(\mathbb{F},\cdot)$ to be an abelian group but instead $(\mathbb{F}_*,\cdot)$, if we do so we have:

$$0 \cdot a = 1 \Leftrightarrow (1 + (-1)) \cdot a = 1 \Leftrightarrow a - a = 1 \Leftrightarrow 0 = 1$$

which is a property we don't really want for our fields for about the same reasons why $1_{\mathbb{N}}$ is not a prime number: it would make the fundamental theorem of arithmetic false. So think of $0_{\mathbb{F}}$ not being $1_{\mathbb{F}}$ as a convenient property that makes the thinking process more elegant.

Definition 2.1. For a ring $\mathbb{F}$, $a, b \in \mathbb{F} \setminus \{0\}$ are *zero divisors* if $a \cdot b = 0$.

Proposition 2.1. Let $\mathbb{F}$ be a field. If $a, b \in \mathbb{F}$ and $ab = 0$, then $a = 0$ or $b = 0$.

Proof. Otherwise, we know that ab would not be inversible, while a and b have inverses thus:

$$ab \cdot b^{-1}a^{-1} = 1$$

which is absurd. □

2.3 The finite field $\mathbb{Z}/n\mathbb{Z}$

Now we are going to be using a particular ring, which we are going to define here. Let $n \in \mathbb{N}_*$, two elements are *congruent* to each other modulo n if their difference is divisible by n. $\forall a \in \mathbb{N}$, let the *congruence class* of a be

$$[a] = \{b \in \mathbb{N} \colon n | a - b\}$$

Let $[\mathbb{N}] := \{[x] \colon x \in \mathbb{N}\}$. Now what is interesting about this definition is that we can define the operations $+$ and $\cdot$ ont the set of congruence classes as:

$$+ \colon \begin{array}{l} [\mathbb{N}] \times [\mathbb{N}] \to [\mathbb{N}] \\ ([a],[b]) \to [a+b] \end{array}$$

And

$$\cdot \colon \begin{array}{l} [\mathbb{N}] \times [\mathbb{N}] \to [\mathbb{N}] \\ ([a],[b]) \to [a \cdot b] \end{array}$$

Which just means

$$[a] + [b] = [a + b] \qquad\qquad [a] \cdot [b] = [a \cdot b]$$

In particular, if $a \neq c$ and $[a] = [c]$, we have $[c] + [b] = [c + b] = [a + b]$, which can be verified easily, and the same goes for $\cdot$.

We are going to note the set of congruent classes modulo n to be

$$\mathbb{Z}/n\mathbb{Z} = \{[0], [1], \ldots, [n-1]\}$$

Where $(\mathbb{Z}/n\mathbb{Z}, +, \cdot)$ is a commutative ring with multiplicative unity [1]. We are going to be interested in the cases where $(\mathbb{Z}/n\mathbb{Z}, +, \cdot)$ is a field, which requires that every non-zero element has a multiplicative inverse. We are going to see for which case $\mathbb{Z}/n\mathbb{Z}$ is a field via Euclide's algorithm.

2.4 Euclide's algorithm

We are going to consider the *greatest common divisor* of two numbers a and b, noted $\gcd(a, b)$ or $(a; b)$ or $a \wedge b$. These properties will be useful in this section:

- Let $d := \gcd(a, b)$. Then $\exists k_a, k_b \in \mathbb{N}$ such that $a = dk_a, b = dk_b$. Thus

  $$a + b = d(k_a, k_b) \Rightarrow \gcd(a, b) | \gcd(a, a + b)$$

  Conversely, let $d' := \gcd(a, a+b)$. Then $\exists k_a, k_{a+b}$ such that $a = d'k_a, a + b = d'k_{a+b}$. Thus

  $$b = d'(k_{a+b} - k_a) \Rightarrow \gcd(a, a + b) | \gcd(a, b)$$

  Thus $\gcd(a, b) = \gcd(a, a + b)$

- More generally, by the same argument,

  $$\gcd(a, b) | \gcd(ka + lb, k'a + l'b)$$

  and if $d = \gcd(ka + lb, k'a + l'b)$, we have in particular that

  $d|l'(ka + lb) - l(k'a + l'b) = (l'k - lk')a \quad d|k'(ka + lb) - k(k'a + l'b) = (k'l - kl')b$

  Thus

  $$d | \gcd((l'k - lk')a, (-l'k + lk')b) = |l'k - lk'| \gcd(a, b)$$

  And in general,

  $$|l'k - lk'| = 1 \Rightarrow \gcd(a, b) = \gcd(ka + lb, k'a + l'b)$$

We can now start constructing the algorithm, which uses the fact that if the euclidean division of a by b gives $a = bk + r$, we have by induction:

$$\gcd(a, b) = \gcd(a - b, b) = \ldots = \gcd(a - bk, b) = \gcd(r, b)$$

By convention, in $\mathbb{Z}$, 0 is divisible by every integer, and thus

$$\gcd(a, 0) = a$$

So now intuitively, starting from $\gcd(a, b) = \gcd(b, r) = \gcd(rk' + r', r) = \gcd(r, r') = \ldots$, if we keep doing this till we get a null rest, we get that the rest just before the null one is the gcd, as stated by the property above. Thus for example, if $r' = 0$ we'd have $\gcd(a, b) = \gcd(r, r') = \gcd(r, 0) = r$.

Now we are also sure that we are eventually going to get a null rest: for $a >= b \in \mathbb{N}$, $a = bk + r$ with $r < b$ and $\gcd(a, b) = \gcd(b, r)$, and this argument can be applied at each step thus we obtain a strictly decreasing series of numbers ($r < b$, and $a > b$ otherwise the gcd is trivial).

Now, knowing that, we can go further in this algorithm with the following theorem

Theorem 2.1 (Bézout)**.** Let $a, b \in \mathbb{N}$, and $d := \gcd(a, b)$. $\exists u, v \in \mathbb{Z}$ such that

$$au + bv = d$$

Proof. Intuitively, at each step we have an euclidean division of the form

$$r_{i-2} = r_{i-1}q_i + r_i$$

where we have $r_{-2} = a$ and $r_{-1} = b$ if r_0 designates the rest of the euclidean division of a by b. Thus right before the last step of the algorithm, let's stay step k, we have

$$r_{k-2} = r_{k-1}q_k + d \Leftrightarrow d = r_{k-2} - r_{k-1}q_k$$

And at the step $k - 1$ we have

$$r_{k-3} = r_{k-2}q_{k-1} + r_{k-1} \Leftrightarrow r_{k-1} = r_{k-3} - r_{k-2}q_{k-1}$$

Thus we can inject the equation of the step $k - 1$ into the equation of the step k to get an expression of d that only depends on the coefficients r_{k-2} and r_{k-3}.

Now it is easy to see that going all the way up to r_{-2} and r_{-1} we'd have an expression of d depending only on those two rests, in the sense that we'd have an equality of the form $au + bv = d$. □

Corollary 2.1.1. if a and b are coprime, i.e. $a \wedge b = 1$, iff $\exists u, v \in \mathbb{Z}$

$$au + bv = 1$$

Which can be reformulated as: let $[a] \in \mathbb{Z}/b\mathbb{Z}$, $[a]$ has an inverse iff a and b are coprime.

Corollary 2.1.2. If p is prime, then $\mathbb{Z}/p\mathbb{Z}$ is a field.

2.5 Characteristic

Definition 2.2. Let $\mathbb{F}$ be a field. The *characteristic* of $\mathbb{F}$ is defined as

$$\operatorname{Char} \mathbb{F} = \min_{k \in \mathbb{N}_*} \sum_{i=1}^{k} 1_{\mathbb{F}} = 0_{\mathbb{F}}$$

If it doesn't exist, we define it as $\operatorname{Char} \mathbb{F} = \infty$.

Theorem 2.2. Let $\mathbb{F}$ be a finite field. Its characteristic is a prime number

Proof. Let $c = \operatorname{Char} \mathbb{F}$. We have

$$\sum_{i=1}^{c} 1 = 0$$

Let $c = ab$ i.e. c is not prime, by contradiction. Then we'd have

$$\sum_{i=1}^{c} 1 = \sum_{i=1}^{ab} 1 = (\sum_{i=1}^{a} 1)(\sum_{j=1}^{b} 1) = 0$$

which means $\mathbb{F}$ would have zero divisors, which is not possible as it is a field. □

3 Linear codes

3.1 Introduction

A code is *linear* if the codewords are a subspace (a subset that is a vector space) of $\mathbb{F}^n$, for a finite field $\mathbb{F}$. This means that linearly combining codewords yields another codeword.

For example, the code $01, 00, 10$ is not a linear code over $\mathbb{F}_2^2$ because $01 + 10 = 11$ is not a codeword. However, $00, 01$ is a linear code over $\mathbb{F}_2^2$.

The dimension of the latter linear code is thus 1 because a base for the code would be $\{01\}$. In the case of $00, 01, 10, 11$, a base for the code would be $\{01, 10\}$ and the dimension would be 2.

More generally

Proposition 3.1. A k-dimensional subpsace of $\mathbb{F}_p^n$ has cardinality p^k

Proof. Let $\mathbb{G}$ be a subspace. Let a base of this subspace be $(v_1, \ldots, v_k)$. Then, every element of $\mathbb{G}$ is of form

$$\sum_{i=1}^{k} \alpha_i v_i$$

where $\alpha_i \in \mathbb{F}_p$. Thus the set of possible $(\alpha_i)_{i=1}^k$ is of cardinality p^k. □

Definition 3.1. Let $\mathbb{F}$ be a finite field and $x \in \mathbb{F}^n$. We define the *Hamming Weight* of x as

$$w(x) = d(0, x)$$

i.e. the number of components that are non-zero.

For example, $w(1, 0, 1) = 2$ and $w(2, 0, 5, 7) = 3$. And the following will make clear our motivations for using linear codes:

Theorem 3.1. Let $\mathcal{C}$ be a code.

$$d_{\min} = \min_{c \in \mathcal{C} \setminus \{0\}} w(c)$$

Proof. Let $x, y \in \mathcal{C}$. The first thing to consider is that $d(x, y) = w(x - y)$, where $x - y$ is a codeword by linearity. Thus:

$$d_{\min} = \min_{x,y \in \mathcal{C}, x \neq y} d(x, y) = \min_{x,y \in \mathcal{C}, x \neq y} w(x - y) \geq \min_{c \in \mathcal{C}, c \neq 0} w(c)$$

and

$$\min_{c \in \mathcal{C}, c \neq 0} w(c) = \min_{c \in \mathcal{C}, c \neq 0} d(c, 0) \geq d_{\min}$$

thus $d_{\min} = \min_{c \in \mathcal{C} \setminus \{0\}} w(c)$ □

the obvious gain with that property is that we can compute in a linear time $d_{\min}$.

3.2 Generator Matrix

3.2.1 Definition

Definition 3.2. Let $\mathcal{C}$ be a code. We define the *generator matrix* as the matrix of base vectors of $\mathcal{C}$.

For example, let's consider the code (0000000, 0011100, 0111011, 1110100, 0100111, 1101000, 1001111, 1010011) which has the base $(0011100, 0111011, 1110100)$. Its generator matrix is going to be

$$\begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 \end{bmatrix}$$

In this case, we know that any codeword can be represented by a vector in $\mathbb{F}_2^3$, as $\mathcal{C} = \text{span}(0011100, 0111011, 1110100)$.

3.2.2 The big picture: source coding and channel coding

We can integrate linear codes in a broader context:

$$\text{Source} \to \text{Source coding (2-ary code)} \to \text{Channel coding}$$

we have in this scheme that with channel coding, we code blocks of k symbols coming from the source code: each block of k binary symbols in the stream of the source code can give a channel code thanks to the generator matrix.

3.2.3 Systematic form

Definition 3.3. A generator matrix G_s of a code is in *systematic form* if it is in reduced echelon form, i.e. $\exists P_{k\times n-k} \in \mathcal{M}_{k\times n-k}(\mathbb{F})$ such that

$$G_s = (I_k, P_{k\times n-k})$$

To get a generatir matrix in systematic form, we need to

i. Find a base for our code

ii. Create a matrix with the base vectors

iii. Transform the matrix into its reduced-echelon form

We notice that for a given source code u, the channel code for u given a generator matrix G_s in systematic form is

$$uG_s = (uI_k, uP_{k\times n-k})$$

i.e. the first k symbols of the channel code are going to be the source code as-is.

The interpretation for $uP_{k\times n-k}$ is that it gives more information about the channel code uI_k. Given a channel code, we can interpret the first k symbols as u, the source code, and check if the following $n-k$ symbols are of form $uP_{k\times n-k}$, which ensures that the channel code is correct.

More specifically, when we say that the symbols are "of form" $uP_{k\times n-k}$, we have a specific way of doing so: by using a *parity-check matrix* $H \in \mathcal{M}_{n-k\times k}(\mathbb{F})$. In particular, this matrix has the the following property: let y be a channel code. Then $yH^\top = 0 \Leftarrow y$ is correct.

Theorem 3.2. Let $\mathcal{C}$ be a code. Let $G_s = (I_k, P)$ in systematic form. Then

$$H = (-P^\top, I_{n-k})$$

Proof. Let y be a channel code. If it is correct, it is of the form uG_s. Now,

$$uG_sH^\top = u(I_k, P)(-P^\top, I_{n-k})^\top = u(-P+P) = 0$$

□

Definition 3.4. Let y be a channel code. The *syndrome* of y is

$$s := yH^\top$$

By the previous theorem, $s = 0 \Leftarrow y$ is correct.

Theorem 3.3. Let H be the parity check matrix of a code. Then the minimum distance of this code is the smallest number of columns d of H that are linearly dependant.

Proof. Let $c \neq 0$ a channel code of weight t. $cH^\top = 0$ implies that H has t dependent columns.

Conversely, if H has t dependant columns, there exists $c \neq 0$ such that $cH^\top = 0$, thus there is a codeword of weight t.

This means that the matrix has t dependant columns iff. there exists a code of weight t. Now, the smallest weight possible is $d_{\min}$, which means that H has $d_{\min}$ dependant columns, and if H had a fewer amount of dependant columns, there would be a non-null codeword which weight is smaller than $d_{\min}$, which is absurd. □

3.3 Standard array

3.3.1 Introductory definitions

We will assume the reader is familiar with equivalence relations $\sim$ on a set $\mathcal{G}$. We will note the equivalence class of $a \in \mathcal{G}$ as $[a]$. Equivalence relations are symmetric, reflexive and transitive binary relations.

Now we will define a useful equivalence relation. Let $(G, \cdot)$ a group and $H \subset G$ a subgroup:

$$a \sim b \Leftrightarrow \exists h \in H \text{ such that } b = a \cdot h$$

Proof.

- **Symmetric:** let $a, b \in G$, such that $a \sim b$. Then $\exists h \in H$ such that $b = a \cdot h$. Then we also have $a = b \cdot h^{-1}$ and thus $b \sim a$.
- **Reflexive:** Let $a \in G$. As $e_G \in H$, we have $a = e_G \cdot a \Rightarrow a \sim a$.
- **Transitive:** Let $a, b, c \in G$ such that $a \sim b, b \sim c$. Thus $\exists h, h' \in H$ such that $a = b \cdot h, b = c \cdot h'$. Thus $a = (c \cdot h') \cdot h = c \cdot (h' \cdot h) \Rightarrow a \sim c$. □

Now, let $a \in G$, we have $[a] = \{b \in G \colon \exists h \in H \text{ such that } a = b \cdot h\} = \{a \cdot h \colon h \in H\}$. Thus we can note $[a] = a \cdot H$, which is called a *coset* of H with respect to a.

Proposition 3.2. $\forall a \in G, |a \cdot H| = |H|$

Proof. The proof is pretty straightforward: we know that the mapping $h \mapsto a \cdot h$ is bijective as $z \mapsto a^{-1} \cdot z$ is the inverse. Thus the domain H and the codomain $a \cdot H$ have the same cardinality. □

3.3.2 Standard array definition

Using the introductory definitions: what is interesting to us as a group is $(\mathbb{F}^n, +)$, with codes that are subset of $(\mathbb{F}^n, +)$, and where the equivalence relation is defined as $x \sim y \Leftarrow \exists c \in \mathcal{C}$ such that $y = x + c$.

Intuitively, each row of the standard array represents a coset of the equivalence relation we just defined:

$$\begin{bmatrix} c_0 = 0 & c_1 & \dots & c_{M-1} \\ t_1 & t_1 + c_1 & \dots & t_1 + c_{M-1} \\ \vdots & \vdots & \ddots & \vdots \\ t_{L-1} & t_{L-1} + c_1 & \dots & t_{L-1} + c_{M-1} \end{bmatrix}$$

thus the t_i are in $\mathbb{F}^n$, and $|\mathcal{C}| = M$. In particular, we chose the coset leaders t_i such that $\forall j \in \{0, \ldots, L-1\}$, $t_j \notin \cup_{i=1}^{j-1} t_i \cdot \mathcal{C}$

3.3.3 Decoding regions

What we want to achieve by defining the standard array, is to define *decoding regions* for the code, i.e. sets that partition $\mathbb{F}^n$:

$$\forall i, j, i \neq j, \mathcal{D}_i \bigcap \mathcal{D}_j = \varnothing \qquad \bigcup_{i=0}^{M-1} \mathcal{D}_i = \mathbb{F}^n$$

And thus we have an algorithm for decoding:

- let y be the output of a channel.
- We determine to which region y belongs, which is unique by construction. Say $y \in \mathcal{D}_r$.
- We interpret y as c_r.

Now in our case, we will define the decoding regions to be the columns of the standard array, i.e. $\mathcal{D}_i$ is the i-th column of the standard array:

$$\mathcal{D}_i = \{c_i, c_i + t_1, \ldots, c_i + t_{L-1}\}$$

We can verify that this construction implies

$$\forall i, j, i \neq j, \mathcal{D}_i \bigcap \mathcal{D}_j = \varnothing \qquad \bigcup_{i=0}^{M-1} \mathcal{D}_i = \mathbb{F}^n$$

3.4 Error probability

In the definition of the standard array, we chose our coset leaders at random, and with the following property:

$$\forall j \in \{0, \ldots, L-1\}, t_j \notin \cup_{i=1}^{j-1} t_i \cdot \mathcal{C}$$

We will see that there is a way to minimize the error probability of the code if we chose particular coset leaders. We consider the case where we have input and output alphabet $X = \{0, 1\}$. The output of the channel is the same as its input with probability ε, we it inverted the input with probability $1 - \varepsilon$.

An error pattern is an element of $\mathbb{F}^n$ that is added to the input code in $\mathbb{F}^n$ by the channel. Given an error pattern $e \in \mathbb{F}^n$, its probability is

$$\varepsilon^{w(e)}(1-\varepsilon)^{n-w(e)} = (\frac{\varepsilon}{1-\varepsilon})^{w(e)}(1-\varepsilon)^n$$

Now the probability of an error ε will in practice be inferior to $1/2$, in which case we can conclude analytically that the probability of the error pattern e is inversely proportional to $w(e)$.

To put all of this in situation: let $c_i \in \mathcal{C}$, where the output of the channel is $y = c_i + e$. We know that we will interpret y as c_i if $y \in \mathcal{D}_i$. We thus need to get the probability that $y \in \mathcal{D}_i$. Now, we know that as long as $e \in \mathcal{D}_0$, we will have $y = c_i + \mathcal{D}_0 \in \mathcal{D}_i$. The probability that $e \in \mathcal{D}_0$ is

$$P = \sum_{e \in \mathcal{D}_0} \varepsilon^{w(e)}(1-\varepsilon)^{n-w(e)} = \sum_{i=0}^{L-1} \varepsilon^{t_i}(1-\varepsilon)^{n-w(t_i)}$$

To minimize this probability, we need to minimize the weights of the coset leaders $(t_i)_{i=0}^{L-1}$, as the probability of each term is inversely proportional to $w(e)$. We can swap elements to do so: if there is j such that $w(t_j) < w(c_i + t_j)$ we can swap those terms. Thus we can imagine an algorithm in $O(n^2)$ that for each coset leader, find the element of its row that has the minimal weight and swap these two.

The resulting code can be decoded by a minimum-distance decoder. Indeed, let $y \in \mathcal{D}_i$. It is a minimum distance decoder if $d(c_i, y) \leq d(c_k, y), \forall k$. Now

$$d(c_k, y) = d(c_k, c_i + t_j) = w(t_j + c_i - c_k) = w(t_j + c_l)$$

where the last equality stems from the fact that the code is linear, thus $\exists c_l$ such that $c_i - c_k = c_l$. Now we have $d(c_k, y) = w(t_j)$, and by the following algorithm:

$$w(t_j) \leq w(t_j + c_l)$$

3.5 Decoding in practice

We now have a clear way of decoding the output of a channel: we find in which region it is and interpret it as the code associated with that region. In practice, that is impractical because the standard array is often too big to be stored, but as we saw $\mathcal{D}_i = c_i + \mathcal{D}_0$. It seems like $\mathcal{D}_0$, i.e. the first column can help us decode systematically.

Each element of a row has the same syndrome as the coset leader:

$$(t_j + c_i)H^\top = t_j H^\top + c_i H^\top = t_j H^\top$$

and two coset leaders don't have the same syndrom: by contradiction let $t, t', t \neq t'$ such that $tH^\top = t'H^\top$. Then $(t-t')H^\top = 0$ and thus $t - t' \in \mathcal{C} \Rightarrow t = t' + c$. Thus t and t' are in the same coset, which is absurd.

Let y be the output of a channel. The following algorithm is more efficient than looking up where y belongs in the standard array:

i. We precompute the syndromes of the coset leaders, and we store the syndromes with their coset leader in an array.

ii. We compute $yH^\top$, which identifies the row of y as we saw.

iii. We lookup the coset leader that corresponds to $yH^\top$, noted t.

iv. We interpet y as $c = y - t$.

v. We solve $uG = c$ for u. If G is in systematic form, it is trivial.

vi. We have decoded the information: it is u.

The first operation, i.e. precomputing the syndromes of the coset leaders, encapsulates more it seems. To find the coset leaders that minimize the error probability, we need to compute the standard array first. However once we have precomputed the syndromes, we don't need the standard array anymore.

While this approach looks great and promising, storing a lookup table of size $n - k$ is unpractical for most applications. We will have to resolve to a better approach.

4 Reed-Solomon codes

4.1 Motivations

Reed Solomon Codes were created in 1960 by Irving Reed and Gustave Solomon. The motivations behind use Reed Solomon codes where in part covered in the last section: we want an efficient decoding algorithm. We will see that Reed Solomon codes are still linear, they are maximum distance separable and have a nice construction.

However to construct them, we will first have to take a little bit of time on polynomials over fields. We will assume that you are somewhat familiar with polynomials, and will not take time on the basics.

4.2 Polynomials over fields

4.2.1 Definition

Let $\mathbb{F}[X]$ be set of functions $\mathbb{F} \to \mathbb{F}$ of form

$$f\colon X \mapsto \sum_{i=0}^{n} a_i X^i$$

for some $n \in \mathbb{N}$, where $a_n \neq 0$. The *degree* of $f \in \mathbb{F}[X]$ is n, and $\mathbb{F}[X]$ is the set of polynomials over $\mathbb{F}$.

Let $u = (u_i)_{i=1}^k \in \mathbb{F}^k$. We define $P_u \in \mathbb{F}[X]$ by:

$$P_u\colon \begin{array}{l} \mathbb{F} \to \mathbb{F} \\ X \mapsto u_1 + u_2 X + \ldots + u_k X^{k-1} \end{array}$$

here, the degree can be less than $k-1$: some components of u can be null.

4.2.2 Lagrange Interpolation

Given $(a_i)_{i=1}^k, (y_i)_{i=1}^k \in \mathbb{F}^k$, we want to find a polynomial f such that $f(a_i) = y_i, \forall i \in \{1, \ldots, k\}$. We will first construct polynomials Q_j such that $Q_j(a_j) = y_j, Q_j(a_i) = 0, j \neq i$.

$$Q_j(x) = y_j \prod_{i=1, i\neq j}^{k} (x - a_i)(a_j - a_i)^{-1} = y_j \prod_{i=1, i\neq j}^{k} \frac{x - a_i}{a_j - a_i}$$

We verify that we have the wanted property:

$$Q_j(a_j) = y_j \prod_{i=1, i\neq j}^{k} \frac{a_j - a_i}{a_j - a_i} = y_j \prod_{i=1, i\neq j}^{k} 1 = y_i$$

and for $l \neq j$:

$$Q_j(a_l) = y_i(\prod_{i=1, i\neq j,l}^{k} \frac{a_l - a_i}{a_j - a_i})(\frac{a_l - a_l}{a_j - a_i}) = 0$$

and we get that $f = \sum_{j=1}^{k} Q_j$ satisfies $f(a_i) = y_i$.

4.2.3 Roots

Proposition 4.1 (Fundamental theorem of algebra)**.** A polynomial in $\mathbb{F}[X] \setminus \{0\}$ which degree is at most k has at most k distinct roots.

Proof. We will prove it by showing that the only polynomial of degree at most $k-1$ which has k distinct roots is 0. Let $(a_i)_{i=1}^k \in \mathbb{F}^k$. We consider the mappings

$$\psi : \begin{array}{l} \mathbb{F}^k \to \mathbb{F}^k \\ u \mapsto (P_u(a_i), \ldots, P_u(a_k)) \end{array}$$

By Lagrange interpolation, we know that given $(y_i)_{i=1}^k$, we can find u such that $\psi(u) = (y_i)_{i=1}^k$. Thus ψ is surjective. Now, the domain and co-domain of ψ are of same cardinality, thus by the pigeonhole principle, the mapping is injective, and bijective.

Now, $\psi(0) = (0)_{\{1,\ldots,k\}}$, and the mapping is bijective thus the polynomial $P_0 = 0$ is the only polynomial of degree at most $k-1$ which has k roots. □

4.3 Reed-Solomon code construction

We choose $q \geq 2$ and place ourselves in the field $\mathbb{F}_q$. We choose $n \in \{0, \ldots, q\}$ and take n distinct elements $(a_i)_{i=1}^n \in \mathbb{F}_q^n$. We can take such distinct elements since $n \leq \operatorname{Card} F_q$. We define the mapping

$$\phi : \begin{array}{l} \mathbb{F}_q^k \to F_q^n \\ u \mapsto (P_u(a_i))_{i=1}^n \end{array}$$

which is a block code of length n over $\mathbb{F}_q$.

So for example, let's choose $q = 3, n = 3, (a_1, a_2, a_3) = (2, 1, 0), k = 2$. We'd then have the mapping

$$\phi\colon \begin{array}{l} \mathbb{F}_3^2 \to F_3^3 \\ u \mapsto (P_u(a_i))_{i=1}^3 \end{array}$$

and the possible codewords are given by $\operatorname{Im}\phi$. $\mathbb{F}_3^2 = \{00, 10, 20, 01, 11, 21, 02, 12, 22\} := \{u_1, \ldots, u_9)\}$, and

$$\begin{aligned} P_{u_1}(X) &= 0 & P_{u_2}(X) &= 1 \\ P_{u_3}(X) &= 2 & P_{u_4}(X) &= X \\ P_{u_5}(X) &= 1 + X & P_{u_6}(X) &= 2 + X \\ P_{u_7}(X) &= 2X & P_{u_8}(X) &= 1 + 2X \\ P_{u_9}(X) &= 2 + 2X \end{aligned}$$

Thus the codewords are given by

$$\begin{aligned} c_1 &= (0, 0, 0) & c_2 &= (1, 1, 1) \\ c_3 &= (2, 2, 2) & c_4 &= (2, 1, 0) \\ c_5 &= (0, 2, 1) & c_6 &= (1, 0, 2) \\ c_7 &= (1, 2, 0) & c_8 &= (2, 0, 1) \\ c_9 &= (0, 1, 2) \end{aligned}$$

4.4 Properties of Reed-Solomon codes

Proposition 4.2. Reed-Solomon codes are linear.

Proof. Let $u, v \in \mathbb{F}_q^k$. We want to show that the linear combination of two codewords gives another codeword, i.e. that $P_u(a_i) + P_v(a_i) = P_w(a_i)$ for some $w \in \mathbb{F}_q^k, a_i \in F_q$. Now,

$$P_u(X) = \sum_{i=0}^{k-1} u_i X^i \qquad P_v(X) = \sum_{i=0}^{k-1} v_i X^i$$

thus for $\lambda \in \mathbb{F}_q$

$$P_u(X) + \lambda P_v(X) = \sum_{i=0}^{k-1} (u_i + \lambda v_i) X^i = P_{u+\lambda v}(X)$$

Now, $u + \lambda v \in \mathbb{F}_q^k$. Thus we have the wanted property: combining linearly components of a code yields a component of same index of another code. Thus the code is linear. □

Proposition 4.3. Reed-Solomon codes have dimension k

Proof. The polynomial P_u is of degree k, and we know that the mapping

$$\psi : \begin{array}{l} \mathbb{F}^k \to \mathbb{F}^k \\ u \mapsto (P_u(a_i), \ldots, P_u(a_k)) \end{array}$$

is bijective. Now in our case we are interested in the coding map:

$$\phi : \begin{array}{l} \mathbb{F}^k \to \mathbb{F}^n \\ u \mapsto (P_u(a_i), \ldots, P_u(a_n)) \end{array}$$

which is garanteed to be injective by the bijectivity of the previous map. This means that $\operatorname{Card} \operatorname{Im} \phi = |\mathbb{F}|^k$, and the dimension is thus k. □

Proposition 4.4. Reed-Solomon codes are maximum distance separable.

Proof. We know that the polynomial used to generate the code has at most $k-1$ roots since $u \in \mathbb{F}^k$. This means that $c := (P_u(a_1), \ldots, P_u(a_n))$ has at most $k-1$ zeros. Or equivalently, $w(c) \geq n - k + 1$. Now Reed-Solomon codes are linear thus

$$d_{\min} = \min_{c \in \mathcal{C}} w(c) \geq n - k + 1$$

And the Singleton's bound is still true thus

$$d_{\min} \leq n - k + 1$$

Which implies that $d_{\min} = n - k + 1$. □

Proposition 4.5. Reed-Solomon codes are linearly independant

Proof. The encoding map ϕ is injective thus the codomain is linearly independant if and only if the domain is. □

4.5 Generator matrix

We know that the code is linear. thus we can construct the generator matrix of form:

$$G = \begin{bmatrix} \phi(e_1) \\ \vdots \\ \phi(e_k) \end{bmatrix}$$

Now $P_u(X) = \sum_{i=1}^{k} u_i X^{i-1}$, thus $P_{e_k}(a_j) = a_j^{k-1}$ and

$$G = \begin{bmatrix} 1 & 1 & \dots & 1 \\ a_1 & a_2 & \dots & a_n \\ \vdots & \vdots & \ddots & \vdots \\ a_1^{k-1} & a_2^{k-1} & \dots & a_n^{k-1} \end{bmatrix}$$

which is the transpose of a Vandermonde matrix.