وزارة التعليم العالي والبحث العلمي

جامعة ديالى

كلية التربية المقداد

قسم الرياضيات

تـــشـفـيـر الرياضـيـــات

بحث مقدم الى كلية التربية المقداد / قسم الرياضيات

وهو جزء من متطلبات نيل شهادة البكالوريوس / رياضيات

مقدم من قبل الطالبان

عساف عباس علوان

مها جاسم كاظم

بأشراف

م.د خالد هادي

2022

اقرار المـشـرف

اشهد بان اعداد هذا لمشروع الموسوم

---------- تشــفــير الرياضــيــات ----------

والمعد من قبل لطالبان ----- عساف عباس علوان و مها جاسم كاظم --------

قد تم بأشرافي في قسم الرياضيات / كلية التربية المقداد / جامعة ديالى

وهو جزء من متطلبات نيل شهادة البكالوريوس / رياضيات

التوقيع :

اسم المشرف :

المرتبة العلمية :

التاريخ :

الاية الكريمة

بسم الله الرحمن الرحيم

(ٱللَّهُ نَزَّلَ أَحۡسَنَ ٱلۡحَدِيثِ كِتَٰبا مُّتَشَٰبِها مَّثَانِيَ تَقۡشَعِرُّ مِنۡهُ جُلُودُ ٱلَّذِينَ يَخۡشَوۡنَ رَبَّهُمۡ ثُمَّ تَلِينُ جُلُودُهُمۡ وَقُلُوبُهُمۡ إِلَىٰ ذِكۡرِ ٱللَّهِۚ ذَٰلِكَ هُدَى ٱللَّهِ يَهۡدِي بِهِۦ مَن يَشَآءُۚ وَمَن يُضۡلِلِ ٱللَّهُ فَمَا لَهُۥ مِنۡ هَادٍ)

صدق الله العظيم (الزمر ٢٣)

الاهـــــــــــداء

إلى من يسعد قلبي بلقياها

إلى روضة الحب التي تنبت أزكى الأزهار

امي

إلى رمز الرجولة والتضحية

إلى من دفعني إلى العلم وبه ازداد افتخار

ابي

إلى من هم اقرب ألي من روحي

إلى من شاركني حضن ألام وبهم استمد عزتي وإصراري

اخوتي

إلى من آنسني في دراستي وشاركني همومي

تذكاراً وتقديراً

أصدقائي

إلى هذه الصرح العلمي الفتي والجبار

كلية التربية المقداد

شكر وتقدير

لابد لنا ونحن نخطو خطواتنا الأخيرة في الحياة الجامعية من وقفه نعود إلى أعوام قضيناها في رحاب الجامعة

مع أساتذتنا الكرام الذين قدموا لنا الكثير باذلين بذلك جهودا كبيرة في بناء جيل الغد لنبعث الأمة من جديد .....

وقبل أن نمضي نقدم أسمى آيات الشكر والامتنان والتقدير والمحبة إلى الذين حملوا أقدس رسالة في الحياة

إلى الذين مهدوا لنا طريق العلم والمعرفة

إلى جميع أساتذتنا الأفاضل

" كن عالما . فإن لم تستطع فكن متعلما ، فإن لم تستطع فأحب العلماء ، فإن لم تستطع فلا تبغضهم"

ونخص بالشكر والتقدير لراعي الرياضيات ومربيها في كلية التربية المقداد صاحب القلب الطيب الذي كانت له قدم رائدة في طريق التعليم والعلم،

الدكتور خالد هادي حميد

الذي نقول له بشرك قول رسول الله صلى الله عليه وسلم

" إن الحوت في البحر ، والطير في السماء ليصلون على معلم الناس الخير"

وكذلك نشكر كل من ساعدنا على إتمام هذا البحث وقدم لنا العون ومد لنا يد المساعدة وزودنا بالمعلومات اللازمة لإتمام هذا البحث

الخلاصة

في هذا البحث تم دراسة احد المفاهيم الاساسية والرئيسية في الرياضيات التطبيقية ، وهو مصطلح التشفير .

حيث تم تحويل مجموعة من الاعداد الحقيقية الموجبة الى نظام الثنائي حيث يأخذ القيم اما 0 او 1

إضافة الى ذلك تم اعطاء خصائص لهذا النظام من حيث تعريف عمليتي الجمع والضرب على الاعداد الثنائية .

وفي الختام تم تصميم نموذج $\mathbb{F}_2^1$ و $\mathbb{F}_2^2$ و $\mathbb{F}_2^3$ و $\mathbb{F}_2^4$

CONTENTS

Introduction

A mathematical problem originating in electrical engineering is the recovery of a signal which is transmitted over a noisy medium. Examples include electrical signals traveling down a wire (e.g. data networking), radio signals traveling through free space (e.g. cellular phones or space probes), magnetization domains on a hard disk, or pits on an optical disk. In the latter cases, the issue is storage rather than transmission; for brevity we will use the term transmission nonetheless.

Abstractly, let ∑ be a finite set, or **alphabet**, of symbols. Often, but not necessarily, ∑= {0,1}, in which case symbols are called **bits**. ∑ * be the (infinite) set of all strings, or **messages**, of zero or more symbols over ∑. Let *M* be an element of ∑* , transmitted over some medium. Due to physical phenomena occurring during transmission, the transmitted string *M* may differ from the received string *M'*. For example, let ∑ be the lower-case letters along with the space character. Errors include, but are not limited to, the following: insertion or duplication of symbols (e.g. "the house" is received as "the houuse"), deletion of symbols (e.g. "the huse"), and/or modified symbols (e.g. "the hopse"). When we type, a common error is transposition ("teh house").

The essential idea is that protection against errors is accomplished by adding additional symbols to *M* in such a way that the redundant information may be used to detect and/or correct the errors in *M'*. This insertion of redundant information is called **coding**. The term **error control** encompasses error detection and error correction. We will be discussing so-called **block codes**, in which a message is divided into blocks of *k* symbols at a time. The transmitter will **encode** by adding additional symbols to each block of ***k*** symbols to form a transmitted block of *n* symbols. The receiver will **decode** by transforming each received block of *n* symbols back into a ***k***-symbol block, making a best estimate which ***k***-symbol block to decode to. (Note that you can mentally fix each misspelling of the house" above, without needing redundant information. You do this (a) by context, i.e. those weren't just random letters, and (b) by using your intelligence. Automated error-control systems typically have neither context nor intelligence, and so require redundancy in order to perform their task.)

Encoding circuitry is typically simple. It is the decoding circuity which is more complicated and hence more expensive in terms of execution time, number of transistors on a chip, power consumption (which translates into battery life), etc.

For this reason, in situations with low error rate it is common for a receiver to detect errors without any attempt at correcting them, then have the sender re-transmit. It is also for this reason that much of the effort in coding-theory research involves finding better codes, and more efficient decoding algorithms.

Different media require different amounts of redundancy. For example, communications between a motherboard and a mouse or keyboard are sufficiently reliable that they typically have no error detection at all. The higher-speed USB protocol uses short cyclic redundancy checks (not discussed today) to implement light error detection. (For engineering reasons, higher-speed communications are more error prone. The basic idea is that at higher data rates, voltages have less time to correctly swing between high and low values.) Compact disks have moderately strong error-correction abilities (specifically, Reed-Solomon codes): this is what permits them to keep working in spite of little scratches. Deep-space applications have more stringent error-correction demands ([**VO]**). An interesting recent innovation is home networking over ordinary power lines ([**Gib**]). Naturally, this requires very strong error correction since it must keep working even when the vacuum cleaner is switched on.

Chapter one

Fundamental terms and examples

1. Fundamental terms and examples

1.1 The binary symmetric channel

We are confining ourselves to communications channels in which only random errors occur, rather than burst or synchronization errors. Also, for today we will talk about binary codes, where the alphabet is {0,1}. We can quantify the random-error property a bit more by assuming there is a probability p of any bit being flipped from 0 to 1 or vice versa. This model of a transmission medium is called the binary symmetric channel, or BSC: binary since the symbols are bits, and symmetric since the probabilities of bit-setting and bit-clearing are the same.

The fundamental theorem of communication is **Shannon's theorem ([Sha])**, which when restricted to the BSC says that if p < 1/2, reliable communication is possible: we can always make a code long enough that decoding mistakes are very unlikely. Shannon's theorem defines the **channel capacity**, i.e. what minimum amount of redundant data needs to be added to make communication reliable. (See [**Sud**] for a nice proof.) Note however that Shannon's theorem proves only the existence of codes with desirable properties; it does not tell how .to construct them

In [**MS**] it is shown that if p is exactly 1/2 then no communication is possible, but that if p > 1/2 then one may interchange 0 and 1, and then assume p< 1/2. (For example, if p = 1, then it is certain that all 0's become l's and vice versa, and after renaming symbols there is no error whatsoever)

If n bits are transmitted in a block, the probability of all bits being wrong is $(1-p)^2$. The probability of an error in the first position is $p(1-p)^{n-1}$, and the same for the other single-position errors. Any given double error has probability $p^2(1-p)^{n-2}$, and so on; the probability of an error in all n positions is p. Since we assume p < 1/2, the most likely scenario is no error at all. Each single-bit error case is the next likely, followed by each of the double-bit error cases, etc. (For example, with n = 3 and p = 0.1, these probabilities are 0.729, 0.081, 0.009, and 0.001.) So, when I send you something that gets garbled in transit, you can only guess what happened to the message. But since we assume that fewer bit errors are more probable, you can use the **maximum .likelihood** assumption to help guide your guesses, as we will see below

1.2 Linear codes

In physics, to facilitate analysis of a problem one often makes certain simplifying assumptions. For example, orbital mechanics is simpler, but fractionally less accurate, if one assumes the earth is a perfect sphere rather than a lumpy oblate spheroid. In particular, one often makes assumptions that permit analysis of a system using linear rather than non-linear differential equations, since the former are easy to solve. In engineering, by contrast, one designs systems rather than studying preexisting systems: one has the liberty of .designing in linearity (and other simplifying assumptions) from the start

In this spirit, to facilitate analysis, we immediately replace the abstract alphabet $\sum$ with the finite field of q elements, $\mathbb{F}_q$. (Recall that a finite field has a prime-power number of elements. See **[LN]**

for background on finite fields. For today, $\boldsymbol{q}$ will simply be 2 so you won't need any particular expertise in finite fields.) Furthermore, since we divide a message $\boldsymbol{M}$ into blocks of k symbols each, i.e. k-tuples over $\mathbb{F}_q$, we have vectors over a field. This permits the application of the well-known and powerful tools of linear algebra

.Definitions 1.2.1 :

- **A block code** (here, we will just call it a **code**) is any subset of the set of all n-tuples over $\sum$ for some positive integer n. Since we take $\sum = \mathbb{F}_q$, this means that a code is any subset $\boldsymbol{C}$ of the vector space $\mathbb{F}_q^n$.

- If C is not just a subset of $\mathbb{F}_q^n$, but a subspace as well, then we say that **C** is a **linear code,** In this case, we take $\boldsymbol{k}$ to be the dimension of **C**. (All codes discussed today will be linear)

- The parameter $\boldsymbol{k}$ is called the **dimension** of the linear code **C**; n is called the **length** of C

- **The encoding problem** is that of **embedding** the smaller vector space $\mathbb{F}_q^k$ into the larger vector space $\mathbb{F}_q^n$ in a maximal way as will be discussed below.

- A vector in $\mathbb{F}_q^k$, is called a **message word**; its image in C is called a codeword

● During transmission, a codeword may be turned into any element of $\mathbb{F}_q^n$. We will call this a **received word**.

Notation. For brevity, we will often write n -tuples in the form 111 rather than (1,1,1). There is no ambiguity as long as each coordinate takes only a single digit, which is certainly the case over $\mathbb{F}_2$

1.3 The repetition codes

Example. The three-bit **repetition** code embeds $\mathbb{F}_2$ into $\mathbb{F}_2^3$ via the following: $0 \rightarrow 000$ and

$1 \rightarrow 111$. Here, $\boldsymbol{k} = 1$ and $n = 3$. Note that there are $2^3 = 8$ elements of $\mathbb{F}_2^3$ .but only two of them are codewords

More generally, we have a **family** of n -bit repetition codes, embedding $\mathbb{F}_2$ into $\mathbb{F}_2^3$: 0 maps to the vector consisting of n zeroes, and 1 maps to n ones. Clearly, .these are linear codes

1.4 Minimum distance

Definition 1.4.1 . The **Hamming weight** of a vector $\boldsymbol{v}$ in $\mathbb{F}_q^n$ is given by the number of non-zero entries in $\boldsymbol{v}$. This is a function $\boldsymbol{\omega}: \mathbb{F}_q^n \rightarrow \boldsymbol{Z}$. For example, .$\omega(101) = 2$

Definition 1.4.2 . The **Hamming distance** between vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ in $\mathbb{F}_q^n$ is given by the number of non-zero entries in their difference. That is, d: $\mathbb{F}_q^n \times \mathbb{F}_q^n$ $\rightarrow Z$ is given by $d(\boldsymbol{u}, \boldsymbol{v}) = w(\boldsymbol{u} - \boldsymbol{v})$. For example, d (101, 110) = $\omega(011) = 2$

Remark. When q =2 this means we simply count the number of .differing slots

Definition 1.4.3 . The minimum distance of a code $\boldsymbol{C}$ is the smallest distance between distinct pairs of vectors of $\boldsymbol{C}$. If $\boldsymbol{C}$ is linear, then the difference of $\boldsymbol{u}$ and $\boldsymbol{v}$ is also in $\boldsymbol{C}$, so the minimum distance is

then the **minimum weight** over all non-zero vectors in $\boldsymbol{C}$. For example, the three-bit repetition code has minimum distance 3. We overload the letter d by writing the minimum distance of $\boldsymbol{C}$ as $d(\boldsymbol{C})$, or simply d. From the context, it's .clear which meaning of d is intended

Note: For some codes it is clear what the minimum distance is. For others,) while it may be relatively easy to compute a lower bound on a code's minimum distance, the **true minimum distance** may be much harder to find. For some families of codes, true minimum distances are unknown.)

1.5 Error detection and error correction

By example, we will see how error-detection and error-correction abilities of a code are related to the code's minimum distance. Suppose we are sending single 0's and 1's using a three-bit repetition code. You may trust me to encode only 0 or 1, as 000 or 111, respectively, but due to noise you might receive any of 000, 001, 010, 011, 100, 101, 110 or 111. If you were to receive the block 111, then you may assume that either I sent 111 and all bits are intact, or I sent 000 and there was a triple bit error. Using the **maximum likelihood** assumption from above, the former conclusion is the more likely. Now suppose you receive the message 101 from me. Which is more likely: that I sent 000 and two bits were flipped, or that I sent 111 and the middle bit was flipped? Again, the .latter is the more likely

That is :

● If you receive 000 (weight 0), then you decode to 0, and you assume there were no errors in transmission.
● If you receive 100, 010, or 001 (weight 1), then you decode to 0, and you • believe there was a single bit error in transmission.

● If you receive 110, 101, or 011 (weight 2), then you decode to 1, and you .believe there was a single bit error in transmission

● If you receive 111 (weight 3), then you decode to 1, and you assume there .were no errors in transmission

In the following figure I mark codewords with an open circle. Maximum-likelihood decoding involves finding the codeword which is nearest to a given :received word

Figure (1.1) : Hamming distance $\mathbb{F}_2^3$

And Hamming weight $\mathbb{F}_2^3$

For this 3-bit repetition code, you can correctly **detect** any one-bit error. If a triple bit error occurs, you won't know it; if a double-bit error occurs, it will look like a single-bit error instead. In these latter two cases, you would have made a

decoding error.

Now suppose we use a four-bit repetition code. **I** encode 0 as 0000 and 1 as 1111. If you receive a vector of weight 0 or 1, you decode to 0; if you receive a vector of weight 3 or 4, then you decode to 1. However, if you receive a vector with two zero bits and two one bits, then you know something is wrong (**I** can be trusted to only have sent 0000 or 1111, neither of which you got), but it's a coin toss whether two bits got set by error, or two bits got cleared by error :

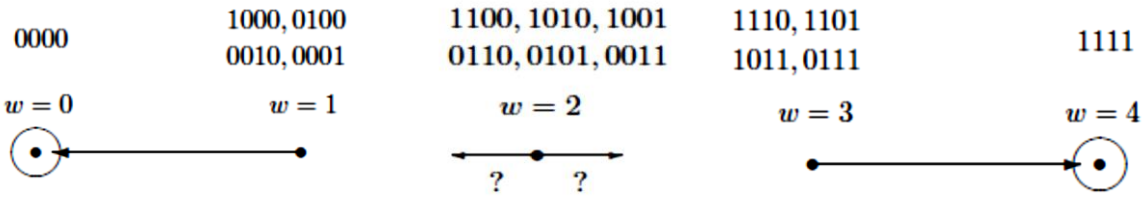


Figure (1.2) : Hamming distance $\mathbb{F}_2^4$

And Hamming weight $\mathbb{F}_2^4$

For this 4-bit repetition code, you can reliably correct any 1-bit error, .but you can only **detect** a 2-bit error

More generally, we see intuitively that if the minimum distance d of a code $\boldsymbol{C}$ is odd, then C can detect and correct up to $(d - 1)/2$ errors per

block. If d is even, then $\boldsymbol{C}$ can correct up to $d/2$ - 1 errors per block, and can detect up to $d/2$ errors per block.

Thus, when an error-control system is being designed, the error statistics of the transmission medium must be known so that the minimum distance can be made high enough that the chance of $d/2$ or more errors occurring in a block is vanishingly small. (Shannon's theorem guarantees the existence of such codes.) Any fixed code may be defeated by worse-than-expected noise: either a system must be designed to handle worst-case noise, or it must be parameterized such that parameters may be adaptively adjusted at run time to match .changing channel conditions

1.6 The even-weight parity check codes

Let n=k+1. Embed $\mathbb{F}_2^k$ into $\mathbb{F}_2^n$ by sending

(a_0 , a_1, ...,a_k) to (a_0, a_1, ..., a_k, a_0+...+a_k)

where the sum is taken mod 2. For example, with k = 4, 1110 maps to 11101. By construction, every codeword has even weight. The extra bit may be thought of as a parity bit: It is 0 when the input message word has an even number of 1 bits, and 1 when the input message word has an odd number of 1 bits. (Of course, we could define an odd-weight parity-check code as well. Since it would lack the zero vector, though, it would be not be a subspace of $\mathbb{F}_2^n$.) Thus, these are called the **even-weight parity** check codes.

Since $\mathbb{F}_2^k$ consists of all k-tuples, including those with 1 in a single position and zeroes elsewhere, the code contains some (k+1)-tuples of weight 2. Since all

codewords have even weight, this means that these parity-check codes have minimum distance 2. From the above discussion, this means

they can detect single-bit errors, but can't correct any errors at all. These are useful in the case when the probability of a single bit error is quite small but non-zero, and the probability of a double bit error is vanishingly small. They .enable the receiver to flag a block as bad, and request the sender to retransmit it

As with the repetition codes, these form a **family** of codes: the n-bit even-weight parity-check codes, embedding $\mathbb{F}_2^{n-1}$ into $\mathbb{F}_2^n$. Since q = 2, the difference of two even-weight vectors is another even-weight vector. Thus these are linear codes.

1.7 A graphical perspective

Here is what the 3-bit repetition and parity-check codes look like, respectively, inside $\mathbb{F}_2^3$:

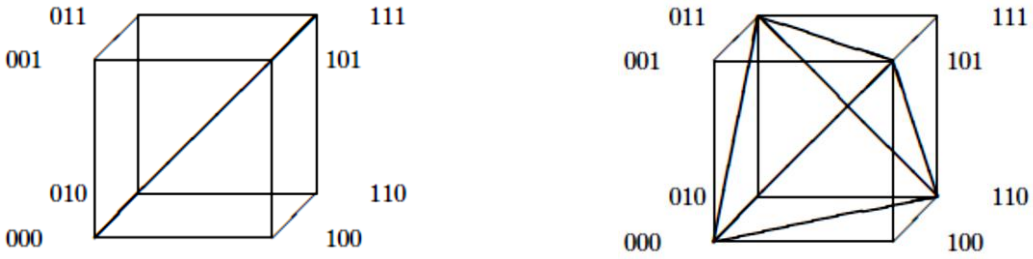


Figure (1.3) : Graphical perspective inside $\mathbb{F}_2^3$

On the left, $\mathbb{F}_2$ could have been sent to any edge, e.g. 000 and 001, but the two codewords would have distance 1 between them; as shown, they have distance 3. Likewise, on the right, $\mathbb{F}_2^2$ could have been sent to a face of the cube, with minimum distance 1; as shown, the codewords are spread out over the cube, as far apart from one another as possible, with minimum distance 2. These are clearly the highest-distance 1-dimensional and 2-dimensional subspaces, respectively, of $\mathbb{F}_2^3$. Here we have q = 2, n = 3 and k = 1 or 2. For higher n, k and q, though, it's not immediately obvious how to spread out codewords in this .maximum-distance manner

In general, the encoding problem consists in large part of finding a way of constructing such embeddings such that all codewords are as far apart from one another as possible. This problem clearly is combinatorial in nature. However, in recent years various approaches have happened to prove fruitful, including finite **([**geometry **([AK])** and algebraic geometry **Pre2]) .**

1.8 Rate, relative minimum distance, and asymptotic

The repetition codes have good error-correcting ability. However, the drawback is that most of the transmitted data is redundant. For the 5-bit repetition codes, only one of every 5 bits is actual data. The parity-check codes, on the other hand, add just a single redundant bit, but tolerate fewer errors.

Definition (1.8.1) . The **rate** of a code is the ratio R = k/n.

The repetition codes have rate R = 1/n. As n increases, R approaches zero. The .parity-check codes have rate R = $(n - 1)/n$, which approaches 1 as n increases

Definition (1.8.2) . The **relative minimum distance** of a code is the ratio $\delta = d/n$.

The repetition codes have relative minimum distance $\delta = n/n = 1$. The parity-check codes have relative minimum distance $2/n$, which approaches 0 as n .increases

Of course, R and δ are both confined to the unit interval. We say that **asymptotically** (as n gets big) the repetition-code family has R = 0 and $\delta = 1$; asymptotically the parity-check family has R = 1 and $\delta = 0$. For large n, the repetition codes carry vanishingly little actual data; their overhead is too large. For large n, the parity-check codes detect vanishingly few errors per block; their overhead is too small.

Definition(1.8.3) . A good code (really, a good family of codes) is one whose asymptotic rate and asymptotic relative minimum distance are both bounded away from zero.

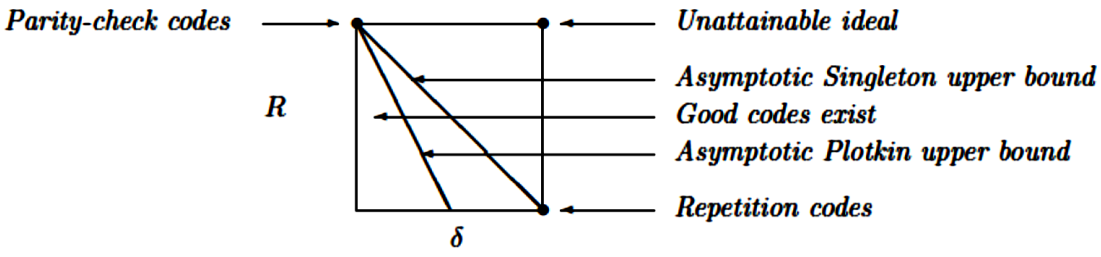
Clearly, the repetition and parity-check codes are not good. It can be shown that good codes exist; see **[MS]** for examples.

1.9 Code parameters; upper and lower bounds

A linear code is parameterized by the four integers n, k, d and q, or equivalently by the four rational numbers n, R, δ and q. Sometimes one says that $\boldsymbol{C}$ is an $[n, k, d]$ code, or perhaps an $[n, k, d]\ q$ code. For example, the 5-bit binary repetition code is a $[5, 1, 5]_2$ code. (Similarly, we might write an asymptotic parameterization of a family of codes as [R, δ] or $[\ R, \delta]\ _q$)

We encode by embedding $\mathbb{F}_q^k$ into $\mathbb{F}_q^n$**.** Not any embedding will do: as we saw in section 2.7, the canonical injection which appends n - k zeroes is an embedding, but it has minimum distance 1. We want to find an embedding which keeps the vectors as far apart from one another as possible, maximizing $\mathbf{d}_{\mathbf{C}}$, in order to maximize the code's error-control ability. Or, given a fixed minimum distance, we would like to minimize n or maximize **k**, to keep the code's rate high. Ideally we would like the rate and the relative minimum distance to both be high, but there are results (see **[MS**], [**PW],** [**Wal**]) which show that there are **upper bounds** on the asymptotic rate and relative minimum .distance

Both R and δ are in the unit interval, so we may think of a parameter space which looks like the unit square :



Note that there are particular codes with parameters in various places on this square. The zero- appending code, given by the map from $\mathbb{F}_q$ to $\mathbb{F}_q^n$ which appends $n + 1$ zeroes, has R = $1/n$ and $\delta = 1/n$. Asymptotically, both are zero. Also, the identity code with $\boldsymbol{n = k = 1}$ has R = 1 and $\delta = 1$. However, this .has no error-control ability at all

The **Singleton bound** states that for all codes, $d_C \leq n - k + 1$. Thus, for any code with $n > 1$, the R = 1, $\delta = 1$ corner is unattainable. Asympotically, the

Singleton bound shows that $R + \delta \leq 1$. This means that the asymptotic (R, δ) of a family of codes must be below the main diagonal. Clearly, this applies to even the identity codes, for which $R = 1$ but $\delta \rightarrow 0$. The Plotkin bound shows that the asymptotic (R, δ) must be below the lower diagonal as well, where the δ intercept is $1 - 1/q$. See **[Wal]** for a lucid discussion of these and other bounds.

The Singleton and Plotkin bounds provide upper limits on the best code families: no codes can be asymptotically better. There are also **lower bounds** which specify how good the best codes can be, but don't constrain how bad the worst codes can be (for example, the zero-appending code mentioned above). One proves a lower bound, showing that there exist codes with (R, δ) above some curve in R, δ space; the problem of actually producing such codes is .another problem entirely. Both of these issues are topics of research

Chapter two

Encoding

2.1 Encoding

2.1.1 The generator matrix

Up to now we haven't really put much linear algebra to work. To facilitate analysis, we now require not only that we have linear block codes mapping injectively from $\mathbb{F}_q^K$ into $\mathbb{F}_q^n$, but furthermore that the injective mapping is a vector-space homomorphism, i.e. a linear transformation. There are many advantages to using a linear transformation, not the least of which is that instead of having to remember $\boldsymbol{q^K}$ images for all the message words, we need to remember only the images of k basis vectors.

Such a linear transformation exists for any linear code. For example, {000, 100,010, 110} is a subspace of $\mathbb{F}_2^3$, but I can send $\mathbb{F}_2^2$ into it by $00 \rightarrow 110$, $01 \rightarrow 100$, $10 \rightarrow 000$, $11 \rightarrow 010$. This is a 1-1 map but it isn't linear since it doesn't send zero to zero. As long as I don't insist on which elements of $\mathbb{F}_q^K$ map to which elements of **C**, though, I can produce a linear map: since $\mathbb{F}_q^K$ and C are vector spaces of the same dimensions over the same field, an isomorphism exists. To obtain it explicitly if only C is given, form a tall matrix the rows of which are all the vectors of C, then row-reduce and discard zero rows. The result is a basis for C. Then, send the ith standard basis vector in $\mathbb{F}_q^K$ to the ith basis vector of C.

However it is obtained, we write **a generator matrix**

$$\mathbf{G}: \mathbb{F}_q^K \rightarrow \mathbb{F}_q^n$$

where C is the image of G in $\mathbb{F}_q^n$. For convenience later on (although it seems quite strange at the moment), we write G as a **k x** $\boldsymbol{n}$ matrix: to encode the message word **m**, we write **mG** rather than **Gm**. (If this seems awkward, you may wish to temporarily think in terms of an $n \times$ k generator matrix, then transpose it when you're done. Also, from the context it is clear whether I'm treating **m** as a row or column vector.)

What is a generator matrix for the repetition codes? Clearly, we write (n = 5 here)

$$G=[11111]$$

For the parity-check code, we want (with n = 5)

$$[\,a\,,\,b,\,c,\,d,\;\; a+b+c+d] = [a\,,\,b,\,c,\,d\,] \begin{bmatrix} 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix}$$

So

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

Of course, when $\boldsymbol{\varepsilon_i}$ is the ith standard basis vector for $\mathbb{F}_q^K$, $\boldsymbol{\varepsilon_i G}$ is the $\boldsymbol{i}$th row of $\boldsymbol{G}$. Unless $\boldsymbol{\kappa} = 1$ and $q = 2$, there will be more than one basis vector, hence more than one permutation of the basis, along with various linear combinations of the basis vectors. Thus the generator matrix is generally not unique. Two different generator matrices are equivalent, though, if they generate the same subspace $\boldsymbol{C}$ of $\mathbb{F}_q^n$. To test for equivalence of two generator matrices, test for equality of their row-echelon forms.

(Computational note: finite fields have the property that computer arithmetic is exact. Thus, there is no roundoff error, and algorithms such as row reduction may be implemented easily for finite fields, with naive pivoting.)

2.1.2 Systematic codes

We've been saying that a linear code $\boldsymbol{C}$ is a $\boldsymbol{k}$-dimensional subspace of $\mathbb{F}_q^n$. From this definition, $\boldsymbol{G}$ could take any form as long as it has rank $\boldsymbol{k}$. However, our two examples so far (repetition and parity-check codes) have an additional property: the first $\boldsymbol{k}$ bits of each n-bit codeword are identical to the $\boldsymbol{k}$ bits of the corresponding message word.

Definition (2.1.2.1) . A linear code is **systematic** if its generator matrix $\boldsymbol{G}$ is of the form $[\mathbf{I}_K|A]$ for some $\boldsymbol{k}$ x (n - $\boldsymbol{k}$) matrix A, where $\mathbf{I}_K$ is the $\boldsymbol{k}$ x $\boldsymbol{k}$ identity matrix.

Any linear code can be made systematic: just put $\boldsymbol{G}$ in row-echelon form.

2.2 Decoding

2.2.1 The parity-check matrix

If a linear code C has been chosen, we've just seen that encoding is easy: it's just matrix multiplication. But how do we decode, and moreover, how do we do so efficiently? This might seem to be a harder problem. In fact, in general it is. There have been codes which were published before any decoding algorithm was known. And even for well-known codes, one area of current research is to develop improved decoding algorithms.

Below, it will be useful to find a so-called **parity-check matrix**, **H**, such that C is precisely the kernel of **H**. (The terminology originally comes from parity-check codes, but it is a poor choice of words: all linear codes, not just the parity-check ones, have a parity-check matrix.) That is, we will want H $\boldsymbol{v}$ to be zero if and only if $\boldsymbol{v}$ is in C. By the rank-nullity theorem, H will necessarily be $(n - k) \times n$. Unlike with G, we post-multiply, i.e. we write H $\boldsymbol{v}$, not $\boldsymbol{v}$ H.

How can such a matrix H be constructed, given G? First, some terminology.

Definition (2.2.1.1) . The **dual code** of C, written $C^{\perp}$, is the set of vectors in $\mathbb{F}_q^n$ which are orthogonal to all vectors of C, using the standard dot product. (Note that the term dual code here has nothing to do with the term dual space from linear algebra.) That is,

$$C^{\perp} = \{\boldsymbol{v} \in \mathbb{F}_q^n : \mathcal{U} \cdot \boldsymbol{v} = 0 \text{ for all } \mathcal{U} \in C\}$$

(The Hamming weight is a vector-space norm, if we define $|c|$ con F_q, to have value 0 when c = 0, 1 otherwise. If we use the standard dot product, then $\mathbb{F}_q^n$ satisfies all the axioms for an inner product space except for the positive-definiteness of the dot product. E.g. if F_q has characteristic 2, the non-zero vector (1,1) dotted with itself is 1+1 = 0. Note that the Hamming weight is computed in $\mathbb{Z}$: it is the number of non-zero coordinates in a vector. However, the dot product is computed in F_q. Thus the Hamming weight and Hamming distance are positive definite, while the dot product is not. This means that inner-product-space results such as $\mathbb{F}_q^n = C \oplus C^{\perp}$ do not apply: the intersection of a subspace and its perp can contain more than just the zero vector. In fact, a code can be self dual, i.e. $C = C^{\perp}$ For example, {00, 11} is a self-dual subspace

of $\mathbb{F}_2^2$. From the result immediately below, a self-dual code must have even n, and k must be n /2.)

We already have G; it remains to actually compute a matrix for H. Suppose that our problem were reversed, i.e. if we had H, how would we compute G? That's easy: since the kernel of H is the image of G (which is C) we could just compute the kernel basis of H, which is a standard elementary linear algebra problem. G would have rows equal to the elements of that basis.

Now, I claim that, fortuitously, the generator matrix of $C^\perp$ is H and the parity-check matrix of $C^\perp$ is G. That is, $C^\perp$'s G and H are swapped from C's. Also note that $(C^\perp)^\perp$ is just C. We are given G, which is C's generator matrix as well as $C^\perp$'s parity-check matrix. The kernel basis of G is the generator matrix for $C^\perp$ which is also the parity-check matrix for C. So this trick means that not only can we get a G by computing a kernel basis of an H, but vice versa as well.

It remains to prove that $C^\perp$ has generator matrix $\boldsymbol{H}$ and parity-check matrix G. Remember the convention that a generator matrix acts by post-multiplication and that a parity-check matrix acts by pre-multiplication. So in this role, $\boldsymbol{H}$ maps $\mathbb{F}_q^{n-k}$ to $\mathbb{F}_q^n$ by sending $\boldsymbol{z}$ to $\boldsymbol{zH}$, and $\boldsymbol{G}$ maps $\mathbb{F}_q^n$ to $\mathbb{F}_q^k$ by sending $\boldsymbol{v}$ to $\boldsymbol{Gv}$. To avoid confusion (only for the duration of this proof) we will write $\boldsymbol{G}$ for $\boldsymbol{G}$: $\mathbb{F}_q^k \to \mathbb{F}_q^n$ acting by post-multiplication and $\boldsymbol{G}$. for $\boldsymbol{G}$: $\mathbb{F}_q^n \to \mathbb{F}_q^k$ acting by pre-multiplication. Likewise, we will write $\boldsymbol{H}$. for $\boldsymbol{H}$: $\mathbb{F}_q^n \to \mathbb{F}_q^{n-k}$ and $\cdot\,\boldsymbol{H}$ for $\boldsymbol{H}$: $\mathbb{F}_q^{n-k} \to \mathbb{F}_q^n$. Plain $\boldsymbol{G}$ and $\boldsymbol{H}$ refer to the matrices without respect to a linear transformation.

We want the following short exact sequences:

$$0 \to \mathbb{F}_q^k \xrightarrow{\cdot G} \mathbb{F}_q^n \xrightarrow{H\cdot} \mathbb{F}_q^{n-k} \to 0$$
$$0 \leftarrow \mathbb{F}_q^k \xleftarrow{G\cdot} \mathbb{F}_q^n \xleftarrow{\cdot H} \mathbb{F}_q^{n-k} \leftarrow 0$$

with im($\boldsymbol{G}$)=C=ker($\boldsymbol{H}$) and im(H$\boldsymbol{H}$) = $C^\perp$ = ker($\boldsymbol{G}$). The short exactness means that .$\boldsymbol{G}$ and $\boldsymbol{H}$ are 1-1, while $\boldsymbol{H}$. and $\boldsymbol{G}$: are onto. Thus, it suffices to show: (1) im($\boldsymbol{H}$) = $C^\perp$; (2) $\boldsymbol{H}$ is 1-1, (3) ker($\boldsymbol{G}$) = $C^\perp$, and (4) $\boldsymbol{G}$ is onto. Now, we already have that the matrix $\boldsymbol{G}$ has rank $\boldsymbol{k}$ and $\boldsymbol{H}$ has rank $n-k$. Since row rank equals

column rank, (4) will follow from (3) by the rank-nullity theorem. Likewise, (2) will follow from (1) since $C^{\perp}$ has dimension $n-k$.

To prove (3), first let $\boldsymbol{v} \in C^{\perp}$. The rows of $\boldsymbol{G}$ form a basis for C; let g_i be the ith row of $\boldsymbol{G}$, for i=1, ..., k, where each g_i, is a vector of length n (since it is in $\mathbb{F}_q^n$). Also let $\boldsymbol{v}$ = (v_1,...,v_n). The matrix-times-vector multiplication $\boldsymbol{G}.\boldsymbol{v}$ consists of dot products of v with the rows of $\boldsymbol{G}$:

$$G \cdot v = \begin{bmatrix} g_1 \\ \vdots \\ g_k \end{bmatrix} \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} g_1 \cdot v \\ \vdots \\ g_k \cdot v \end{bmatrix}$$

Since each g_i ; is in C and since $\boldsymbol{v}$ is in $C^{\perp}$, all the dot products are zero and so $\boldsymbol{G}.\boldsymbol{v}$=0.

Conversely, let $\boldsymbol{v} \in$ ker($\boldsymbol{G}$.). Then $\boldsymbol{G} \bullet \boldsymbol{v}$ = 0. Again, this product consists of dot products of rows of $\boldsymbol{G}$ with $\boldsymbol{v}$, so $g_i \bullet \boldsymbol{v}$ = 0 for all g_i's. Let c be an arbitrary element of C. Since the g_i's are a basis for C, c = $\sum_{i=1}^{k} c_i\, g_i$ for some $c_i\, g_i$ in.$\mathbb{F}_q$ Then

$$\boldsymbol{v} \bullet \mathrm{c} = \boldsymbol{v} \,.\sum_{i=1}^{k} c_i\, g_i$$

$$=\sum_{i=1}^{k} c_i(v \,.\, g_i\,) \;= 0$$

Therefore $\boldsymbol{v} \in C^{\perp}$

To prove that im($\boldsymbol{H}$) = $C^{\perp}$ notice in general that when a matrix X acts on a standard basis by Xεi, the image of that basis consists of the columns of X. Likewise, when X acts on a standard basis by εi**X**, the image of that basis consists of the rows of X. It will suffice to show that the rows

of $\boldsymbol{H}$ are a basis for C+. Remember that we set up $\boldsymbol{H}$ to check the elements of C, and since G has rows forming a basis for C, necessarily

$$HC^{\perp} = 0$$

This means that the rows of $\boldsymbol{H}$ are orthogonal to the rows of G, which shows that the rows of $\boldsymbol{H}$ are in C-. Since we know that $\boldsymbol{H}$ has rank n – k, the image of the standard basis for $\mathbb{F}_q^{n-k}$ under $\boldsymbol{H}$ is linearly independent, and im($\boldsymbol{H}$) must be all of Ch.

Example(2.2.1.1) . Let's carry out this computation for the two families of codes we've seen so far. The 5-bit repetition code has generator matrix

$$G=[1\ 1\ 1\ 1\ 1]$$

We then compute the kernel basis (in row-echelon form)

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

Intuitively, this makes sense: recalling that we are working mod 2, this means that $\boldsymbol{H}\ \boldsymbol{v}$ is 0 only when $\boldsymbol{v}$ has all coordinates the same. The two possible cases are 00000 and 11111, which are precisely the codewords of the 5-bit repetition codes.

Next, the 5-bit parity-check code has generator matrix

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

We then compute

$$H = [1\ 1\ 1\ 1\ 1]$$

Intuitively, this also makes sense: pre-multiplying $\boldsymbol{v}$ by $\boldsymbol{H}$ just adds up the bits of v mod 2. The result will be zero precisely when $\boldsymbol{v}$ has even parity, which is the case iff $\boldsymbol{v}$ is in C.

As an added bonus, since the first G is the same as the second $\boldsymbol{H}$, and vice versa, we now see that the repetition and parity-check families are duals of one another.

2.2.2 Computing $\boldsymbol{H}$ for systematic codes

The parity-check matrix is particularly easy to compute when C is systematic, i.e. when $\boldsymbol{G}$ is in row-echelon form. For example, take $\boldsymbol{k} = 3$, $n = 6$ and suppose

$$\boldsymbol{G} = [I_k \mid A] = \begin{bmatrix} 1 & 0 & 0 & a_{14} & a_{15} & a_{16} \\ 0 & 1 & 0 & a_{24} & a_{25} & a_{26} \\ 0 & 0 & 1 & a_{34} & a_{35} & a_{36} \end{bmatrix}$$

Writing $G^t\boldsymbol{m}$ rather than $\boldsymbol{m}\mathsf{G}$ to save horizontal space on the paper, a message word (m_1,m_2 m_3) is encoded as

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ a_{14} & a_{24} & a_{34} \\ a_{15} & a_{25} & a_{35} \\ a_{16} & a_{26} & a_{36} \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \end{bmatrix} = \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ a_{14}\,m_1 + a_{24}m_2 + a_{34}m_3 \\ a_{15}m_1 + a_{25}m_2 + a_{35}m_3 \\ a_{16}m_1 + a_{26}m_2 + a_{36}m_3 \end{bmatrix}$$

We want to write an $\boldsymbol{H}$ such that $\boldsymbol{H}$ times this codeword is zero, but that's easy:

$$\begin{bmatrix} -a_{14} & -a_{24} & -a_{34} & 1 & 0 & 0 \\ -a_{15} & -a_{25} & -a_{35} & 0 & 1 & 0 \\ -a_{16} & -a_{26} & -a_{36} & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ a_{14}\,m_1 + a_{24}m_2 + a_{34}m_3 \\ a_{15}m_1 + a_{25}m_2 + a_{35}m_3 \\ a_{16}m_1 + a_{26}m_2 + a_{36}m_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

We can generalize this example to see that if

$$\boldsymbol{G} = [I_k \mid A]$$

Then

$$\boldsymbol{H} = [\, -A^T \mid I_{n-k}]$$

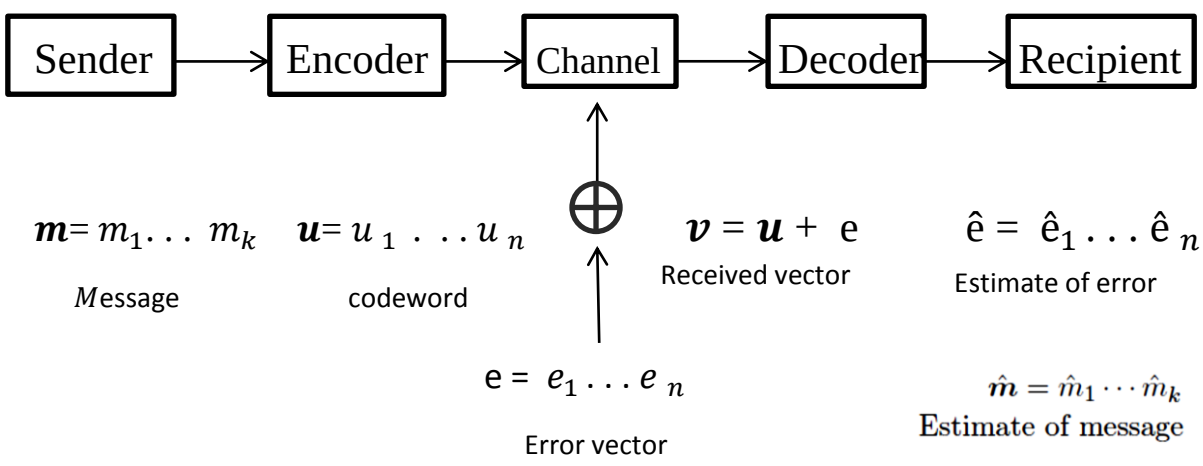
Thus, systematic G and $\boldsymbol{H}$ may be computed from one another by inspection.

2.2.3 Brute-force decoding

If ***I*** send you an encoded message, with errors in transit, how do you decode to find out what ***I*** really meant to say? If the message space is small, i.e. if $\boldsymbol{q}$ and $\boldsymbol{n}$ are small, then you could simply make a list of all possible elements of $\mathbb{F}_q^n$, with the nearest-neighbor codeword precomputed by hand for each. This is what we did in section 2.5 However this is infeasible for larger codes, which might have billions of codewords or more: it requires having a table of size q^n.

2.2.4 The coat of arms

A useful diagram from [**MS**], attributed therein to David Slepian, is the following:



Since we are assuming our channel only inserts random errors, without changing the block length by loss of synchronization, we can think of the error vector as being added to the codeword during transmission. Since we embed our $\mathbb{F}_q^k$ into $\mathbb{F}_q^n$ using a linear transformation (rather than any old injective map), and since $\boldsymbol{Hu}$ is zero for all codewords $\boldsymbol{u}$, we have the following key fact:

$$\boldsymbol{v} = \boldsymbol{u} + \mathrm{e}$$

$$H\,\boldsymbol{v} = H(\boldsymbol{u} + \mathrm{e}) = H\,\boldsymbol{u} + H\mathrm{e} = H\mathrm{e}$$

2.2.5 Standard-array decoding

Definition(2.2.5.1) . The quantity $\boldsymbol{Hv}$ =$\boldsymbol{H}$e from the previous section is called the syndrome of $\boldsymbol{v}$. When we form the quotient space $\mathbb{F}_q^n$/C, from elementary algebra we know that the cosets of C partition $\mathbb{F}_q^n$. Since C is precisely the kernel of $\boldsymbol{H}$, if two received vectors are in the same coset,

$$u \sim \boldsymbol{v}$$

$$u - \boldsymbol{v} \in \mathsf{C}$$

$$H(u - \boldsymbol{v}) = 0$$

$$Hu = H\boldsymbol{v}$$

Thus, two vectors are in the same coset iff they share the same syndrome.

Now, u was transmitted; $\boldsymbol{v} = u$ + e was received, but the receiver can only guess at what e is. Since $\boldsymbol{v}$ and e have the same syndrome, the true error vector e is somewhere in $\boldsymbol{v}$'s coset. Furthermore, since we are using the **maximum likelihood** assumption mentioned in section 2.1, the most likely error vector **ê** is the smallest-weight vector in $\boldsymbol{v}$'s coset. (A decoding error means e ≠ ê.)

So, the **standard-array decoding algorithm** has two stages: the first stage is some precomputation before any data is received; the second is done as each block is received.

Precomputation stage:

- Write down the elements of $\mathbb{F}_q^k$ and encode each element. This is a two-by-q^k table, pairing up message words and codewords. Sort this by codeword for easy lookup later.

- Write down the quotient space $\mathbb{F}_q^n$/C. This requires making, for the moment, a matrix of all q^n elements of $\mathbb{F}_q^n$. (Note that this algorithm is also not OK for large codes, although the resulting tables will be smaller than for the brute-force method.)

- For each coset, search for the smallest-weight element in the coset. This is called the coset leader. Compute and remember the syndrome of the coset leader; forget about the rest of the coset.

- Make a list pairing up syndromes and coset leaders. This is a two-by-q^{n-k} table. Sort this by syndrome for easy lookup later.

Decoding stage:

• Given a received vector $\boldsymbol{v}$, compute its syndrome s.
• Look up this syndrome in the precomputed syndrome / leader table.
• Find the most likely error vector **ê** corresponding to s.
• Compute û = $\boldsymbol{v}$ - ê.
• Look up **û** in the precomputed message/codeword table to obtain $\hat{\boldsymbol{m}}$.
.This is our best guess of what the transmitter sent

Note that both table lookups are done on sorted data. This means we don't have to sequentially scan either table at run time. The syndromes are all of $\mathbb{F}^{n-k}$, so we can use base-*q* arithmetic to go directly to the desired element of the syndrome /leader table. For the message/codeword table, we can use a binary search, with a number of lookups roughly log2 of the table size.

(**Note** that the message/codeword table isn't necessary. Once we have a codeword $\hat{\boldsymbol{u}}$, we can solve the linear system $\hat{\boldsymbol{u}} = \hat{\boldsymbol{m}}\boldsymbol{G}$ for $\hat{\boldsymbol{m}}$ using row reduction. This reduces table space, at the expense of making the decoding stage use more computation.)

Example(2.2.5.1). Let's compute the standard array for the 3-bit repetition code. We have

$$G = [\,1\ 1\ 1\,]\ ,\ H = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$

The message words are 0 and 1. Their images under G are 000 and 111. So, the message/codeword table is as follows:

m	u
0	000
1	111

The possible received vectors (all of $\mathbb{F}_q^n$) are:

000,001,010,011, 100, 101, 110, 111

C is: 000,111

$\mathbb{F}_q^n$/C is:

{000, 111},
{010, 101},
{100, 011},
{001, 110}

The coset leaders are the minimum-weight vectors of each coset. These are as follows, with corresponding syndromes:

ê	s
000	00
010	01
100	10
001	11

Here is an example of using the standard array to decode a received vector:

• Receive $\boldsymbol{v}$ = 011.

• Compute s = $\boldsymbol{Hv}$ = 10

• 10 in binary is 2 in decimal, so go to row 2 (with row indices starting at 0) of the syn-drome/leader table.

• At that spot, find **ê** = 100.

• Compute $\hat{u} = \boldsymbol{v} - \hat{e} = 011 - 100 = 111$.

• Match codeword 111 with message word 1 in the message/codeword table to obtain $\hat{m} = 1$.

التوصيات

1- يوصي الباحثان بأن يتم استخدام هذا التصميم في دراسة المصفوفات التي تكون على شكل مصفوفات مربعة

2- يتم استخدام هذا التصميم في ارسال المعلومات عبر channel قناة الارسال في هندسة الاتصالات

References

1-[**AK**] Assmus, E.F. and Key, J.*d*. Designs and Their Codes. Cambridge University Press, 1994.

2-[Ber] E. Berlekamp. Algebraic Coding Theory (revised 1984 edition). Aegean Park Press, 1984.

3-[Gol] Golay, M.J.E. Notes on digital coding. Proceedings of the IRE, 37:657, June 1949.

4-[**Gib**] Gibbs, W.W. The Network in Every Room. Scientific American, February 2002.

5-[Ham] Hamming, R.W. Error Detecting and Error Correcting Codes. Bell System Technical Journal, 29:147-160, April 1950.

6-[HKCSS] Hammons, A.R. et al. The Z4-Linearity of Kerdock, Preparata, Goethals and Related Codes. http://www.research.att.com/onjas/doc/linear.ps

7-[LN] R. Lidl and H. Niederreiter. Finite Fields. Cambridge University Press, 1997.

8-[**MS**] MacWilliams, F.J. and Sloane, *n*.J.A. The Theory of Error-Correcting Codes. Elsevier Science B.V., 1997.

9-[PW] Peterson, W.W. and Weldon, E.J. Error-Correcting Codes (2nd ed.). MIT Press, 1972.

10-[**Pre1**] Pretzel, O. Error-Correcting Codes and Finite Fields. Oxford University Press, 1996.

11-[**Pre2**] Pretzel, O. Codes and Algebraic Curves. Oxford University Press, 1998.

12-[Sha] Shannon, C.E. A mathematical theory of communication. Bell System Technical Journal,

27:379-423, 623-656, 1948.

13-[Sud] Sudan, M. Algorithmic Introduction to Coding Theory.

http://theory.lcs.mit.edu/~madhu/coding/ibm

14-[VO] Vanstone, S.A. and van Oorschot, P.C. An Introduction to Error Correcting Codes with Applications. Kluwer Academic Publishers, 1989.

15-[Wal] Walker, J. Codes and Curves.

http://www.math. unl.edu/~jwalker/papers/rev.pdf

Table (1) : $\oplus$ and $\odot$ in $\mathbb{F}_2^1$

$\mathbb{F}_2^1 = \{ 0 , 1\}$

$(\mathbb{F}_2^1 , \oplus , \odot)$

$\oplus$	0	1	Hamming weight	Hamming distance
0	0	1	0	0
1	1	1	1	1

$\odot$	0	1	Hamming weight	Hamming distance
0	0	0	0	0
1	1	1	1	1

Table (2) : $\oplus$ and $\odot$ in $\mathbb{F}_2^2$

$\mathbb{F}_2^2 = \{ 00 , 01, 10,11\}$

$(\mathbb{F}_2^2 , \oplus , \odot)$

$\oplus$	00	01	10	11	Hamming weight	Hamming distance
00	00	01	10	11	0	0
01	01	00	11	10	1	1
10	10	11	00	01	1	1
11	11	10	01	00	2	2

$\odot$	00	01	10	11	Hamming weight	Hamming distance
00	00	00	00	00	0	0
01	00	01	00	01	1	1
10	00	00	10	10	1	1
11	00	01	10	11	2	2

Table (3) : $\oplus$ and $\odot$ in $\mathbb{F}_2^3$

$\mathbb{F}_2^3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$

$(\mathbb{F}_2^2 , \oplus , \odot)$

$\oplus$	000	001	010	011	100	101	110	111	Hamming Weight	Hamming distance
000	000	001	010	011	100	101	110	111	0	0
001	001	000	011	010	101	100	111	110	1	1
010	010	011	000	001	110	111	100	101	1	1
011	011	010	001	000	111	110	101	100	2	2
100	100	101	110	111	000	001	010	011	1	1
101	101	100	111	110	001	000	011	010	2	2
110	110	111	100	101	010	011	000	001	2	2
111	111	110	101	100	011	010	001	000	3	3

$\odot$	000	001	010	011	100	101	110	111	Hamming Weight	Hamming distance
000	000	000	000	000	000	000	000	000	0	0
001	000	001	000	001	000	001	000	001	1	1
010	000	000	010	010	000	000	010	010	1	1
011	000	001	010	011	000	100	010	011	2	2
100	000	000	000	000	100	100	100	100	1	1
101	000	001	000	001	100	101	100	101	2	2
110	000	000	010	010	100	100	110	110	2	2
111	000	001	010	011	100	101	110	111	3	3

Table (4) : $\oplus$ and $\odot$ in $\mathbb{F}_2^4$

$\mathbb{F}_2^4$ = {0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111}

($\mathbb{F}_2^4$, $\oplus$, $\odot$)

$\oplus$	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111	H.W	H.D
0000	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111	0	0
0001	0001	0000	0011	0010	0101	0100	0111	0110	1001	1000	1011	1010	1101	1100	1111	1110	1	1
0010	0010	0011	0000	0001	0110	0101	0100	0101	1010	1011	1000	1001	1110	1111	1100	1101	1	1
0011	0011	0010	0001	0000	0111	0110	0101	0100	1011	1010	1001	1000	1111	1110	1101	1100	2	2
0100	0100	0101	0110	0111	0000	0001	0010	0011	1100	1001	1110	1111	1000	1001	1010	1011	1	1
0101	0101	0100	0111	0110	0001	0000	0011	0010	1101	1110	1111	1110	1001	1000	1011	1010	2	2
0110	0110	0111	0100	0101	0010	0011	0000	0001	1110	1111	1100	1101	1010	1011	1000	1001	2	2
0111	0111	0110	0101	0100	0011	0010	0001	0000	1111	1110	1101	1100	1011	1010	1001	1000	3	3
1000	1000	1001	1010	1100	1100	1101	1110	1111	0000	0001	0010	0011	0100	0101	0110	0111	1	1
1001	1001	1000	1011	1010	1101	1100	1111	1110	0001	0000	0011	0010	0101	0100	0111	0110	2	2
1010	1010	1011	1000	1001	1110	1111	1100	1101	0010	0011	0000	0001	0110	0111	0100	0101	2	2
1011	1011	1010	1001	1000	1111	1110	1101	1100	0011	0010	0001	0000	0111	0110	0101	0100	3	3
1100	1100	1101	1110	1111	1000	1001	1010	1011	0100	0101	0110	0111	0000	0001	0010	0011	2	2
1101	1101	1100	1111	1110	1001	1010	1011	1010	0101	0100	0111	0110	0001	0000	0011	0010	3	3
1110	1110	1111	1100	1101	1010	1011	1000	1001	0110	0111	0100	0101	0010	0011	0000	0001	3	3
1111	1111	1110	1101	1100	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001	0000	4	4

Table (4) : $\oplus$ and $\odot$ in $\mathbb{F}_2^4$

$\mathbb{F}_2^4$ = {0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111}

($\mathbb{F}_2^4$, $\oplus$, $\odot$)

$\odot$	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111	H.W	H.D
0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000	0	0
0001	0000	0001	0000	0001	0000	0001	0000	0001	0000	0001	0000	0001	0000	0001	0000	0001	1	1
0010	0000	0000	0010	0010	0000	0000	0010	0010	0000	0000	0010	0010	0000	0000	0010	0010	1	1
0011	0000	0001	0010	0011	0000	0001	0010	0011	0000	0001	0010	0011	0000	0001	0010	0011	2	2
0100	0000	0000	0000	0000	0100	0100	0100	0100	0000	0000	0000	0000	0100	0100	0100	0100	1	1
0101	0000	0001	0000	0001	0100	0101	0100	0101	0000	0001	0000	0001	0100	0101	0100	0101	2	2
0110	0000	0000	0010	0010	0100	0100	0110	0110	0000	0000	0010	0010	0100	0100	0110	0110	2	2
0111	0000	0001	0010	0011	0100	0101	0110	0111	0000	0001	0010	0011	0100	0101	0110	0111	3	3
1000	0000	0000	0000	0000	0000	0000	0000	0000	1000	1000	1000	1000	0100	1000	1000	1000	1	1
1001	0000	0001	0000	0001	0000	0001	0000	0001	1000	1001	1000	1001	0100	1001	1000	1001	2	2
1010	0000	0000	0010	0010	0000	0000	0010	0010	1000	1000	1010	1010	0100	1001	1010	1010	2	2
1011	0000	0001	0010	0011	0000	0001	0010	0011	1000	1001	1010	1011	0100	1001	1010	1011	3	3
1100	0000	0000	0000	0000	0100	0100	0100	0100	1000	1000	1000	1000	1100	1100	1100	1100	2	2
1101	0000	0001	0000	0001	0100	0101	0100	0101	1000	1001	1000	1001	1100	1101	1100	1101	3	3
1110	0000	0000	0010	0010	0100	0100	0110	0110	1000	1000	1010	1010	1100	1100	1110	1110	3	3
1111	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111	4	4

Table (5) : Hamming distance and Hamming weight in $\mathbb{F}_2^1$

$X = (x_\circ,\ x_1,\ x_2, \ldots)$

$Y = (y_\circ,\ y_1,\ y_2, \ldots)$

$X \odot Y = (x_\circ y_\circ \oplus\ x_1 y_1 \oplus\ x_1\, y_2, \ldots)$ = number (0 or 1)

$X \oplus Y = (x_\circ \oplus\ y_\circ,\ \ x_1 \oplus y_1\ ,\ x_2 \oplus\ y_2, \ldots)$ = vector

$(\mathbb{F}_2^1 , \oplus , \odot)$

$\mathbb{F}_2^1 = \{ 0 , 1\}$

$\oplus$	0	1	Hamming Weight	Hamming distance
0	(0,0)	(0,1)	0	0
1	(1.0)	(1,1)	1	1

$\oplus$	0	1	Hamming Weight	Hamming distance
0	0	0	0	0
1	0	1	1	1

Table (6) : Hamming distance and Hamming weight in $\mathbb{F}_2^2$

$\mathbb{F}_2^2$ = { 00 , 01, 10,11}

($\mathbb{F}_2^2$, $\oplus$, $\odot$)

$\oplus$	00	01	10	11	Hamming weight	Hamming distance
00	(0,0)	(0,1)	(1,0)	(1,1)	0	0
01	(0,1)	(0,0)	(1,1)	(1,0)	1	1
10	(1,0)	(1,1)	(0,0)	(0,1)	1	1
11	(1,1)	(1,0)	(0,1)	(0,0)	2	2

($\mathbb{F}_2^2$, $\odot$)

$\odot$	00	01	10	11	Hamming weight	Hamming distance
00	0	0	0	0	0	0
01	0	1	0	1	1	1
10	0	0	1	1	1	1
11	0	1	1	0	2	2

Table (7) : Hamming distance and Hamming weight in $\mathbb{F}_2^3$

$\mathbb{F}_2^3$ = {000, 001, 010, 011, 100, 101, 110,111}

($\mathbb{F}_2^3$, $\oplus$, $\odot$)

$\oplus$	000	001	010	011	100	101	110	111	Hamming Weight	Hamming distance
000	(0,0,0)	(0,0,1)	(0,1,0)	(0,1,1)	(1,0,0)	(1,0,1)	(1,1,0)	(1,1,1)	0	0
001	(0,0,1)	(0,0,0)	(0,1,1)	(0,1,0)	(1,0,1)	(1,0,0)	(1,1,1)	(1,1,0)	1	1
010	(0,1,0)	(0,1,1)	(0,0,0)	(0,0,1)	(1,1,0)	(1,1,1)	(1,0,0)	(1,0,1)	1	1
011	(0,1,1)	(0,1,0)	(0,0,1)	(0,0,0)	(1,1,1)	(1,1,0)	(1,0,1)	(1,0,0)	2	2
100	(1,0,0)	(1,0,1)	(1,1,0)	(1,1,1)	(0,0,0)	(0,0,1)	(0,1,0)	(0,1,1)	1	1
101	(1,0,1)	(1,0,0)	(1,1,1)	(1,1,0)	(0,0,1)	(0,0,0)	(0,1,1)	(0,1,0)	2	2
110	(1,1,0)	(1,1,1)	(1,0,0)	(1,0,1)	(0,1,0)	(0,1,1)	(0,0,0)	(0,0,1)	2	2
111	(1,1,1)	(1,1,0)	(1,0,1)	(1,0,0)	(0,1,1)	(0,1,0)	(0,0,1)	(0,0,0)	3	3

$\odot$	000	001	010	011	100	101	110	111	Hamming Weight	Hamming distance
000	0	0	0	0	0	0	0	0	0	0
001	0	1	0	1	0	1	0	1	1	1
010	0	0	1	1	0	0	1	1	1	1
011	0	1	1	0	0	1	1	0	2	2
100	0	0	0	0	1	1	1	1	1	1
101	0	1	0	1	1	0	1	0	2	2
110	0	0	1	1	1	1	0	0	2	2
111	0	1	1	0	1	0	0	1	3	3

Table (8) : Hamming distance and Hamming weight in $\mathbb{F}_2^4$

$\mathbb{F}_2^4$ = {0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111}

($\mathbb{F}_2^4$, $\oplus$, $\odot$)

$\oplus$	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111	H.W	H.D
0000	0,0,0,0	0,0,0,1	0,0,1,0	0,0,1,1	0,1,0,0	0,1,0,1	0,1,1,0	0,1,1,1	1,0,0,0	1,0,0,1	1,0,1,0	1,0,1,1	1,1,0,0	1,1,0,1	1,1,1,0	1,1,1,1	0	0
0001	0,0,0,1	0,0,0,0	0,0,1,1	0,0,1,0	0,1,0,1	0,1,0,0	0,1,1,1	0,1,1,0	1,0,0,1	1,0,0,0	1,0,1,1	1,0,1,0	1,1,0,0	1,1,0,0	1,1,1,1	1,1,1,0	1	1
0010	0,0,1,0	0,0,1,1	0,0,0,0	0,0,0,1	0,1,1,0	0,1,1,1	0,1,0,0	0,1,0,1	1,0,1,0	1,0,1,1	1,0,0,0	1,0,0,1	1,1,1,0	1,1,1,1	1,1,0,0	1,1,0,1	1	1
0011	0,0,1,1	0,0,1,0	0,0,0,1	0,0,0,0	0,1,1,1	0,1,1,0	0,1,0,1	0,1,0,0	1,0,1,1	1,0,1,0	1,0,0,1	1,0,0,0	1,1,1,1	1,1,1,0	1,1,0,1	1,1,0,0	2	2
0100	0,1,0,0	0,1,0,1	0,1,1,0	0,1,1,1	0,0,0,0	0,0,0,1	0,0,1,0	0,0,1,1	1,1,0,0	1,1,0,1	1,1,1,0	1,1,1,1	1,0,0,0	1,0,0,1	1,0,1,0	1,0,1,1	1	1
0101	0,1,0,1	0,1,0,0	0,1,1,1	0,1,1,0	0,0,0,1	0,0,0,0	0,0,1,1	0,0,1,0	1,1,0,1	1,1,0,1	1,1,1,1	1,1,1,0	1,0,0,1	1,0,0,0	1,0,1,1	1,0,1,0	2	2
0110	0,1,1,0	0,1,1,1	0,1,0,0	0,1,0,1	0,0,1,0	0,0,1,1	0,0,0,0	0,0,0,1	1,1,1,0	1,1,1,1	1,1,0,0	1,1,0,1	1,0,1,0	1,0,0,1	1,0,0,0	1,0,0,1	2	2
0111	0,1,1,1	0,1,1,0	0,1,0,1	0,1,0,0	0,0,1,1	0,0,1,0	0,0,0,1	0,0,0,0	1,1,1,1	1,1,1,0	1,1,0,1	1,1,0,0	1,0,1,1	1,0,1,0	1,0,0,1	1,0,0,0	3	3
1000	1,0,0,0	1,0,0,1	1,0,1,0	1,0,1,1	1,1,0,0	1,1,0,1	1,1,1,0	1,1,1,1	0,0,0,0	0,0,0,1	0,0,1,0	0,0,1,1	0,1,0,0	0,1,0,1	0,1,1,0	0,1,1,1	1	1
1001	1,0,0,1	1,0,0,0	1,0,1,1	1,0,1,0	1,1,0,1	1,1,0,0	1,1,1,1	1,1,1,0	0,0,0,1	0,0,0,0	0,0,1,1	0,0,1,0	0,1,0,1	0,1,0,0	0,1,1,1	0,1,1,0	2	2
1010	1,0,1,0	1,0,1,1	1,0,0,0	1,0,0,1	1,1,1,0	1,1,1,1	1,1,0,0	1,1,0,1	0,0,1,0	0,0,1,1	0,0,0,0	0,0,0,1	0,1,1,0	0,1,1,1	0,1,0,0	0,1,0,1	2	2
1011	1,0,1,1	1,0,1,0	1,0,0,1	1,0,0,0	1,1,1,1	1,1,1,0	1,1,0,1	1,1,0,0	0,0,1,1	0,0,1,0	0,0,0,1	0,1,0,0	0,1,1,1	0,1,1,0	0,1,0,1	0,1,0,0	3	3
1100	1,1,0,0	1,1,0,1	1,1,1,0	1,1,1,1	1,0,0,0	1,0,0,1	1,0,1,0	1,0,1,1	0,1,0,0	0,1,0,1	0,1,1,0	0,1,1,1	0,0,0,0	0,0,0,1	0,0,1,0	0,0,1,1	2	2
1101	1,1,0,1	1,1,0,0	1,1,1,1	1,1,1,0	1,0,0,1	1,0,0,0	1,0,1,1	1,0,1,0	0,1,0,1	0,1,0,0	0,1,1,1	0,1,1,0	0,0,0,1	0,0,0,0	0,0,1,1	0,0,1,0	3	3
1110	1,1,1,0	1,1,1,1	1,1,0,0	1,1,0,1	1,0,1,0	1,0,1,1	1,0,0,0	1,0,0,1	0,1,1,0	0,1,1,1	0,1,0,0	0,1,0,1	0,0,1,0	0,0,1,1	0,0,0,0	0,0,0,1	3	3
1111	1,1,1,1	1,1,1,0	1,1,0,1	1,1,0,0	1,0,1,1	1,0,1,0	1,0,0,1	1,0,0,0	0,1,1,1	0,1,1,0	0,1,0,1	0,1,0,0	0,0,1,1	0,0,1,0	0,0,0,1	0,0,0,0	4	4

Table (8) : Hamming distance and Hamming weight in $\mathbb{F}_2^4$

$\mathbb{F}_2^4$ = {0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111}

($\mathbb{F}_2^4$, $\oplus$, $\odot$)

$\odot$	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111	H.W	H.D
0000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0001	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	1	1
0010	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	1	1
0011	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	2	2
0100	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	1	1
0101	0	1	0	1	1	0	1	0	0	1	0	1	1	0	1	0	2	2
0110	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	2	2
0111	0	1	1	0	1	0	0	1	0	1	1	0	1	0	0	1	3	3
1000	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
1001	0	1	0	1	0	1	0	1	1	0	1	0	1	0	1	0	2	2
1010	0	0	1	1	0	0	1	1	1	1	0	0	1	1	0	0	2	2
1011	0	1	1	0	0	1	1	0	1	0	0	1	1	0	0	1	3	3
1100	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	2	2
1101	0	1	0	1	1	0	1	0	1	0	1	0	0	1	0	1	3	3
1110	0	0	1	1	1	1	0	0	1	1	0	0	0	0	1	1	3	3
1111	0	1	1	0	1	0	0	1	1	0	0	1	0	1	1	0	4	4