Universitat Autònoma de Barcelona

Departament d'Enginyeria de la Informació i de les Comunicacions

Nonlinear codes: representation, constructions, minimum distance computation and decoding

Submitted to Universitat Autònoma de Barcelona in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science

by Fanxuan Zeng

Cerdanyola del Vallès, June 2014

Advisors: Dr. Mercè Villanueva and Dr. Jaume Pujol

Professors at Universitat Autònoma de Barcelona

I certify that I have read this thesis entitled “Nonlinear codes: representation, constructions, minimum distance computation and decoding” and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Cerdanyola del Vallès, June 2014

Dr. Mercè Villanueva
(Advisor)

Dr. Jaume Pujol
(Advisor)

To Xiaojuan Lian

Abstract

Coding theory deals with the design of error-correcting codes for the reliable transmission of information across noisy channels. An error-correcting code (or code) is a process, which consists on expressing a sequence of elements over an alphabet in such a way that any introduced error can be detected and corrected (with limitation), and it is based on adding redundancy elements. This process includes encoding, transmitting and decoding the sequence of elements. Most of the used codes are block codes and most of them have a linear structure, which facilitates the process of encoding and decoding. In this dissertation, nonlinear error-correcting codes are studied. Despite nonlinear codes do not have the same good properties for encoding and decoding as linear ones, they have interest because some of best codes are nonlinear.

The first question that arises when we use nonlinear codes is their representation. Linear codes can be represented by using a generator or parity-check matrix. The best way to represent a nonlinear code is by using the kernel/coset representation, which allows us to represent it through some representative codewords instead of all codewords. In this dissertation, this representation is studied and efficient algorithms to compute the kernel and coset representatives from the list of codewords are given. In addition, properties such as equality, inclusion, intersection and union between nonlinear codes are given in terms of this representation. Also, some well known code constructions (extended, punctured,...) are described by manipulating directly the kernel and coset representatives of the constituent nonlinear codes.

In order to identify a code (linear or nonlinear), the length n, number of codewords M and minimum distance d are the most important parameters. The length n and size M are comparatively easy to compute. On the other hand, to determine the minimum distance of a code is not so easy. As a matter of fact, it has been proven to be an NP-hard problem [37]. However, some algorithms have been developed to compute the minimum distance for linear codes using different approaches: Gröbner bases [7], tree structure [25], probabilistic algorithms [13, 23] and vector enumeration [39]. For nonlinear codes, except for some special families, no general algorithms have been developed to compute their minimum distance. Using the kernel/coset representation and the Brouwer-Zimmermann's algorithm to compute the minimum distance for linear codes, new algorithms to compute the minimum distance for nonlinear codes are described.

The hardest problem in the process of transmitting information is decoding. For linear codes, a general decoding algorithm is the syndrome decoding. However, there is not any general decoding method for nonlinear codes. Based on the kernel/coset representation and the minimum distance computation, new general algorithms to decode linear and nonlinear codes are proposed. For some linear codes (codes with a big codimension), the proposed algorithms have better performance than the syndrome decoding algorithm. For nonlinear codes, this is the first general method for decoding, which is comparable to syndrome decoding for linear ones.

Finally, most of these algorithms have been evaluated using the Magma software, and a new Magma package to deal with binary nonlinear codes has been developed, based in the results given in this dissertation.

Resum

La teoria de codis estudia el disseny de codis correctors d'errors per a la transmisió fidedigne d'informació per un canal amb soroll. Un codi corrector d'errors (o simplement codi) és un procés que consisteix en expressar una seqüència d'elements sobre un alfabet de tal manera que qualsevol error que sigui introduït pot ser detactat i corregit (amb limitacions), i està basat en la tècnica d'afegir elements redundants. Aquest procés inclou la codificació, la transmisió i la descodificació de la seqüència d'elements. La majoria dels codis utilitzat són codis bloc i la majoria d'ells tenen una estructura lineal, que facilita el procés de codificació i descodificació.

En aquesta memòria, estudiarem codis correctors d'errors no lineals. Malgrat els codis no lineals no tenen les mateixes bones propietats per codificar i descodificar com els lineals, el codis no lineals tenen interès atès que alguns dels millors codis no són lineals.

La primera qüestió que apareix quan s'utilitzen codis no lineals és la seva representació. Els codis lineals poden ser representats utilitzant una matriu generadora o una matriu de control. La millor manera de representar un codi no lineal és utilitzar la representació *kernel/coset*, que permet representar un codi mitjançant unes quantes paraules-codi representatives del codi en lloc de totes les paraules-codi. En aquesta memòria, s'estudia aquesta representació i es proposen algorismes eficients per calcular el *kernel* i els representants dels *cosets* a partir de la llista de totes les paraules-codi. A més, s'utilitza aquesta representació per estudiar algunes propietats com la igualtat, inclusió, intersecció i unió de codis no lineals. També són analitzades algunes construccions ben conegudes (codi estès, codi *punctured*,...) manipulant directament el *kernel* i els representants dels *cosets* dels codis no lineals que en formen part.

Per identificar un codi (lineal o no lineal), els paràmetres més importants són la longitud n, el nombre de paraules-codi M i la distància mínima d. La longitud n i la mida M són comparativament fàcils de calcular. Però, calcular la distància mínima d'un codi no és tant fàcil. De fet, s'ha demostrat

que és un problema NP-*hard* [37]. No obstant, alguns algorismes han estat desenvolupats per calcular la distància mínima dels codis lineals utilitzant diversos mètodes: bases de Gröbner [7], arbres [25], algorismes probabilístics [13, 23] i enumeració de vectors [39]. Però, per als codis no lineals, excepte per a algunes famílies, no s'han desenvolupat algorismes generals per calcular la distància mínima. Utilitzant la representació *kernel/coset* i l'algorisme Brouwer-Zimmermann per calcular la distància mínima dels codis lineals, s'han descrit nous algorismes per calcular la distància mínima dels codis no lineals.

El problema més costós en el procés de transmisió de la informació és la descodificació. Per als codis lineals, la descodificació via síndrome és el mètode més general. No obstant, no existeix un algorisme de descodificació general per a codis no lineals. Basats en la representació *kernel/coset* i en el càlcul de la distància mínima, es proposen nous algorismes per descodificar codis lineals i no lineals. Per a alguns codis lineals (codis amb una codimensió gran) els algorismes proposats tenen un comportament millor que la descodificació via síndrome. Per als codis no lineals, és el primer mètode general de descodificació proposat comparable amb la descodificació via síndrome per a codis lineals.

Finalment, la majoria d'aquests algorismes han estat avaluats usant el sistema MAGMA i s'ha desenvolupat un nou mòdul de MAGMA basat en els resultats d'aquesta tesis.

Acknowledgements

Firstly, I would like to thank my advisers Dr. Mercè Villanueva and Dr. Jaume Pujol for the patience and effort to help me in the past three years. They not only helped me to find my way to coding theory, but also helped me to make the process enjoyable. I would also thank my wife Xiaojuan Lian for the love and support.

During the three years in dEIC group at UAB, I received many help and support, I earned not only knowledge, but also enjoyment and wisdom. Thanks a lot to everyone in the group.

Contents

List of Figures

Chapter 1

Introduction

Coding theory is focused on the study of methods for efficient and accurate transfer of information from one place to another. This theory has been developed for the needs of minimizing noises during the transfer of information, which includes compact disc recordings, the transmission of information across telephone lines, data transfer from one computer to another or from memory to CPU, and information transmission from a distant source such as a weather or communication satellites.

The physical medium through which the information is transmitted is called a *channel*. Telephone lines and the atmosphere are examples of channels. Undesirable disturbances, called *noise*, may cause the received information to differ from what was transmitted. Noise can be caused by sunspots, lightning, folds in a magnetic tape, competing telephone messages, poor typing, poor hearing, or many other things.

Coding theory deals with the problem of detecting and correcting transmission errors caused by noise on the channel. The fundamental problem in coding theory is to determine which message was sent based on what is received. Figure 1.1 provides a rough idea of a general information transmission system.

Unlike *source coding*, which deals with data compression or bit-rate reduction and involves encoding information using fewer bits than the original representation, *channel coding* is focused on applying techniques to control the errors during the transmission process. These techniques are studied within the coding theory. The most important part of the diagram, as far as we are concerned, is the noise, for without it there would be no need for the development of the coding theory. In practice, the control we have over this noise is the choice of a good channel to use for transmission and the use

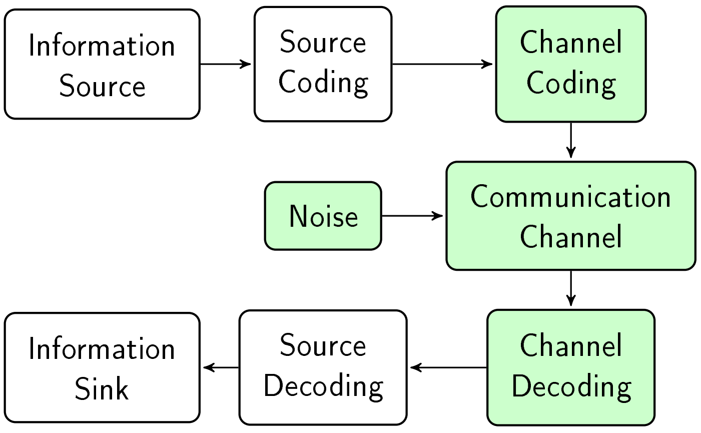


Figure 1.1: A general information transmission system.

of various noise filters to combat certain types of interference which may be encountered. These are engineering problems and will not be discussed in this dissertation.

In many cases, the information to be sent is transmitted by a sequence of 1's and 0's. We call any of the two values 0 or 1 a *bit*. A *word* is a sequence of bits. The *length* of a word is the number of bits in the word. For example, 0110101 is a word of length seven. A word is transmitted by sending its bits, one after another, across a *binary channel*. The term "binary" refers to the fact that only 0 and 1 are used in the channel.

In a *perfect*, or noiseless, channel, the bit sent is always the bit received. But unfortunately (or fortunately) no channel is perfect. It is just the matter of some channels are less noisy, or more reliable, than others. A binary channel is called *symmetric* if 0 and 1 are transmitted with equal accuracy; that is, the probability of receiving the correct bit is independent of which bit, 0 or 1, is being transmitted. The *reliability* of a binary symmetric channel (BSC) is a real number p, $0 \leq p \leq 1$, where p is the probability that the bit sent is the bit received. Any channel with $0 < p \leq 1/2$ can be easily converted into a channel with $1/2 \leq p < 1$. Hence forth we will always assume that we are using a BSC with probability p satisfying $1/2 \leq p < 1$. If p is the probability that the bit received is the same as the bit sent, then $1 - p$ is the probability that the bit received is not the same as the bit sent. Figure 1.2 may clarify how a BSC operates. In most cases, it may be hard to estimate the actual value of p for a given channel. However, the actual value of p does not influence significantly the development of coding techniques

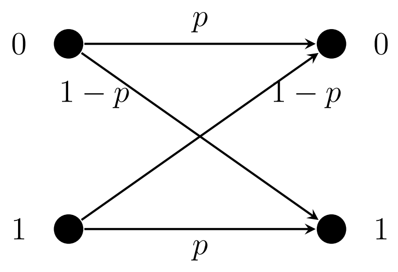


Figure 1.2: A binary symmetric channel (BSC).

within the coding theory.

The BSC is a special case of a general, discrete input, discrete output channel. In practice, the information may contain more than two characters. Generally, an input to the channel is a symbol from a set $X = \{x_0, x_1, \ldots, x_{q-1}\}$ with q elements, and an output at the receiving end of the channel consist of a symbol from a set $Y = \{y_0, y_1, \ldots, y_{Q-1}\}$ with Q elements. We assume that the channel is memoryless. This channel is known as a discrete memoryless channel (DMC) and is represented as shown in Figure 1.3. The inputs and the outputs can then be related by a set of qQ conditional probabilities

$$P(Y = y_i \mid X = x_j) = P(y_i|x_j),$$

where $i \in \{0, 1, \ldots, Q-1\}$ and $j \in \{0, 1, \ldots, q-1\}$. The conditional probability $P(y_i|x_j)$ is defined as the *channel transition probability* and is denoted by p_{ji}. The set of conditional probabilities $\{P(y_i|x_j)\}$ that characterises a DMC can be arranged as a matrix of the form $\Pi = (p_{ji})$, which is called the *probability transition matrix* [8].

In order to transmit an information through a channel, a code must be chosen according to the channel. For example, in order to transfer a paragraph in English, $26 + 10$ characters are needed. Other languages normally need more characters. For example, more than 2500 characters are needed for the daily used Chinese language. In order to represent all languages in a computer, a Unicode system is build to handling the text expressed in the most of the world's writing systems. The latest version of Unicode contains a collection of more than 110000 characters with each of them represented by a 16-bit code unit to be transmitted using the binary channel of computer system.

A *code* C is a set of words in a certain alphabet. A *block code* is a code

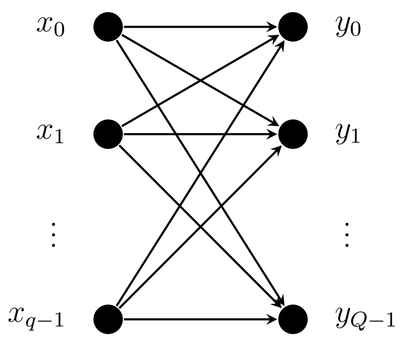


Figure 1.3: A discrete memoryless channel (DMC).

C having all its words of the same length; this number is called the *length* of the code. For example, the Unicode system is a binary block code of length 16. From now on, we will only consider block codes. Therefore, for us, the term code will always mean a block code. The words belonging to a given code C, will be called *codewords*. We shall denote the number of codewords in a code C by $M = |C|$.

In order to transmit a message through a channel, the message needs to be *encoded* at the sending end and *decoded* at the receiving end. For example, if we want to transmit a simple sentence “Hi!” using a Unicode system, we need to encode “H” to 0048, “i” to 0069, and “!” to $FF01$; transfer it through a hexadecimal channel; and luckily, if no errors occur during the transmission, we obtain $00480069FF01$. By dividing it into 3 blocks of length 4, and by looking into the Unicode table, we can decode it back as sent. If one error occurs and $00480065FF01$ is received (with an error on the 8th entry), the decode result would be "He!".

To correct possible errors, some redundancies have to be included in every codeword in such a way that all codewords of the code are separated as much as possible. Let c denote the codeword sent through the channel and w the received word. Then, decode w means to find the codeword c' closest to w. For example, if we have two messages, 0 and 1, encoded as 000 and 111, respectively; and we send $c = 000$ through a BSC, the receiver obtains one of the following sequences: 000, 001, 010, 100, 011, 101, 110, or 111. If $w = 001$ is received, we decode w as 000 because it would be the codeword more similar to w. If $w = 110$ is received, 111 would be the most similar one.

The concept of *similarity* can be defined according to a metric. The *Hamming distance* between two codewords is the number of coordinates where they differ. The smallest Hamming distance between any two distinct codewords of a code is called the *minimum distance* of the code and is denoted by d. The minimum distance, together with the length and the number of codewords of a code, are the three most important parameters of a code. The *Hamming weight* of a codeword is the number of its nonzero coordinates and the *minimum weight* of a code is the smallest weight of all its nonzero codewords. The computation of the minimum weight and minimum distance of a code is a fundamental topic in coding theory.

For more information on the topic on which this dissertation is based, refer to the books [18, 19, 24]. More specifically, we have mainly used reference [18] for the brief introduction to this topic given in this chapter.

Among all kinds of codes, *linear codes* are studied the most for their linearity, which helps to simplify their representation using a generator matrix and to compute their parameters (length, number of codewords and minimum distance). Moreover, for linear codes, the minimum distance coincides with the minimum weight, which is easier to compute. However, some nonlinear codes are better than any linear code with the same parameters. For example, Kerdock and Preparata codes are binary nonlinear codes that contain more codewords than any comparable binary linear code presently known [29, 21].

Due to the lack of linearity, the representation of nonlinear codes is usually hard. A first solution would be to find whether they have another structure or not. For example, there are binary nonlinear codes which have a $\mathbb{Z}_4$-linear or $\mathbb{Z}_2\mathbb{Z}_4$-linear structure and, therefore, they can also be compactly represented using a quaternary generator matrix [6, 16]. In general, nonlinear codes without any known structure can also be compactly represented. Specifically, they can be seen as a union of cosets of a linear subcode of the code [1, 28]. This allows us to represent a code as a set of representative codewords instead of as a set with all codewords. Moreover, these representative codewords can be organized as a matrix, called *parity-check system* [17], which is a generalization of the parity-check matrix for linear codes [24].

The main objectives of this dissertation are the following:

- The first objective is to describe and analyze the complexity of some algorithms to compute the kernel and coset representatives, in order to represent nonlinear codes in a more efficient way.

- The second objective is to establish some properties and describe some constructions of new codes from given ones in terms of the kernel and coset representatives.

- The third objective is to propose and analyze algorithms to compute the minimum weight and minimum distance of nonlinear codes, based on the coset structure and the known Brouwer-Zimmermann's algorithm for linear codes.

- The forth objective is to design new general decoding algorithms for linear codes based on the previous algorithms, and then generalize these decoding methods into nonlinear codes using also the coset structure.

- The fifth objective is to implement functions to construct nonlinear codes, compute their minimum weight and minimum distance, and simulate the decoding process, using the results and algorithms given in the previous chapters. This will mean to extend the functionality of a new package in Magma to work with binary nonlinear codes.

All results in this dissertation are written in such a way that they can be easily transformed into algorithms. The performance of these algorithms is evaluated and an estimation of the number of enumerated codewords needed in the computations is given.

The overview of this dissertation is the following:

- Chapter 2 exposes definitions and known results on coding theory. Specifically, about the construction of new linear codes from old, the computation of the minimum weight of linear codes, and the syndrome decoding for linear codes. These results will be generalized into nonlinear codes in later chapters.

- Chapter 3 describe and analyze algorithms to compute the kernel and coset representatives of nonlinear codes in an efficient way. Then, it generalizes those known constructions given in Chapter 2 for linear codes into nonlinear codes, in terms of the kernel and coset representatives.

- Chapter 4 describes and analyzes algorithms to compute the minimum weight and minimum distance of nonlinear codes. The performance of these algorithms is studied by giving an estimation of the number of enumerated codewords needed in the computations.

- Chapter 5 describes new general decoding methods for linear codes based on the algorithms described in Chapter 4, and then generalizes these methods to decode nonlinear codes.
- Chapter 6 gives an introduction to a Magma package to deal with nonlinear codes, describing the main functions implemented to construct nonlinear codes, compute their minimum weight and minimum distance, and simulate the decoding process, using the algorithms discussed in the previous chapters.
- Chapter 7 exposes our conclusions and proposes some future lines of research on this topic.

Finally, mention that all the work included in this dissertation has been presented to several international conferences [31, 32, 33, 35], published in the proceedings of two of these conferences [33, 35], and submitted to a journal [36]. Moreover, all this work has been partially supported by the Spanish MICINN under Grants TIN2010-17358 and TIN2013-40524-P, and by the Catalan AGAUR under Grants 2009SGR-1224 and 2014SGR-691.

Chapter 2

Introduction to coding theory

Among all kinds of codes, linear codes are the most studied, since their linearity property allows to simplify their representation using a generator matrix and to compute their parameters (length, number of codewords and minimum distance) in an efficient way. Moreover, it allows to describe efficient methods to construct new linear codes from old, and to encode and decode them. On the other hand, nonlinear codes are less studied because of their lack of linearity, although some nonlinear codes are better than any linear code with the same parameters.

In this chapter, some of the basic definitions and properties of linear codes are presented. All these properties are then generalized into nonlinear codes in the later chapters. For more information on these results on linear codes, refer to [18, 24, 2, 28, 19, 39]. More specifically, we have mainly used reference [18] for Sections 2.3 and 2.4, and reference [39] for Section 2.7.

2.1 Finite fields

A *field* is an algebraic structure consisting of a set together with two operations, usually called addition (denoted by $+$) and multiplication (denoted by $\cdot$ but often omitted), which satisfy certain axioms [3, 12]. A *finite field* is a field with a finite order (i.e. a finite number of elements), also called a *Galois field*. The order of a finite field is always a prime number or a power of a prime number [3, 12]. For each prime power, p^n ($n \geq 1$, and p prime), there exists exactly one (up to an isomorphism) finite field with p^n elements, denoted by $GF(p^n)$ or $\mathbb{F}_{p^n}$.

The simplest finite fields are the prime fields $GF(p)$, where p is a prime number. The finite field $GF(p)$ is isomorphic to the finite set of integers modulo p, denoted by $\mathbb{Z}_p$. In other words, the set of integers modulo p forms a field of order p, denoted by $GF(p)$ or $\mathbb{F}_p$. Note that $a = b$ in $GF(p)$ means the same as $a \equiv b \pmod{p}$. The elements of $GF(p)$ are $\{0, 1, 2, \ldots, p-1\}$, and the operations $+$ and $\cdot$ are carried out mod p. For example, $\mathbb{F}_2$ is the binary field $\mathbb{F}_2 = \{0, 1\}$; $\mathbb{F}_3$ is the ternary field $\mathbb{F}_3 = \{0, 1, 2\}$, where for example, $1+2 = 3 \equiv 0 \pmod 3$, $2 \cdot 2 = 4 \equiv 1 \pmod 3$, $1-2 = -1 \equiv 2 \pmod 3$, etc [24]. Note that $2 \cdot 2 \equiv 0 \pmod 4$ in the ring of integers modulo 4, so 2 has no inverse, and the ring of integers modulo 4, denoted by $\mathbb{Z}_4$, is not a finite field with 4 elements.

The finite field $GF(p^n)$, $n > 1$, p prime, can be defined in the following way. Let $\mathbb{F}_p[x]$ be the set of polynomials in the indeterminate x with coefficients in $\mathbb{F}_p$. This set, along with the usual addition and multiplication of polynomials, forms a commutative ring with unity. Recall that a polynomial $d(x) \in \mathbb{F}_p[x]$ is a divisor or factor of $f(x) \in \mathbb{F}_p[x]$ if there is $g(x) \in \mathbb{F}_p[x]$ such that $f(x) = g(x)d(x)$. Note that $f(x)$ and 1 are trivial divisors of $f(x)$, and any other divisor is said to be a nontrivial or proper divisor of $f(x)$. A nonconstant polynomial $f(x) \in \mathbb{F}_p[x]$ is said to be *irreducible* over $\mathbb{F}_p$ if it has no proper divisors in $\mathbb{F}_p[x]$; otherwise it is said to be *reducible* over $\mathbb{F}_p$. To construct a field of characteristic p, we begin with a irreducible polynomial $f(x) \in \mathbb{F}_p[x]$. If $f(x)$ has degree n, then the residue class ring $\mathbb{F}_p[x]/(f(x))$ is a finite field with p^n elements, that is $GF(p^n)$. An element $\alpha \in GF(p^n)$ is *primitive* if $\alpha^m \neq 1$ for all $1 \leq m \leq p^n - 2$. Equivalently, α is primitive if every nonzero element in $GF(p^n)$ can be expressed as a power of α. For example, a finite field with 4 elements, denoted by $\mathbb{F}_4$, can be constructed as $\mathbb{F}_4 = \mathbb{F}_2[x]/(1 + x + x^2)$, where $f(x) = 1 + x + x^2$ is irreducible. Therefore, $\mathbb{F}_4 = \{[0], [1], [x], [1+x]\}$, where $[g(x)]$ denotes the residue class of $g(x)$. Since $\alpha = [x]$ is primitive, we can also write $\mathbb{F}_4 = \{0, 1, \alpha, \alpha^2\}$, where $1 + \alpha = \alpha^2$.

In this dissertation, we will study codes whose alphabet is a finite field $\mathbb{F}_q$ with q elements ($q = p^n$, $n \geq 1$, and p prime). Although we will present our general results over arbitrary fields, we will often specialize to the field $\mathbb{F}_2$ with two elements and $\mathbb{F}_3$ with three elements.

2.2 Codes over finite fields

Let $\mathbb{F}_q^n$ denote the vector space that consists of all vectors of length n over the finite field $\mathbb{F}_q$. We will sometimes write the vectors $(a_1, a_2, \ldots, a_n) \in \mathbb{F}_q^n$ in the form $a_1 a_2 a_3 \cdots a_n$. An (n, M) *code* C over $\mathbb{F}_q$ is a subset of $\mathbb{F}_q^n$ of size M. The vectors that belong to C are called *codewords*, and n is referred as the *length* of the code.

The field $\mathbb{F}_2$ has had a very special place in the history of coding theory, and codes over $\mathbb{F}_2$ are called *binary codes.* Similarly, codes over $\mathbb{F}_3$ are termed *ternary codes*, while codes over $\mathbb{F}_4$ are called *quaternary codes.* The term "quaternary" has also been used to refer to codes over the ring of integers modulo 4 [19, 16, 38]. In general, codes over $\mathbb{F}_q$ are also called *q-ary codes.*

A q-ary code C is called to be an $[n, k]$ *linear* code over $\mathbb{F}_q$ if C is a k-dimensional subspace of $\mathbb{F}_q^n$. An $[n, k]$ linear code C over $\mathbb{F}_q$ has $M = q^k$ codewords. Because of their algebraic structure, they are easier to describe, encode, and decode than nonlinear codes. The code alphabet for linear codes is usually a finite field, although sometimes other algebraic structures (such as the integers modulo 4) can be used to define codes that are also called "linear" [19, 16, 38]. The two most common ways to represent a linear code are with either a generator matrix or a parity check matrix.

A *generator matrix* for an $[n, k]$ linear code C is any $k \times n$ matrix G whose rows forms a basis of C. In general, there are many generator matrices for a linear code C. For any set of k linear independent column vectors of a generator matrix G, the corresponding set of coordinate positions forms an *information set* for C. The remaining $r = n - k$ coordinate positions are termed a *redundancy set* and r is called the *redundancy* of C. If the first k coordinate positions form an information set, the linear code has an unique generator matrix of the form $(I_k|A)$, where I_k is the $k \times k$ identity matrix and A is a $k \times (n - k)$ matrix. Such a generator matrix is said to be in *standard form.* A generator matrix G is *systematic* if among its columns can be found the columns of a $k \times k$ identity matrix, in which case G is said to be systematic on those columns or the corresponding coordinate positions. Note that a standard generator matrix is a special type of systematic generator matrix.

Example 2.2.1. *The code $C_1 = \{00000, 11000, 01111, 10111\}$ is a $[5, 2]$ binary linear code with 4 codewords. Since it is linear and $4 = 2^2$, it is easy to see that it is a subspace of $\mathbb{F}_2^5$ of dimension 2. For example, G_1 is a generator matrix for the code C_1, and G_1' is the generator matrix in standard form, where*

$$G_1 = \begin{pmatrix} 11000 \\ 01111 \end{pmatrix} \quad \textit{and} \quad G_1' = \begin{pmatrix} 10111 \\ 01111 \end{pmatrix}.$$

The first two coordinate positions form an information set for C_1 and G_1' is said to be systematic on the first two columns. The last three coordinate positions form a redundancy set for C_1, so the redundancy of C_1 is 3.

Example 2.2.2. *The code* $C_2 = \{0000, \alpha^2 0\alpha\alpha,\ \alpha\alpha 00,\ 1\alpha\alpha\alpha,\ \alpha^2 111,\ 1\alpha^2 11,\ \alpha 011,\ 10\alpha^2\alpha^2,\ \alpha 1\alpha\alpha,\ 0\alpha 11,\ 0\alpha^2\alpha\alpha,\ \alpha^2\alpha\alpha^2\alpha^2,\ 01\alpha^2\alpha^2, 1100,\ \alpha\alpha^2\alpha^2\alpha^2,\ \alpha^2\alpha^2 00\}$ *over* $\mathbb{F}_4 = \mathbb{F}_2[x]/(x^2+x+1)$ *is a* $[4,2]$ *quaternary linear code with* 16 *codewords, where* $\alpha = [x]$. *Since it is linear and* $16 = 4^2$*, it is easy to see that it is a subspace of* $\mathbb{F}_4^4$ *of dimension* 2. *For example,* G_2 *is a generator matrix for the code* C_2*, and* G_2' *is the generator matrix in standard form, where*

$$G_2 = \begin{pmatrix} 1 & 0 & \alpha^2 & \alpha^2 \\ 1 & 1 & 0 & 0 \end{pmatrix} \quad \text{and} \quad G_2' = \begin{pmatrix} 1 & 0 & \alpha^2 & \alpha^2 \\ 0 & 1 & \alpha^2 & \alpha^2 \end{pmatrix}.$$

The first two coordinate positions form an information set for C_2 *and* G_2' *is said to be systematic on the first two columns. The last three coordinate positions form a redundancy set for* C_2*, so the redundancy of* C_2 *is* 2.

Let C be any code over $\mathbb{F}_q$. The *dual code* of C, denoted by $C^\perp$, is the code

$$C^\perp = \{x \in \mathbb{F}_q^n \mid x \cdot c = 0, \text{ for all } c \in C\},$$

where $x \cdot c$ denote the ordinary inner product between the two vectors $x = (x_1, x_2, \dots, x_n)$ and $c = (c_1, c_2, \dots, c_n)$, that is

$$x \cdot c = \sum_{i=1}^{n} x_i c_i.$$

Note that $C^\perp$ is a linear code even if C is nonlinear. In case that C is an $[n,k]$ linear code over $\mathbb{F}_q$, then $C^\perp$ is an $[n, n-k]$ linear code. Moreover, it is easy to see that a generator matrix H of $C^\perp$ is an $(n-k)\times n$ matrix, which is called a *parity check matrix* for the $[n,k]$ linear code C. In general, there are also several possible parity check matrices for C. If C has a generator matrix in standard form $(I_k|A)$, then a parity check matrix for C can be computed as $(-A^T|I_{n-k})$. Using a parity check matrix H, we can also see the $[n,k]$ linear code C as

$$C = \{x \in \mathbb{F}_q^n \mid Hx^T = \mathbf{0}\},$$

where $\mathbf{0}$ is the all-zero vector.

Example 2.2.3. *Let* C_1 *be the same binary linear code given in Example 2.2.1. Let*

$$H_1 = \begin{pmatrix} 11001 \\ 00101 \\ 00011 \end{pmatrix} \quad \text{and} \quad H_1' = \begin{pmatrix} 11100 \\ 11010 \\ 11001 \end{pmatrix}.$$

The matrix H_1 *is a parity check matrix for the code* C_1 *and a generator matrix for the dual code* $C_1^\perp$*, which is a* $[5,3]$ *linear code. Since* G_1' *is in standard form, another parity check matrix* H_1' *for* C_1 *can be computed from* G_1'.

Example 2.2.4. *Let C_2 be the same quaternary linear code given in Example 2.2.2. Let*

$$H_2 = \begin{pmatrix} 1 & 1 & 0 & \alpha \\ 0 & 0 & 1 & 1 \end{pmatrix} \quad \text{and} \quad H_2' = \begin{pmatrix} \alpha^2 & \alpha^2 & 1 & 0 \\ \alpha^2 & \alpha^2 & 0 & 1 \end{pmatrix}.$$

The matrix H_2 is a parity check matrix for the code C_2 and a generator matrix for the dual code $C_2^\perp$, which is a $[4, 2]$ linear code. Since G_2' is in standard form, another parity check matrix H_2' for C_2 can be computed from G_2'.

Let us assume we are at the receiving end of a channel and we want to receive a message sent from the transmitter at the other end. The transmitter is, of course, one we have ourselves previously designed. There are two quantities over which we have no control. The first one is the probability p that our channel will transmit a symbol correctly. In case we consider a BSC, it would be the reliability p, $1/2 \leq p < 1$. The second one is the number of possible messages M that might be transmitted. The actual messages are not as important as the number of possible messages. The two main problems on coding theory are the following:

- *Encoding:* In the process of encoding, we have to determine a code over $\mathbb{F}_q$ to use for sending our messages. We must make some choices. If we only consider linear codes, first of all, we select a positive integer k such that $M \leq q^k$. Next, we decide how many coordinates we need to add to each information vector of length k to ensure that as many errors can be corrected or detected as we require; this is the choice of the codewords and the length of the code, n. To transmit a particular message, the transmitter finds the information vector of length k assigned to that message, then transmits the codeword of length n corresponding to that information vector of length k.

  If we use an $[n, k]$ linear code C over $\mathbb{F}_q$, we can encode using the generator matrix G of C by mapping an information vector $x \in \mathbb{F}_q^k$ to the codeword $xG \in \mathbb{F}_q^n$. By matrix multiplication, xG is a codeword consisting of a linear combination of the rows of G. If G is a standard generator matrix, then the first k coordinates of the transmitted codeword xG contain the information vector x.

- *Decoding:* The process of decoding, that is, determining which codeword c (and thus which message) was sent when a vector w is received, is more complex. Finding efficient (fast) decoding algorithms is a major area of research in coding theory because of their practical applications.

H	101000	e	000101	o	001111	W	110111	r	010010
l	001100	d	000100		111011	!	111110	,	111100

Table 2.1: A (partial) map to encode English letters into binary vectors.

Example 2.2.5. *In the case we want to transmit an English sentence, considering lowercase and uppercase letters, at least* 52 *messages to represent all letters are needed. Moreover, if we want to transmit it through a binary channel, since* $2^5 < 52 < 2^6$*, then* $k = 6$ *is chosen and some extra messages can be assigned to punctuations. Table 2.1 shows the corresponding vectors for some letters, from a complete map between English letters and binary vectors of length 6. Then, for example, the sentence "Hello, World!" is transformed into the vectors:* 101000, 000101, 001100, 001100, 001111, 111100, 111011, 110111, 001111, 010010, 001100, 000100, 111110.

To correct possible errors during the transmission, we can add two extra coordinates to each vector of length $k = 6$*, obtaining vectors of length* $n = 8$*, which will be the codewords of the code. Let* C_3 *be the binary linear code of length* 8 *and dimension* 6*, given by the generator matrix*

$$G_3 = \begin{pmatrix} 10000010 \\ 01000011 \\ 00100010 \\ 00010010 \\ 00001011 \\ 00000111 \end{pmatrix}.$$

An information vector of length 6 can be encoded into a vector of length 8 by multiply it by the generator matrix G_3*. For example,* $(101000) \cdot G_3 = (10100000)$ *and* $(111011) \cdot G_3 = (11101111)$*. As* G_3 *is in standard form, the first* 6 *coordinates of the resulting vector are exactly the ones of the information vector.*

The encoded vectors are then transmitted through the channel. If luckily, no errors occur during the transmission, we obtain the same sent vectors, and the original information can be treated as the first 6 *coordinates of the received vectors. However, a real channel is always noisy, and a decoding process is needed. A general decoding method for linear codes is the syndrome decoding, which can be found described in Section 2.4.*

If we choose to transmit an English sentence through a ternary channel, since $3^3 < 52 < 3^4$*, then only* 4 *coordinates are needed, so we take* $k = 4$*.*

H	1002	e	2010	o	1000	W	0021	r	2121
l	1111	d	2211		1221	!	2112	,	1122

Table 2.2: A (partial) map to encode English letters into ternary vectors.

Table 2.2 shows the corresponding vectors for some letters, from a complete map between English letters and ternary vectors of length 4. According to the table, "Hello, World!" is transformed into the vectors: 1002, 2010, 1111, 1111, 1000, 1122, 1221, 0021, 1000, 2121, 1111, 2211, 2112.

Again, to correct possible errors during the transmission, we add two extra coordinates to each vector of length $k = 4$*, obtaining vectors of length* $n = 6$*. Let* C_4 *be the ternary linear code of length* 6 *and dimension* 4*, given by the generator matrix*

$$G_4 = \begin{pmatrix} 100010 \\ 010020 \\ 001020 \\ 000112 \end{pmatrix}.$$

An information vector of length 4 can be encoded into a vector of length 6 by multiply it by the generator matrix G_4*. For example,* $(1002) \cdot G_3 = (10021)$ *and* $(1111) \cdot G_3 = (111102)$*. As* G_3 *is in standard form, the first* 4 *coordinates of the resulting vector are exactly the ones of the information vector. The decoding process can be found described in Section 2.4.*

When a vector w is received at the output side of a channel, we have to determine which codeword c was sent. Among all possible sent codewords, we choose the most "similar" codeword to w. The concept of similarity is defined according to a metric. The *Hamming distance* $d_H(u, v)$, or simply the distance, between two vectors $u, v \in \mathbb{F}_q^n$ is the number of coordinates in which u and v differ. The *Hamming weight* $wt_H(u)$, or simply the weight, of a vector $u \in \mathbb{F}_q^n$ is the number of coordinates which are nonzero, or equivalent $wt_H(u) = d_H(u, \mathbf{0})$, where $\mathbf{0}$ is the all-zero vector of length n. Note also that the distance between u and v is the same as the weight of their difference, that is, $d_H(u, v) = wt_H(u - v)$.

For a code C containing at least two codewords, the *minimum (Hamming) weight* $wt(C)$ of C is the smallest weight of all the nonzero codewords in C. The smallest Hamming distance between any two distinct codewords in C is called the *minimum (Hamming) distance* $d(C)$ of C. Note that since $d_H(u, v) = wt_H(u - v)$, the minimum distance of a code is the smallest value of $wt_H(u-v)$ as u and v, $u \neq v$, range over all possible codewords. Therefore, for linear codes, the minimum distance coincides with the minimum weight. If C is an $[n,k]$ linear code with minimum distance d, then it is also referred as an $[n,k,d]$ linear code. In a similar way, if C is an (n,M) code, not necessarily linear, with minimum distance d, then it is also referred as an (n,M,d) code.

Example 2.2.6. *For the linear codes C_1 and C_2, given in Example 2.2.1 and 2.2.2, respectively, it is easy to see that $wt(C_1)=d(C_1)=2$ and $wt(C_2)=d(C_2)=2$. For the linear codes C_3 and C_4, given in Example 2.2.5, we have that $wt(C_3)=d(C_3)=2$ and $wt(C_4)=d(C_4)=2$. Finally, for the small ternary nonlinear code $C=\{0000,1212,2121,1122,2211\}$, $wt(C)=4$ and $d(C)=2$.*

Two codes C_1 and C_2 of length n over $\mathbb{F}_q$ are said to be *equivalent* if there is a vector $a\in\mathbb{F}_q^n$, a monomial matrix $\mathcal{M}$ and an automorphism Γ of the field $\mathbb{F}_q$ such that $C_2=\{\mathcal{M}\Gamma(c)+a\ :\ c\in C_1\}$. Note that two equivalent codes have the same minimum distance. In the binary case, two codes C_1 and C_2 of length n are said to be *permutation equivalent* if there exists a coordinate permutation π such that $C_2=\{\pi(c)\ :\ c\in C_1\}$. They are said to be *equivalent* if there exists a vector $a\in\mathbb{F}_2^n$ and a coordinate permutation π such that $C_2=\{a+\pi(c)\ :\ c\in C_1\}$.

If C is a linear code over $\mathbb{F}_q$, then $\mathbf{0}\in C$; but if C is nonlinear, then $\mathbf{0}$ does not need to belong to C. In this case, we can always consider a new code $C'=C-c$ for any $c\in C$, which is equivalent to C, such that $\mathbf{0}\in C'$. Therefore, from now on, we assume that $\mathbf{0}\in C$.

2.3 Error detection and error correction

As we mentioned in the previous section, the main focus of coding theory is to detect and correct errors during the transmission of information through the channel. We start with an example to help us understanding the concept of detecting and correcting errors.

Example 2.3.1. *Let $C_1=\{00,01,10,11\}$. Since every vector in the space $\mathbb{F}_2^2$ is a codeword of C_1, any error added to a codeword will result to another codeword, so C_1 cannot detect any error.*

Let $C_2=\{000000,010101,101010,111111\}$, which can be constructed from C_1 by repeating each codeword three times. This is an example of a repetition code. Suppose that 110101 *is received, since this is not a codeword, we can detect that at least one error has occurred. The codeword* 010101 *can be formed by changing one bit, but all other codewords can be formed by changing more bits. Therefore, we expect that* 010101 *is the most likely codeword transmitted, so we correct* 110101 *to* 010101. *In fact if any of the codewords, $c \in C_2$, is transmitted and one error occurs during the transmission, then the unique closest codeword to the received vector is c.*

It is apparent that the addition of bits to codewords may improve the error correction and detection capabilities of the code. However, clearly the longer the codewords, the longer it takes to transmit each message. The *information rate* (or just *rate*) of a code is a number that is designed to measure the proportion of each codeword that is carrying the message. Specifically, the information rate of a code C over $\mathbb{F}_q$ of length n is defined as

$$R = \frac{1}{n}\log_q |C|.$$

Since $1 \leq |C| \leq q^n$, it is clear that the information rate ranges between 0 and 1. It is 1 if every vector is a codeword and 0 if $|C| = 1$.

Let C be a code over $\mathbb{F}_q$ of length n. If $c \in C$ is sent and $w \in \mathbb{F}_q^n$ is received, then $e = w - c$ is the *error vector*. Any vector $e \in \mathbb{F}_q^n$ can occur as an error vector, and we wish to know which error vectors can be detected or corrected by C. We say that the code C *detects* the error vector e if and only if $c + e$ is not a codeword, for every $c \in C$. In other words, e is detected if for any transmitted codeword c, the decoder, upon receiving $c + e$ can recognize that it is not a codeword and hence that some error has occurred. We say that the code C *corrects* the error vector e if and only if, for all $c \in C$, $c + e$ is closer to c than to any other codeword in C. Moreover, the code C is said to be a *t-error correcting code* if it corrects all error vectors of weight at most t and does not correct at least one error vector of weight $t + 1$. If C is a t-error correcting code, then t is called the *error correcting capability* of C.

To examine vectors closest to a given codeword, we bring in the concept of spheres around codewords. The *sphere* of radius r centered at a vector $c \in \mathbb{F}_q^n$ is defined to be the set

$$S_r(c) = \{v \in \mathbb{F}_q^n \mid d_H(c, v) \leq r\}$$

of all vectors whose distance from c is less than or equal to r. The number of vectors in $S_r(c)$ is equal to

$$\sum_{i=0}^{r} \binom{n}{i} (q-1)^i.$$

These spheres considered around all codewords of a code C are pairwise disjoint provided their radius is chosen small enough.

Theorem 2.3.2. *[19] If d is the minimum distance of a code C (linear or nonlinear) and $t = \lfloor (d-1)/2 \rfloor$, then the spheres of radius t around distinct codewords are disjoint.*

Theorem 2.3.3. *[19] If d is the minimum distance of a code C (linear or nonlinear), $t = \lfloor (d-1)/2 \rfloor$, a codeword c is sent and w is received where t or fewer errors have occurred, then c is the unique codeword closest to w.*

Find the codeword c closest to the received vector w according to the Hamming distance and decode w as c is called *nearest neighbor decoding.* Nearest neighbor decoding uniquely and correctly decodes any received vector in which at most t errors have occurred in transmission. Since the minimum distance of a code C is d, there exist two distinct codewords such that the spheres of radius $t+1$ around them are not disjoint. Therefore, if more than t errors occur, nearest neighbor decoding may yield more than one nearest codeword. Thus C is a t-error correcting code, but not a $(t+1)$-error correcting code.

For purposes of decoding as many errors as possible, for given n and M, we wish to find an (n, M) code C with as high a minimum distance d as possible. Alternately, given n and d, one wish is to send as many messages as possible; thus we want to find a code with the largest number of codewords M, or, in the linear case, the highest dimension k.

The nearest neighbor decoding defines a decoding scheme. In general, given a decoding scheme, the *probability of error*, P_{err}, is the probability that the decoder outputs a vector c' distinct from the sent codeword c. For a given information rate R, Shannon's Theorem shows how small we can make P_{err} using an error correcting code.

For a BSC with reliability p, the *capacity* of the channel is

$$C(p) = 1 + p \log_2 p + (1-p) \log_2 (1-p).$$

Theorem 2.3.4 (Shannon's Theorem)**.** *[24] For any $\epsilon > 0$, if $R < C(p)$ and n is sufficiently large, there is a binary code of information rate R with probability of error $P_{err} < \epsilon$.*

A similar result holds for any discrete memoryless channel.

The goal of research in coding theory is to produce codes that fulfill the conditions of Shannon's Theorem. Once the code is chosen for application, encoding is usually rather straightforward. On the other hand, decoding efficiently can be a much more difficult task.

Suppose that we have an overall view of the transmission process, knowing both the codeword c that is transmitted and the vector w that is received. For any given c and w, let $\phi_p(c, w)$ be the probability that if the codeword c is sent over a BSC with reliability p then a vector w is received. Since we are assuming that noise is distributed randomly, we can treat the transmission of each bit as an independent event. Therefore, if c and w disagree in d positions, then we have $n - d$ bits correctly transmitted and d incorrectly transmitted and thus,

$$\phi_p(c, w) = p^{n-d}(1 - p)^d.$$

In practice, we know w, the vector received, but we do not know the actual codeword c that was sent. However, each codeword c determines an assignment of probabilities $\phi_p(c, w)$ to vectors, w. Each such assignment is a mathematical model and we choose the model (that is, the codeword c) which agrees most with observation, in this case, which makes the vector received most likely. That is, assume c is sent when w is received if

$$\phi_p(c, w) = \max\{\phi_p(u, w) \mid u \in C\}.$$

This formally establishes the procedure for correcting vectors which until now we had adopted as being an intuitively sensible procedure: correct w to a codeword which disagrees with w in as few positions as possible, since such a codeword is the most likely to have been sent, given that w was received. Theorem 2.3.5 provides a criterion for finding such a codeword c.

Theorem 2.3.5. *[18] Suppose we have a BSC with reliability p such that $1/2 < p < 1$. Let c_1 and c_2 be two codewords and w a vector, each of length n. Suppose that c_1 and w disagree in d_1 positions and c_2 and w disagree in d_2 position. Then*

$$\phi_p(c_1, w) \leq \phi_p(c_2, w) \text{ if and only if } d_1 \geq d_2.$$

2.4 Maximum likelihood decoding

According to the last theorem, we can describe a procedure, called *maximum likelihood decoding* (*MLD*), for deciding which codeword c in C was sent, when a vector w in $\mathbb{F}_q^n$ is received. There are actually two kinds of MLD.

(1) *Complete Maximum Likelihood Decoding (CMLD).* If there is one and only one codeword $c \in C$ closer to w than any other codeword in C, we decode w as c. That is, if $d_H(c, w) < d_H(c', w)$ for any $c' \in C$, $c' \neq c$, then decode w as c. If there are several codewords in C close to w, i.e, at the same distance from w, then we select arbitrarily one of them and conclude that it was the codeword sent.

(2) *Incomplete Maximum Likelihood Decoding (IMLD).* Again, if there is a unique codeword $c \in C$ closest to w, then we decode w as c. However, if there are several codewords in C at the same distance from w, then we request a retransmission. In some cases, we might even ask for a retransmission if the receiver vector w is too far away from any codeword in the code.

The decoding problem now becomes one of finding an efficient algorithm that will correct up to t errors, where t is the error correcting capability of the code. One of the most obvious decoding algorithms is to examine all codewords until one is found at distance t or less from the received vector. But obviously this is a realistic decoding algorithm only for codes with a small number of codewords. Another obvious algorithm is to make a table consisting of a nearest codeword for each of the q^n vectors in $\mathbb{F}_q^n$ and then look up a received vector in the table in order to decode it. This is also impractical when q^n is large.

For an $[n, k, d]$ linear code C over $\mathbb{F}_q$, we can, however, devise an algorithm using a table with q^{n-k} rather than q^n entries where one can find the nearest codeword by looking up one of these q^{n-k} entries. This general decoding algorithm for linear codes is called *syndrome decoding.* Since a linear code C is an elementary abelian subgroup of the additive group of $\mathbb{F}_q^n$, its distinct *cosets* $C + v$ partition $\mathbb{F}_q^n$ into q^{n-k} sets of size q^k [24]. Two vectors v_1 and v_2 belong to the same coset of C if and only if $v_1 - v_2 \in C$. The weight of a coset is the smallest weight of a vector in the coset, and any vector of this smallest weight in the coset is called *coset leader.* The all-zero vector is the unique coset leader of the linear code C. More generally, every coset of weight at most $t = \lfloor (d-1)/2 \rfloor$ has a unique coset leader.

The *syndrome* of a vector $v \in \mathbb{F}_q^n$, with respect to a parity check matrix H for the linear code C, is the vector in $\mathbb{F}_q^{n-k}$ defined as

$$\mathrm{Syn}(v) = Hv^T.$$

The linear code C consists of all vectors whose syndrome is equal to the all-zero vector. As H has rank $n - k$, every vector in $\mathbb{F}_q^{n-k}$ is a syndrome.

Lemma 2.4.1. *[19] $v_1, v_2 \in \mathbb{F}_q^n$ are in the same coset of C if and only if $Syn(v_1) = Syn(v_2)$.*

Since two vectors belonging to the same coset have the same syndrome, then there exists a one-to-one correspondence between cosets of C and its syndromes. We denote by C_s the coset of C consisting of all vectors in $\mathbb{F}_q^n$ with syndrome s.

Suppose a codeword sent over a communication channel is received as a vector w. Since in nearest neighbor decoding we seek a vector e of smallest weight such that $w-e \in C$, nearest neighbor decoding is equivalent to finding a vector e of smallest weight in the coset containing w, that is, a coset leader of the coset containing w.

Syndrome decoding requires a table with only q^{n-k} entries, which may be a vast improvement over a table of q^n vectors showing which codeword is closet to each of these. However, there is a cost for shortening the table: before looking in the table of syndromes, one must perform a matrix-vector multiplication in order to determine the syndrome of the received vector. Then the table is used to look up the syndrome and find the coset leader.

2.5 Covering radius

The minimum distance d is a simple measure of the goodness of a code. For a given length n and number of codewords M, a fundamental problem in coding theory is to produce a (n, M, d) code with the largest possible d. Alternatively, given n and d, determine the maximum number $A_q(n, d)$ of codewords in a code over $\mathbb{F}_q$ of length n and minimum distance at least d. For linear codes, the maximum number of codewords is denoted by $B_q(n, d)$. Clearly, $B_q(n, d) \leq A_q(n, d)$. For modest values of n and d, $A_2(n, d)$ and $B_2(n, d)$ have been determined and tabulated [11].

According to the definition of the spheres of radius $t = \lfloor (d-1)/2 \rfloor$ around codewords, which are pairwise disjoint, we can have the following inequality, commonly referred to as the *Sphere Packing Bound* or the *Hamming Bound.*

Theorem 2.5.1. *[24]*

$$B_q(n, d) \leq A_q(n, d) \leq \frac{q^n}{\sum_{i=0}^{t} \binom{n}{i} (q-1)^i}, \text{ where } t = \lfloor (d-1)/2 \rfloor.$$

When we obtain equality in the Hamming Bound, we actually fill the space $\mathbb{F}_q^n$ with disjoint spheres of radius t. In other words, every vector in $\mathbb{F}_q^n$ is contained in precisely one sphere of radius t centered around a codeword. When we have a code for which this is true, the code is called *perfect*.

When we do not have a perfect code, in order to fill the space $\mathbb{F}_q^n$ with spheres centered at codewords, the sphere must have radius larger than t. It is easy to see that when we increase the sphere size, some spheres will not be pairwise disjoint. The *covering radius* of a code C, denoted by $\rho(C)$, is the smallest integer s such that $\mathbb{F}_q^n$ is the union of the spheres of radius s centered at the codewords of C. Equivalently,

$$\rho(C) = \max\{d_H(x, C) \mid x \in \mathbb{F}_q^n\},$$

where $d_H(x, C) = \min\{d_H(x, c) \mid c \in C\}$.

Obviously, $t \leq \rho(C)$ and $t = \rho(C)$ if and only if C is perfect. Therefore, a code is perfect if and only if its covering radius equals its error correcting capability. If the code is not perfect, its covering radius is larger than its error correcting capability.

If C is a code with error correcting capability t and covering radius $t+1$, then C is called *quasi-perfect*. There are many known linear and nonlinear quasi-perfect codes (e.g. certain double error correcting BCH codes and some punctured Preparata codes). However, unlike perfect codes, there is no general classification [19].

2.6 Constructing new codes from old

As it is known, many interesting and important codes arise by modifying or combining existing codes. In this section, we will discuss five different known techniques to construct new linear codes from given linear codes. Specifically, we will describe how to construct extended codes, punctured codes, shortened codes, direct sums, and Plotkin sums from linear codes over a finite field. However, all these descriptions can also be applied to nonlinear codes [19, 24].

2.6.1 Punctured codes

Let C be an $[n, k, d]$ linear code over $\mathbb{F}_q$. The *punctured code* of C in the jth coordinate, denoted by C^j, is the code consisting of the codewords of C after deleting the jth coordinate in each codeword. The punctured code of a linear code is also linear and its length is $n-1$. If G is a generator matrix for C, then a generator matrix for C^j is obtained from G by deleting the column j (and omitting a all-zero or duplicate row that may occur). Since C contains q^k codewords, the only way that C^j could contain fewer codewords is if two codewords of C agree in all by coordinate j. In that case, C has minimum distance $d = 1$ and a codeword of weight 1 whose nonzero entry is in coordinate j. The minimum distance decrease by 1 only if a minimum weight codeword of C has a nonzero jth coordinate. Summarizing, we have Theorem 2.6.1.

Theorem 2.6.1. *[19] Let C be an $[n, k, d]$ linear code over $\mathbb{F}_q$, and let C^j be the code C punctured on the jth coordinate.*

(i) If $d > 1$, C^j is an $[n-1, k, d']$ linear code where $d' = d - 1$ if C has a minimum weight codeword with a nonzero jth coordinate and $d' = d$ otherwise.

(ii) If $d = 1$, C^j is an $[n-1, k, 1]$ linear code if C has no codeword of weight 1 whose nonzero entry is in coordinate j; otherwise, if $k > 1$, C^j is an $[n-1, k-1, d']$ linear code with $d' \geq 1$.

Example 2.6.2. *Let C be the $[7, 3, 3]$ binary linear code, defined by the below generator matrix G. The punctured code C^5 is the code consisting of the codewords of C after deleting the 5th coordinate. Therefore, since C is linear, C^5 is also a linear code of length 6, dimension 3, and minimum distance 2, with generator matrix G^5.*

$$G = \begin{pmatrix} 1001111 \\ 0100101 \\ 0011100 \end{pmatrix} \qquad G^5 = \begin{pmatrix} 100111 \\ 010001 \\ 001100 \end{pmatrix}$$

2.6.2 Extended codes

It is possible to create longer codes by adding a coordinate. There are many possible ways to extend a code, but the most common is to choose the extension so that the sum of all coordinates is 0.

Let C be an $[n, k, d]$ linear code over $\mathbb{F}_q$. The *extended code* of C, denoted by $\widehat{C}$, is defined as

$$\widehat{C} = \{x_1x_2\cdots x_{n+1} \in \mathbb{F}_q^{n+1} \mid x_1x_2\cdots x_n \in C \text{ with } \sum_{i=1}^{n+1} x_i = 0\}.$$

Since C is linear, the extended code $\widehat{C}$ is also linear. Moreover, $\widehat{C}$ is an $[n+1,k,\widehat{d}]$ linear code, where $\widehat{d}=d$ or $d+1$. Let G and H be generator and parity check matrices, respectively, for C. Then, a generator matrix $\widehat{G}$ for $\widehat{C}$ can be obtained from G by adding an extra column to G so that the sum of the coordinates of each row of $\widehat{G}$ is 0. A parity check matrix $\widehat{H}$ for $\widehat{C}$ is the matrix

$$\widehat{H} = \begin{pmatrix} 1 & \cdots & 1 & 1 \\ & & & 0 \\ & H & & \vdots \\ & & & 0 \end{pmatrix}.$$

This construction is also referred to as adding an overall parity check coordinate.

If C is an $[n,k,d]$ binary linear code, then the extended code $\widehat{C}$ contains only even weight vectors and is an $[n+1,k,\widehat{d}]$ code, where $\widehat{d}$ equals d if d is even and equals $d+1$ if d is odd. In the nonbinary case, however, whether or not $\widehat{d}$ is d or $d+1$ is not so straightforward. For an $[n,k,d]$ linear code C over $\mathbb{F}_q$, denote by d_e the minimum distance between pairs of codewords having the same overall parity check coordinate and by d_o the minimum distance between pairs of codewords having different overall parity check coordinate. Therefore, $d=\min\{d_e,d_o\}$. If $d_e \leq d_o$, then $\widehat{C}$ has minimum distance $\widehat{d}=d_e$; otherwise $\hat{d}=d_o+1$ [24].

Example 2.6.3. *Let C be the $[7,3,3]$ binary linear code, defined by the below generator matrix G. The extended code $\widehat{C}$ is the linear code of length 8, dimension 3, and minimum distance 4, with generator matrix $\widehat{G}$.*

$$G = \begin{pmatrix} 1001111 \\ 0100101 \\ 0011100 \end{pmatrix} \qquad \widehat{G} = \begin{pmatrix} 10011111 \\ 01001011 \\ 00111001 \end{pmatrix}$$

2.6.3 Shortened codes

Let C be an $[n,k,d]$ linear code over $\mathbb{F}_q$ and $C_0 \subseteq C$ be the subset of C containing the codewords having 0 in the jth coordinate. The *shortened code* of C in the jth coordinate, denoted by C_0^j, is the code consisting of the codewords of C having 0 in the jth coordinate and deleting this coordinate. That is, C_0^j can be seen as the punctured code of C_0 in the jth coordinate.

The shortened code C_0^j is an $[n-1,k',d']$ linear code with $k-1 \leq k' \leq k$ and $d \leq d'$. Note that the shortened code C_0^j is a subcode of the punctured code C^j (i.e. $C_0^j \subseteq C^j$).

Example 2.6.4. *Let C be the $[7,3,3]$ binary linear code, defined by the below generator matrix G. The shortened code C_0^5 is the linear code of length 6, dimension 2, and minimum distance 4, with generator G_0^5.*

$$G = \begin{pmatrix} 1001111 \\ 0100101 \\ 0011100 \end{pmatrix} \qquad G_0^5 = \begin{pmatrix} 101011 \\ 011101 \end{pmatrix}$$

2.6.4 Direct sum construction

Let C_i be an $[n_i, k_i, d_i]$ linear code for $i \in \{1,2\}$, both over the same finite field $\mathbb{F}_q$. Then, their *direct sum* is the $[n_1+n_2, k_1+k_2, \min\{d_1,d_2\}]$ linear code

$$C_1 \oplus C_2 = \{(c_1,c_2) \mid c_1 \in C_1,\ c_2 \in C_2\}.$$

If C_i has generator matrix G_i and parity check matrix H_i, then

$$G_1 \oplus G_2 = \begin{bmatrix} G_1 & 0 \\ 0 & G_2 \end{bmatrix} \quad \text{and} \quad H_1 \oplus H_2 = \begin{bmatrix} H_1 & 0 \\ 0 & H_2 \end{bmatrix}$$

are a generator matrix and parity check matrix for $C_1 \oplus C_2$, respectively.

Since the minimum distance of the direct sum of two codes does not exceed the minimum distance of either of the codes, the direct sum of two codes is generally of little use in applications and is primarily of theoretical interest.

Example 2.6.5. *Let C_1 be the $[7,3,3]$ binary linear code and C_2 be the $[5,3,2]$ binary linear code with generator matrices G_1 and G_2, respectively, where*

$$G_1 = \begin{pmatrix} 1001111 \\ 0100101 \\ 0011100 \end{pmatrix} \quad \textit{and} \quad G_2 = \begin{pmatrix} 10100 \\ 01101 \\ 00011 \end{pmatrix}.$$

The direct sum $C_1 \oplus C_2$ is the $[12,6,2]$ binary linear code with generator matrix

$$G_1 \oplus G_2 = \left(\begin{array}{c|c} 1001111 & 00000 \\ 0100101 & 00000 \\ 0011100 & 00000 \\ \hline 0000000 & 10100 \\ 0000000 & 01101 \\ 0000000 & 00011 \end{array}\right).$$

2.6.5 Plotkin sum construction

Let C_i be an $[n, k_i, d_i]$ linear code for $i \in \{1, 2\}$, both over the same finite field $\mathbb{F}_q$. Then, their *Plotkin sum* is the $[2n, k_1 + k_2, \min\{2d_1, d_2\}]$ linear code

$$C_1 \oplus (C_1 + C_2) = \{(c_1, c_1 + c_2) \mid c_1 \in C_1,\ c_2 \in C_2\}.$$

If C_i has generator matrix G_i and parity check matrix H_i, then a generator matrix and parity check matrix for $C_1 \oplus (C_1 + C_2)$ are

$$\begin{bmatrix} G_1 & G_1 \\ 0 & G_2 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} H_1 & 0 \\ -H_2 & H_2 \end{bmatrix}, \quad \text{respectively.}$$

Unlike the direct sum construction given in the previous subsection, the Plotkin sum construction can produce codes that are important for reasons other than theoretical. For example, different families of Reed-Muller codes can be constructed in this way [24, 30].

Example 2.6.6. *Let C_1 be the $[7, 3, 3]$ binary linear code and C_2 be the $[7, 2, 4]$ binary linear code with generator matrices G_1 and G_2, respectively, where*

$$G_1 = \begin{pmatrix} 1001111 \\ 0100101 \\ 0011100 \end{pmatrix} \quad \text{and} \quad G_2 = \begin{pmatrix} 1010110 \\ 0110101 \end{pmatrix}.$$

The Plotkin sum $C_1 \oplus (C_1 + C_2)$ is the $[14, 5, 4]$ binary linear code with generator matrix

$$\left(\begin{array}{c|c} 1001111 & 1001111 \\ 0100101 & 0100101 \\ 0011100 & 0011100 \\ \hline 0000000 & 1010110 \\ 0000000 & 0110101 \end{array}\right).$$

2.7 Minimum distance computation

Computing the minimum distance of a code over a finite field is necessary in order to establish its error correcting capability. However, it is computationally difficult, and has been proven to be an NP-hard problem [37].

Recall that the minimum distance coincides with the minimum weight when the code is linear. In this section, we will describe the brute force algorithm, Brouwer's algorithm and Brouwer-Zimmermann's algorithm to compute the minimum weight for linear codes over finite fields [39, 41, 2]. The

Brouwer-Zimmermann's algorithm is the best known enumerative algorithm for linear codes, and can be found implemented in the computational algebra system Magma [9, 14, 39]. To write the next subsections, we have mainly used reference [39]. For more information on these algorithms, refer to [39, 41, 2].

Other algorithms related to the problem of computing the minimum weight for linear codes can be found in [7] where Gröbner bases are used, [25] where a tree structure is used, and [10] where an efficient method to compute the number of codewords of fixed weights is described. Probabilistic algorithms on computing the minimum weight of linear codes are described in [13, 23]. Another algorithm based on Gröbner bases to compute the distance distribution for systematic nonlinear codes can be found in [15].

2.7.1 Work factor

In order to establish the complexity of the algorithms described in this section, a criterion is needed. All the algorithms are based on the enumeration of vectors, adding vectors and computing the weight of vectors. In most circumstances, the enumeration is structured such that a single vector addition is performed for each time the weight of a vector is determined. This gives rise to a natural performance measure, which is referred to as *work* [20]. Despite of the fact that there are certain ways to speed up the computation on vectors, we will use the concept of work to compare the performance of different algorithms. One unit of work represents both the addition and the subsequent weight determination of a single coordinate. An estimate of the total work an algorithm performs is called *work factor*. Note that since CPU operations typically deal with either 32 or 64 bits at once, a single unit of work costs less than one CPU operation [39].

Work factor gives accurate estimation of run times and is a good tool for comparing the performance of enumeration-based algorithms. Throughout this dissertation, we will use the work factor as the primary measure to evaluate the performance of algorithms related to the minimum weight and minimum distance computation. Along with the theoretical estimation of work factor, some real tests on some algorithms are performed on Magma version V2.18-3, running on a server with an Intel Xeon processor (clock speed 2.40GHz) and 32GB of memory.

2.7.2 Brute force algorithm

The most straightforward method to compute the minimum weight and minimum distance is called *brute force*, which consists in checking the weight of every codeword to compute the minimum weight, and checking the distance between any pair of codewords to compute the minimum distance. Therefore, the work factors for computing the minimum weight and minimum distance of an (n, M, d) code over $\mathbb{F}_q$ using the brute force algorithm are

$$\log_2(q) \cdot n \cdot M \quad \text{and} \quad \log_2(q) \cdot n \cdot \binom{M}{2}, \tag{2.1}$$

respectively.

The brute force method to compute the minimum weight or minimum distance can be satisfactory for codes with small cardinality, but it is really impractical for those with a large cardinality. However, for linear codes, there exist other algorithms that allow to reduce the proportion of codewords that need to be enumerated in the process of computing the minimum weight, which is the same as computing the minimum distance. There are two main approaches used to compute the minimum weight of an $[n, k, d]$ linear code over $\mathbb{F}_q$: algorithms based on enumeration of codewords [39, 41, 2] and probabilistic algorithms [13, 23].

2.7.3 Brouwer's algorithm

The algorithm to compute the minimum weight of a linear code given by Brouwer dates from the 1980s, though it has never been published [39, 40]. Comparing with the brute force method, Brouwer's algorithm reduces the proportion of codewords of the linear code to be enumerated, by obtaining a *lower bound* on the weight of those codewords which have not been enumerated and an *upper bound* on the weight of those codewords which have been enumerated. When the upper bound is equal or smaller than the lower bound, we obtain the minimum weight of the linear code without enumerating necessarily all its codewords.

Lower bounds are obtained by enumerating codewords from systematic generator matrices of the linear code. More specifically, Brouwer's algorithm is based on Lemma 2.7.1 and Theorem 2.7.2, which give the lower bounds. The *information weight* of a codeword $v \in C$, denoted by $wt_I(v)$, is the Hamming weight of v restricted to the coordinate positions included in the information set I for C.

Lemma 2.7.1. *[39] Let G be a systematic generator matrix of a $[n,k,d]$ linear code C over $\mathbb{F}_q$, with information set I. For any $r<k$, if*

$$S=\{mG \mid m\in\mathbb{F}_q^k,\ wt_H(m)\leq r\},$$

then the information weights of all $v\in C\backslash S$ satisfy

$$wt_I(v)\geq r+1.$$

Note that Lemma 2.7.1 only gives a lower bound on the information weight of unenumerated codewords. In other words, only the weight restricted to a set of k coordinate positions is considered, while any weight on the remaining $n-k$ coordinate positions is being neglected.

By manipulating a generator matrix G of a linear code C, we can obtain a sequence of systematic generator matrices $G_1,\ldots,G_h$ of C with pairwise disjoint information sets. These matrices can be obtained applying Gaussian elimination to the coordinate positions which are not contained in the information sets of the previous computed matrices. For an $[n,k,d]$ linear code, we can obtain at most $h=\lfloor n/k\rfloor$ such generator matrices, which use $h\cdot k$ columns of the generator matrix in the the process of computing the minimum weight.

Theorem 2.7.2. *[39] Let $G_1,\ldots,G_h$ be a sequence of systematic generator matrices for a $[n,k,d]$ linear code C over $\mathbb{F}_q$, with pairwise disjoint information sets. For any $r<k$, if*

$$S_i=\{mG_i \mid m\in\mathbb{F}_q^k,\ wt_H(m)\leq r\},$$

for each matrix G_i, then all $v\in C\backslash\bigcup_{i=1}^h S_i$ satisfy

$$wt_H(v)\geq h(r+1).$$

Brouwer's algorithm follows naturally from Theorem 2.7.2. In Brouwer's algorithm, the enumeration process consists of several *steps* given by $r\in\{1,\ldots,k\}$. In the rth step, all linear combinations of r rows from the h generator matrices are enumerated, each one with the information coordinates deleted. Then, we obtain a lower bound given by $h(r+1)$, and a upper bound given by the minimum weight of the enumerated codewords. The enumeration process is terminated as soon as the upper bound is equal or smaller than the lower bound. Figure 2.1 illustrates the process involved in Brouwer's algorithm. The step r in which the process finishes is called *termination step* and denoted by $\bar{r}$. Then, we have that the work factor of Brouwer's algorithm is given by the following Proposition 2.7.3.

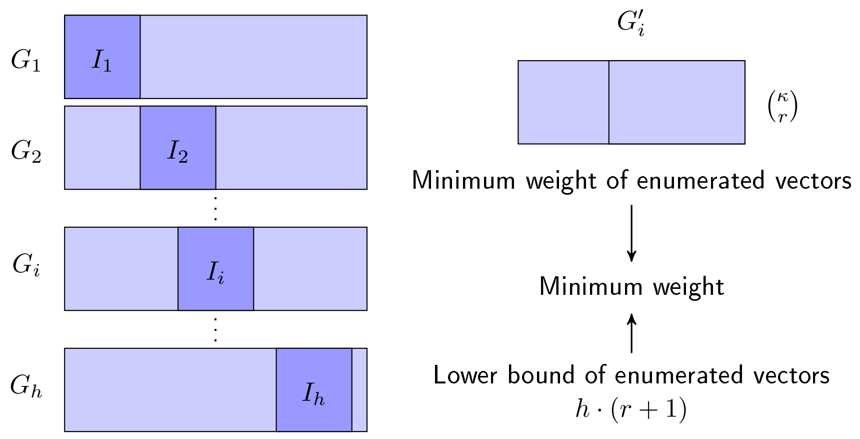


Figure 2.1: Illustration of the Brouwer's algorithm.

Proposition 2.7.3. *[39] The work factor for computing the minimum weight of an $[n, k, d]$ linear code over $\mathbb{F}_q$ using Brouwer's algorithm is*

$$W_{Brouwer}(n, k, d; q) = \log_2(q)(n-k)\lfloor n/k \rfloor \sum_{r=1}^{\bar{r}} \binom{k}{r} (q-1)^{r-1},$$

where $\bar{r}$ is the smallest integer such that

$$\lfloor n/k \rfloor (\bar{r}+1) \geq d.$$

2.7.4 Zimmermann's improvement

The lower bounds on the minimum weight given by Brouwer's algorithm consist on the summation of bounds from the h pairwise disjoint information sets. Note that there are at most $\lfloor n/k \rfloor$ disjoint information sets, but actually, due to linear dependencies between columns, it is usually difficult to find exactly $\lfloor n/k \rfloor$ disjoint information sets, which means that probably $h < \lfloor n/k \rfloor$. In these cases, several columns do not contribute in the growing of the lower bounds.

An improvement of Brouwer's algorithm by Zimmermann [41] enables us to use all the columns during the computation of the minimum weight by allowing overlaps in the coordinate positions of the information sets. This means that every column can contribute to the lower bound, which may allow the lower bound to grow faster. According to Zimmermann's improvement, if an information set of a generator matrix is introduced with some overlaps

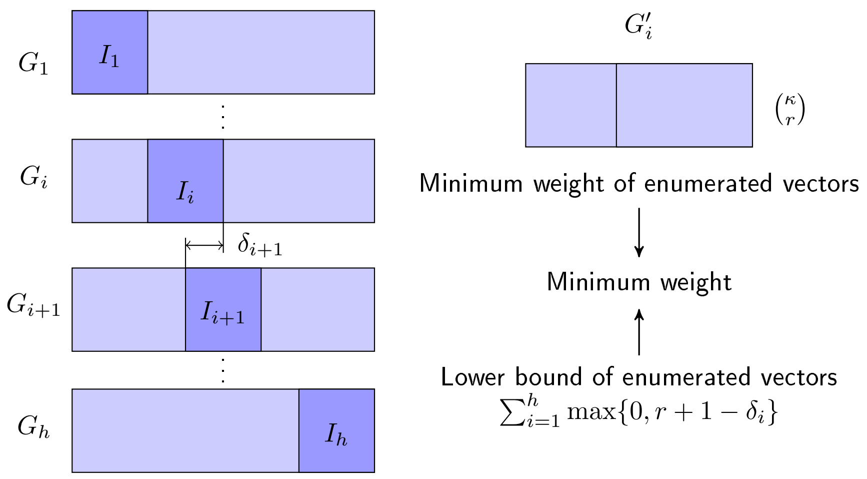

G_1
I_1
G_i
I_i
δ_{i+1}
G_{i+1}
I_{i+1}
G_h
I_h
G'_i
$\binom{\kappa}{r}$
Minimum weight of enumerated vectors
Minimum weight
Lower bound of enumerated vectors
$\sum_{i=1}^{h} \max\{0, r+1-\delta_i\}$

Figure 2.2: Illustration of the Zimmermann's algorithm.

with the information sets of the preceding generator matrices, then its contribution to the lower bound can simply be reduced by the size of the overlap. Let $G_1, \ldots, G_h$ be the sequence of systematic generator matrices with information sets $I_1, \ldots, I_h$, respectively, not necessarily pairwise disjoint. The number of overlapped coordinate positions between one information set I_i and its preceding, $I_1, I_2, \ldots, I_{i-1}$, is called the *rank deficit* and is denoted by δ_i for $i \in \{1, \ldots, h\}$. Note that $\delta_1 = 0$.

Theorem 2.7.4. *[39] Let $G_1, \ldots, G_h$ be a sequence of systematic generator matrices for a $[n, k, d]$ linear code C over $\mathbb{F}_q$, with rank deficits $\delta_1, \ldots, \delta_h$. For any $r < k$, if*

$$S_i = \{mG_i \mid m \in \mathbb{F}_q^k,\ wt_H(m) \leq r\},$$

for each matrix G_i, then all $v \in C \setminus \bigcup_{i=1}^{h} S_i$ satisfy

$$wt_H(v) \geq \sum_{i=1}^{h} \max\{0, r+1-\delta_i\}.$$

Along with this generalized lower bound given by Theorem 2.7.4, Zimmermann also described a linear algebra process to obtain the sequence of systematic generator matrices $G_1, \ldots, G_h$. Gaussian elimination is repeated until all coordinate positions have been included in at least one information set. Figure 2.2 illustrates the process involved in Zimmermann's algorithm. Note that Brouwer's lower bound can be considered to be an special case of this generalized lower bound of Zimmermann, in which every information set has a rank deficit of 0.

The process of enumeration of Zimmermman's improvement is the same as Brouwer's algorithm. In the most usual situations, when it is possible to find the maximal number of disjoint information sets, Zimmermann's algorithm allows to use an extra generator matrix whose information set includes the "left-over" coordinate positions [39]. The rank deficit of this final matrix is $k - (n \bmod k)$. Then, we have that the work factor of Zimmermann's algorithm is given by the following Proposition 2.7.5. Note that, in this case, if $k \mid n$, then Brouwer's and Zimmermann's algorithms are identical.

Proposition 2.7.5. *[39] The work factor for computing the minimum weight of an $[n, k, d]$ linear code over $\mathbb{F}_q$ using Zimmermann's improvement is*

$$W_{Zimm}(n, k, d; q) = \log_2(q)(n-k)\lceil n/k \rceil \sum_{r=1}^{\bar{r}} \binom{k}{r} (q-1)^{r-1},$$

where $\bar{r}$ is the smallest integer such that

$$\lfloor n/k \rfloor (\bar{r}+1) + \max\{0, \bar{r}+1-(k-n \bmod k)\} \geq d.$$

In fact, although Zimmermann's improvement uses more generator matrices, the growth rate of the lower bound is not always faster than Brouwer's algorithm. A selection of the fastest one for a given code is needed. In this dissertation, the term *Brouwer-Zimmermann's algorithm* refers to the optimal choice between Brouwer's and Zimmermann's version of the algorithm.

Chapter 3

Representation and constructions

In coding theory, linear codes are studied the most for their practical advantages given by the linearity property. For example, they can be compactly represented by its generator matrices or parity-check matrices. On the other hand, comparing with linear codes, nonlinear codes are less studied partially because of the hardness of having a compact representation which allows to encode and decode also in an efficient way. However, it is known that some nonlinear codes have more codewords than any linear one with the same parameters, length and minimum distance, so they are better than any linear code. It is also known that nonlinear codes can be seen as a union of cosets of a linear subcode of the code, called kernel.

In this chapter, firstly, we show that this representation allows to store nonlinear codes more efficiently. Then, we describe and analyze the complexity of some algorithms to efficiently compute the kernel and coset representatives from a given nonlinear code, first for binary codes and then generalized to q-ary codes ($q > 2$). Finally, we give some properties and describe constructions of new codes from given ones in terms of this representation.

3.1 Representation of nonlinear codes

3.1.1 Representation of binary nonlinear codes

Given a binary code C, the problem of storing C in memory is a well known problem. If C is linear, that is, it is a subgroup of $\mathbb{F}_2^n$, then it can be compactly represented using a binary generator matrix. On the other hand, if C is nonlinear, then a solution would be to know whether it has another structure or not. For example, there are nonlinear binary codes which have a $\mathbb{Z}_4$-linear or $\mathbb{Z}_2\mathbb{Z}_4$-linear structure and, therefore, they can also be compactly represented using a quaternary generator matrix [6, 16]. In general, binary codes without any of these structures can be represented as a union of cosets of a binary linear subcode of C [1]. This allows us to represent a binary code as a set of representative codewords instead of as a set with all codewords. Moreover, these representative codewords can be organized as a matrix, called *parity-check system* [17], which is a generalization of the parity-check matrix for linear codes [24].

Two structural properties of binary codes are the rank and dimension of the kernel. The *rank* of a binary code C, denoted by ϱ, is simply the dimension of the linear span, $\langle C \rangle$, of C. The *kernel* of a binary code C is defined as $K_C = \{x \in C \ : \ x + C = C\}$ [1]. Since $\mathbf{0} \in C$, K_C is a binary linear subcode of C. We denote by κ the dimension of K_C. In general, C can be written as a union of cosets of K_C, and K_C is the largest such linear code for which this is true [1]. Therefore,

$$C = \bigcup_{i=0}^{t} \Big(K_C + v_i \Big), \tag{3.1}$$

where $v_0 = \mathbf{0}$, $t + 1 = M/2^{\kappa}$, $M = |C|$, and $v_1, \dots, v_t$ are representatives of the cosets of K_C. Note that $t \neq 1$, because if $t = 1$, $C = K_C \cup (K_C + v_1)$, but then C would be linear, so $C = K_C$. It is also important to emphasize that the coset representatives are not necessarily the ones having minimum weight in each coset. The parameters ϱ and κ can be used to distinguish between nonequivalent binary codes, since equivalent ones have the same ϱ and κ.

Let C be a binary code of length n with kernel K_C of dimension κ and t coset representatives given by the set $L = \{v_1, \dots, v_t\}$. Note that we can represent C as the kernel K_C plus the coset representatives L. Since K_C is linear, it can be compactly represented by its binary generator matrix G of size $\kappa \times n$. Therefore, considering L as the matrix where in the t rows there are the coset representatives, the binary code C can be also represented by the matrix $\binom{G}{L}$. Since the kernel takes up a memory space of order $O(n\kappa)$, the kernel plus the t coset representatives take up a memory space of order $O(n(\kappa + t))$. For the case $t = 0$, that is, when the binary code C is linear, we have that $C = K_C$ and the code can be represented by its binary generator matrix, so the memory space is of order $O(n\kappa)$. On the other hand, for the case $t + 1 = M$, this solution is as bad as representing the code as a set of all its codewords, so it takes up a memory space of order $O(nM)$.

For example, applying this representation to the set of all completely classified binary perfect codes of length 15 and extended perfect codes of length 16, we obtain very significant compression rates. It is known that there are exactly 5983 binary perfect codes of length 15 and 2165 binary extended perfect codes of length 16, each one having 2048 codewords [26]. The binary perfect codes of length 15 have kernels of different dimensions, distributed as it is shown in Table 3.1 [26]. Therefore, instead of taking up $5983 \cdot 2048 \cdot 4 = 49012736$ hexadecimal numbers by encoding each codeword in hexadecimal notation, it only takes 3677928 hexadecimal numbers by storing the codewords of a generator matrix of the kernel and the set of coset representatives for each binary code. This gives a compression rate of 92.5%. Similarly, the extended perfect codes of length 16 can be compressed from $2165 \cdot 2048 \cdot 4 = 17735680$ hexadecimal numbers to 1439336, which gives a compression rate of 91.9%. Note that although most of these codes have kernels with small dimension, that is, they are far from being linear, we obtain a high compression.

Dimension of kernel	1	2	3	4	5	6	7	8	9	11
Number of codes	19	177	1295	2896	1222	305	48	17	3	1

Table 3.1: Dimension of the kernels for the binary perfect codes of length 15.

As we have seen, the above matrix $\binom{G}{L}$ gives us a compact representation for binary codes. Equivalently, a binary code C can also be represented in a compact way using an $(n-\kappa) \times (n+t)$ binary matrix $(H\ S)$, where H is a generator matrix of the dual code $K_C^\perp$ and $S = (Hv_1^T\ Hv_2^T\ \dots\ Hv_t^T)$. This matrix is called the *parity-check system* of the binary code C, and the binary linear code generated by $(H\ S)$ is called the *super dual* of C [17]. Unlike the matrix $\binom{G}{L}$, any matrix equivalent to the parity-check system $(H\ S)$ can be used to represent the binary code C. Note that if C is a linear code, the super dual is the dual code $C^\perp$ and the parity-check system is a parity-check matrix of C [17].

Algorithm 1 describes a straightforward algorithm to compute the kernel and coset representatives of a binary code C, by using just the definition of the kernel. This algorithm requires the classification of the M codewords of C. Moreover, if we assume that the codewords are sorted, then $M^2 \log M$ operations (additions and searches) would need to be executed. Since $M = 2^\kappa(t+1)$, Algorithm 1 is exponential in κ, the dimension of K_C. Despite of the exponential behaviour, using some well known properties of the kernel,

algorithm 1 Kernel and coset representatives computation.

```
Data: A sorted binary code $C$.
Result: The kernel $K_C$ and coset representatives $L = \{v_1, \dots, v_t\}$.
begin
    $K_C \leftarrow \emptyset$
    $L \leftarrow \emptyset$
    for $c \in C$ do
        if $C + c \subseteq C$ then
            $K_C \leftarrow K_C \cup \{c\}$
    $R \leftarrow C \backslash K_C$
    while $R \neq \emptyset$ do
        $v \leftarrow$ First($R$)
        $L \leftarrow L \cup \{v\}$
        $R \leftarrow R \backslash (K_C + v)$
    return $K_C$, $L$
```

in most cases, it is possible to improve Algorithm 1 in order to compute the kernel in a more efficient way.

An improvement of Algorithm 1 is described in Algorithm 2. This new algorithm uses the following three properties of the kernel. Let K' be a subset of the kernel of a binary code C, $K' \subseteq K_C$,

(1) if $k \in K_C$, then $K' + k \subseteq K_C$;

(2) if $c \in C$ and $(C \backslash K') + c \subseteq C$, then $c \in K_C$;

(3) if $c \notin K_C$, then $(K' + c) \cap K_C = \emptyset$.

Therefore, depending on κ, the complexity can be reduced. To analyze Algorithm 2 we study the worst and best case. In the worst case, $\kappa = 0$ and we still need $M^2 \log M$ operations as Algorithm 1. The best case is when C is linear, that is, $C = K_C$. Then, in each iteration the cardinality of the kernel is duplicated. Thus, we need $2^\kappa - 2^i$ additions and searches in each step i, $i \in \{0, \dots, \kappa-1\}$. This means that we need $\sum_{i=0}^{\kappa-1}(2^\kappa - 2^i) = 2^\kappa(\kappa-1)+1$ additions and searches. Hence, the number of operations is $(M(\kappa-1)+1)\log M$, where $M = 2^\kappa$.

A *partial kernel* of the kernel K_C is a linear subcode of K_C. Note that in Algorithm 2 a partial kernel is built in each step. Therefore, in the general case, the number of operations depends strongly on how the partial kernel is growing. If the kernel is small or the partial kernel grows slowly up to the

algorithm 2 Kernel and coset representatives computation.

Data: A sorted binary code C.

Result: The kernel K_C and coset representatives $L = \{v_1, \ldots, v_t\}$.

```
begin
    $K_C \leftarrow \{\mathbf{0}\}$
    $C^* \leftarrow C \backslash \{\mathbf{0}\}$
    $L \leftarrow \emptyset$
    $R \leftarrow \emptyset$
    while $|C^*| > 0$ do
        $c \leftarrow \texttt{First}(C^*)$
        if $C^* + c \subseteq C$ then
            $C^* \leftarrow C^* \backslash (K_C + c)$
            $K_C \leftarrow K_C \cup (K_C + c)$
        else
            $R \leftarrow R \cup (K_C + c)$
            $C^* \leftarrow C^* \backslash (K_C + c)$
    while $R \neq \emptyset$ do
        $v \leftarrow \texttt{First}(R)$
        $L \leftarrow L \cup \{v\}$
        $R \leftarrow R \backslash (K_C + v)$
    return $K_C$, $L$
```

kernel, the number of operations is close to the worst case. Otherwise, the number of operations can be reduced significantly using also the following property given by Proposition 3.1.1.

Proposition 3.1.1. *Let K' be a partial kernel of a binary code C and $L' = \{v'_1, \ldots, v'_{t'}\}$ be the corresponding coset representatives, that is, $C = \bigcup_{i=0}^{t'}(K' + v'_i)$, where $v'_0 = \mathbf{0}$. Then, for each $v'_j \in L'$, $K' + v'_j \subseteq K_C$ if and only if $v'_j + v'_i \in C$ for all $i \in \{1, \ldots, t'\}$.*

Proof. If $K'+v'_j \subseteq K_C$, then $v'_j \in K_C$ and, by the kernel definition, $v'_j+v'_i \in C$ for all $i \in \{1, \ldots, t'\}$. Suppose that $v'_j + v'_i \in C$ for all $i \in \{1, \ldots, t'\}$. To prove that $K' + v'_j \subseteq K_C$, it is enough to show that $v'_j \in K_C$. For any $c \in C$, there exist $k' \in K'$ and $v'_i \in L'$ such that $c = k' + v'_i$. Then, $v'_j + c = v'_j + k' + v'_i \in K' + v'_j + v'_i \subseteq C$, since $K' \subseteq K_C$ and $v'_j + v'_i \in C$. $\square$

Note that if C is binary nonlinear and $M = 2^r$ ($r \geq 2$), then $|K_C| \leq 2^{r-2}$; and if $M = 2^r \cdot s$ ($r \geq 0$, $s \geq 3$ odd), then $|K_C| \leq 2^r$. Hence, in Algorithm 2, when $M = 2^r$ and the dimension of a partial kernel K' is $r-1$, the code C is linear, so $K_C = C$; and when $M = 2^r \cdot s$ and the dimension of a partial kernel K' is r, $K_C = K'$.

For large M, the computation of the kernel and coset representatives of a binary nonlinear code C using Algorithm 2 can still be inefficient. However, some well known code constructions allow to compute the kernel and coset representatives of new codes in a very efficient way, as can be seen in Section 3.2.

3.1.2 Representation of q-ary nonlinear codes

The representation for binary nonlinear codes can be generalized into q-ary nonlinear codes with some small modifications. As for binary nonlinear codes, we also have that any q-ary nonlinear code can be represented as a union of cosets of a linear subcode of C. The *kernel* of a q-ary code C is defined as $K_C = \{x \in C : \lambda x + C = C, \ \ \forall \lambda \in \mathbb{F}_q\}$ [28]. Like in the binary case, since $\mathbf{0} \in C$, K_C is a linear subcode of C and κ denotes its dimension. In general, C can be written as a union of cosets of K_C, and K_C is the largest such linear code for which this is true [28]. Therefore,

$$C = \bigcup_{i=0}^{t} \Big(K_C + v_i \Big), \tag{3.2}$$

algorithm 3 Kernel and coset representatives computation.

```
Data: A sorted $q$-ary code $C$.
Result: The kernel $K_C$ and coset representatives $L = \{v_1, \dots, v_t\}$.
begin
    $K_C \leftarrow \emptyset$
    $L \leftarrow \emptyset$
    for $c \in C$ do
        if $\{C + \lambda c : \lambda \in \mathbb{F}_q \backslash \{0\}\} \subseteq C$ then
            $K_C \leftarrow K_C \cup \{\lambda c : \lambda \in \mathbb{F}_q\}$
    $R \leftarrow C \backslash K_C$
    while $R \neq \emptyset$ do
        $v \leftarrow$ First($R$)
        $L \leftarrow L \cup \{v\}$
        $R \leftarrow R \backslash \{K_C + v\}$
    return $K_C$, $L$
```

where $v_0 = \mathbf{0}$, $t + 1 = M/q^\kappa$, $M = |C|$, and $v_1, \dots, v_t$ are representatives of the cosets of K_C.

As for binary nonlinear codes, any q-ary nonlinear code C can be represented by a matrix $\binom{G}{L}$, where G is a generator matrix of K_C of size $\kappa \times n$ and $L = \{v_1, \dots, v_t\}$ is a set of coset representatives considered as the matrix where in the rows there are the coset representatives. Again, since the kernel takes up a memory space of order $O(n\kappa)$, the kernel plus the t coset representatives take up a memory space of order $O(n(\kappa + t))$. Moreover, another matrix, the parity-check system and the super dual of a q-ary code can also be defined in the same way as for binary codes.

Algorithm 3 describes a straightforward algorithm to compute the kernel and coset representatives of a q-ary code C, by using just the definition of the kernel. The main difference with Algorithm 1 lies in the fact that if $c \in K_C$, then $\lambda c \in K_C$ for all $\lambda \in \mathbb{F}_q$. Again, if we assume that the codewords are sorted, then $(q-1)M^2 \log M$ operations (additions and searches) would need to be executed. Since $M = q^\kappa(t + 1)$, Algorithm 3 is also exponential in κ, the dimension of K_C.

Algorithm 3 can be improved by using the same properties used to improve Algorithm 1 into Algorithm 2. Let K' be a subset of the kernel of a q-ary code C, $K' \subseteq K_C$,

(1) if $k \in K_C$, then $K' + \lambda k \subseteq K_C$ for all $\lambda \in \mathbb{F}_q$;

(2) if $c \in C$ and $(C \backslash K') + \lambda c \subseteq C$ for all $\lambda \in \mathbb{F}_q$, then $c \in K_C$;

(3) if $c \notin K_C$, then $(K' + \lambda c) \cap K_C = \emptyset$ for all $\lambda \in \mathbb{F}_q \backslash \{0\}$.

Therefore, depending on κ, the complexity can be reduced. These improvements have been used to describe Algorithm 4. The analysis of this new algorithm is similar to the analysis of Algorithm 2. In the worst case, $\kappa = 0$ and we still need $(q-1)M^2 \log M$ operations as Algorithm 3. The best case is when C is linear, that is, $C = K_C$. Then, in each iteration the cardinality of the kernel grows q times. Thus, we need $(q-1)(q^\kappa - q^i)$ additions and searches in each step i, $i \in \{0, \ldots, \kappa - 1\}$. This means that we need $\sum_{i=0}^{\kappa-1}(q-1)(q^\kappa - q^i) = q^\kappa(\kappa(q-1)-1)+1$ additions and searches. Hence, the number of operations is $(M(\kappa(q-1)-1)+1)\log M$, where $M = q^\kappa$.

As in Algorithm 2 for binary codes, in Algorithm 4, a partial kernel is built in each step. Therefore, in the general case, the number of operations depends strongly on how the partial kernel is growing. Also, in some cases, the algorithm can be improved by using Proposition 3.1.2, which is a generalization of Proposition 3.1.1.

Proposition 3.1.2. *Let K' be a partial kernel of a q-ary code C and $L' = \{v'_1, \ldots, v'_{t'}\}$ be the corresponding coset representatives, that is, $C = \bigcup_{i=0}^{t'}(K' + v'_i)$, where $v'_0 = \mathbf{0}$. Then, for each $v'_j \in L'$, $K' + v'_j \subseteq K_C$ if and only if $\lambda v'_j + v'_i \in C$ for all $\lambda \in \mathbb{F}_q$ and $i \in \{1, \ldots, t'\}$.*

Proof. If $K' + v'_j \subseteq K_C$, then $v'_j \in K_C$ and, by the kernel definition, $\lambda v'_j + v'_i \in C$ for all $\lambda \in \mathbb{F}_q$ and $i \in \{1, \ldots, t'\}$.

Suppose that $\lambda v'_j + v'_i \in C$ for all $\lambda \in \mathbb{F}_q$ and $i \in \{1, \ldots, t'\}$. To prove that $K' + v'_j \subseteq K_C$, it is enough to show that $v'_j \in K_C$. For any $c \in C$, there exist $k' \in K'$ and $v'_i \in L'$ such that $c = k' + v'_i$. Then, $\lambda v'_j + c = \lambda v'_j + k' + v'_i \in K' + \lambda v'_j + v'_i \subseteq C$, since $K' \subseteq K_C$ and $\lambda v'_j + v'_i \in C$. $\square$

Note that if $M = q^r \cdot s$ ($r \geq 0$, $s \geq 1$, $\gcd(s, q) = 1$), then $|K_C| \leq q^r$; so when the dimension of a partial kernel K' is r, $K_C = K'$. For the binary case, that is when $q = 2$, as have seen in Subsection 3.1.1, we can improve the bound on the cardinality of the kernel, K_C. If C is binary nonlinear and $M = 2^r$ ($r \geq 2$), then $|K_C| \leq 2^{r-2}$. Hence, in Algorithm 4, when $M = 2^r$ and the dimension of a partial kernel K' is $r-1$, the code C is linear, so $K_C = C$.

Also note that when $q = p^t$ ($t \geq 2$, p prime) Algorithm 4 can be improved by considering only the t generators of the vector space $\mathbb{F}_q$ of dimension t over $\mathbb{F}_p$ for λ, instead of all elements in $\mathbb{F}_q$.

algorithm 4 Kernel and coset representatives computation.

Data: A sorted q-ary code C.
Result: The kernel K_C and coset representatives $L = \{v_1, \dots, v_t\}$.

```
begin
    $K_C \leftarrow \{\mathbf{0}\}$
    $C^* \leftarrow C \backslash \{\mathbf{0}\}$
    $L \leftarrow \emptyset$
    $R \leftarrow \emptyset$
    while $|C^*| > 0$ do
        $c \leftarrow \texttt{First}(C^*)$
        if $\{C^* + \lambda c : \lambda \in \mathbb{F}_q \backslash \{0\}\} \subseteq C$ then
            $C^* \leftarrow C^* \backslash \{K_C + \lambda c : \lambda \in \mathbb{F}_q \backslash \{0\}\}$
            $K_C \leftarrow \{K_C + \lambda c : \lambda \in \mathbb{F}_q\}$
        else
            for $\lambda \in \mathbb{F}_q \backslash \{0\}$ such that $\lambda c \in C$ do
                $R \leftarrow R \cup \{K_C + \lambda c\}$
                $C^* \leftarrow C^* \backslash \{K_C + \lambda c\}$
    while $R \neq \emptyset$ do
        $v \leftarrow \texttt{First}(R)$
        $L \leftarrow L \cup \{v\}$
        $R \leftarrow R \backslash \{K_C + v\}$
    return $K_C$, $L$
```

For large M, the computation of the kernel and coset representatives of a q-ary code C using Algorithm 4 can still be inefficient. However, some well known code constructions allow to compute the kernel and coset representatives of new codes in a very efficient way, as can be seen in Section 3.2.

3.2 Constructions of nonlinear codes

Using the coset representation given in Section 3.1, rather than using the list of all codewords, we can manipulate and construct new q-ary nonlinear codes from old ones in a more efficient way. Specifically, in this section, we show how to establish the equality and inclusion of two given nonlinear codes from their kernels and coset representatives, and how to compute the kernel and coset representatives of new codes (union, intersection, extended code, punctured code, shortened code, direct sum, Plotkin sum) from given ones, which are already represented in this way. Note that all these results are written to be implemented easily as algorithms.

3.2.1 Properties of nonlinear codes

Let $col(S)$ denote the set of columns of the matrix S.

Proposition 3.2.1. *[17] Let C be a q-ary code of length n with parity-check system $(H\ S)$. Then, $c \in C$ if and only if $Hc^T \in \{\mathbf{0}\} \cup col(S)$.*

Proposition 3.2.2. *[17] Let C be a q-ary code of length n with rank ϱ, dimension of the kernel κ and parity-check system $(H\ S)$. Then, $\varrho = n - rank(H) + rank(S)$ and $\kappa = n - rank(H)$.*

Let C_1 and C_2 be two q-ary codes of length n_1 and n_2, respectively. Let $(H_1\ S_1)$ and $(H_2\ S_2)$ be the parity-check systems of C_1 and C_2, respectively. The matrices H_1 and H_2 are the generator matrices of the dual codes $K_{C_1}^{\perp}$ and $K_{C_2}^{\perp}$, and κ_1 and κ_2 the dimension of the kernel of C_1 and C_2, respectively. The coset representatives for C_1 and C_2 are the sets $\{v_1, \dots, v_{t_1}\}$ and $\{w_1, \dots, w_{t_2}\}$, which give us the matrices $S_1 = (H_1 v_1^T\ H_1 v_2^T\ \dots\ H_1 v_{t_1}^T)$ and $S_2 = (H_2 w_1^T\ H_2 w_2^T\ \dots\ H_2 w_{t_2}^T)$, respectively. Finally, the super duals of C_1 and C_2 are the linear codes generated by $(H_1\ S_1)$ and $(H_2\ S_2)$, respectively.

Proposition 3.2.3 (Equality)**.** *Let C_1 and C_2 be two q-ary codes of length n. Then, $C_1 = C_2$ if and only if $K_{C_1} = K_{C_2}$, $t_1 = t_2$, and $H_2 v_i^T \in col(S_2)$ for all $i \in \{1, \dots, t_1\}$.*

Proof. It is clear that if $C_1 = C_2$, then $K_{C_1} = K_{C_2}$, $t_1 = t_2$ and $v_i \in C_2 \backslash K_{C_2}$ for all $i \in \{1, \dots, t_1\}$. By Proposition 3.2.1, $v_i \in C_2 \backslash K_{C_2}$ if and only if $H_2 v_i^T \in col(S_2)$.

On the other hand, if $K_{C_1} = K_{C_2}$ and $t_1 = t_2$, then $|C_1| = |C_2|$. Moreover, given a codeword $c \in C_1$, $c = k + v_i$ for some $i \in \{0, 1, \dots, t_1\}$ and $k \in K_{C_1}$, where $v_0 = \mathbf{0}$. If $i = 0$, then $c \in K_{C_1} = K_{C_2} \subseteq C_2$. If $i \in \{1, \dots, t_1\}$, then $H_2 c^T = H_2(k+v_i)^T = H_2 k^T + H_2 v_i^T = H_2 v_i^T \in col(S_2)$, since $k \in K_{C_1} = K_{C_2}$. Therefore, by Proposition 3.2.1, we have that $c \in C_2$. Since $C_1 \subseteq C_2$ and $|C_1| = |C_2|$, we obtain that $C_1 = C_2$. □

Proposition 3.2.4 (Inclusion)**.** *Let C_1 and C_2 be two q-ary codes of length n. Let $K = K_{C_1} \cap K_{C_2}$ of dimension κ and $K_{C_1} = \bigcup_{j=0}^{h_1} (K + x_j)$, where $h_1 = q^{\kappa_1 - \kappa} - 1$. Then, $C_1 \subseteq C_2$ if and only if $H_2(x_j + v_i)^T \in \{\mathbf{0}\} \cup col(S_2)$, for all $i \in \{0, 1, \dots, t_1\}$ and $j \in \{0, 1, \dots, h_1\}$, where $v_0 = x_0 = \mathbf{0}$.*

Proof. Note that $C_1 = \bigcup_{i=0}^{t_1} (K_{C_1} + v_i) = \bigcup_{i=0}^{t_1} \bigcup_{j=0}^{h_1} (K + x_j + v_i)$. It is clear that if $C_1 \subseteq C_2$, then $x_j + v_i \in C_2$, which is equivalent to $H_2(x_j + v_i)^T \in \{\mathbf{0}\} \cup col(S_2)$ for all $i \in \{0, 1, \dots, t_1\}$ and $j \in \{0, 1, \dots, h_1\}$, by Proposition 3.2.1.

On the other hand, given a codeword $c \in C_1$, $c = k + x_j + v_i$ for some $i \in \{0, 1, \dots, t_1\}$, $j \in \{0, 1, \dots, h_1\}$ and $k \in K$. Thus, $H_2 c^T = H_2(k + x_j + v_i)^T = H_2 k^T + H_2(x_j + v_i)^T = H_2(x_j + v_i)^T \in \{\mathbf{0}\} \cup col(S_2)$, since $k \in K \subseteq K_{C_2}$. Finally, by Proposition 3.2.1, we can assure that $c \in C_2$, so $C_1 \subseteq C_2$. □

Proposition 3.2.5 (Intersection)**.** *Let C_1 and C_2 be two q-ary codes of length n. Let $K = K_{C_1} \cap K_{C_2}$ of dimension κ and $K_{C_1} = \bigcup_{j=0}^{h_1} (K + x_j)$ and $K_{C_2} = \bigcup_{j=0}^{h_2} (K + y_j)$, where $h_1 = q^{\kappa_1 - \kappa} - 1$, $h_2 = q^{\kappa_2 - \kappa} - 1$ and $x_0 = y_0 = \mathbf{0}$. Let $C = C_1 \cap C_2$. Then, $K \subseteq K_C$ and $C = \bigcup_{v \in L_I} (K + v)$, where $L_I = \{x_j + v_i :$ $j \in \{0, 1, \dots, h_1\},\ i \in \{0, 1, \dots, t_1\}$ and $H_2(x_j + v_i)^T \in \{\mathbf{0}\} \cup col(S_2)\}$.*

Proof. First, we show that $K \subseteq K_C$. Given a $k \in K = K_{C_1} \cap K_{C_2}$, for any $v \in C = C_1 \cap C_2$, we have that $k + v \in C_1$ since $v \in C_1$ and $k \in K_{C_1}$, and equivalently, we have that $k + v \in C_2$ since $v \in C_2$ and $k \in K_{C_1}$. Therefore, $k + v \in C$ for any $v \in C$, and we obtain that $k \in K_C$.

Note that $C_1 = \bigcup_{i=0}^{t_1} (K_{C_1} + v_i) = \bigcup_{i=0}^{t_1} \bigcup_{j=0}^{h_1} (K + x_j + v_i)$. For any $j \in \{0, 1, \dots, h_1\}$ and $i \in \{0, 1, \dots, t_1\}$, $K + x_j + v_i \subseteq C_2$ if and only if $x_j + v_i \in C_2$, since $K \subseteq K_{C_2}$. By Proposition 3.2.1, the condition $x_j + v_i \in C_2$ is equivalent to $H_2(x_j + v_i)^T \in \{\mathbf{0}\} \cup col(S_2)$. Therefore, the result follows. □

Proposition 3.2.6 (Union). *Let C_1 and C_2 be two q-ary codes of length n. Let $K = K_{C_1} \cap K_{C_2}$ of dimension κ and $K_{C_1} = \bigcup_{j=0}^{h_1}(K + x_j)$ and $K_{C_2} = \bigcup_{j=0}^{h_2}(K + y_j)$, where $h_1 = q^{\kappa_1 - \kappa} - 1$, $h_2 = q^{\kappa_2 - \kappa} - 1$ and $x_0 = y_0 = \mathbf{0}$. Let $C = C_1 \cup C_2$. Then, $K \subseteq K_C$ and $C = \bigcup_{v \in L_U}(K + v)$, where $L_U = \{x_j + v_i \ : \ j \in \{0, 1, \dots, h_1\},\ i \in \{0, 1, \dots, t_1\}\} \backslash L_I \cup \{y_j + w_i \ : \ j \in \{0, 1, \dots, h_2\}, i \in \{0, 1, \dots, t_2\}\}$.*

Proof. Using the same arguments as in Proposition 3.2.5, we can see that $K \subseteq K_C$. We have that $C = C_1 \cup C_2 = \bigcup_{i=0}^{t_1} \bigcup_{j=0}^{h_1}(K + x_j + v_i) \cup \bigcup_{i=0}^{t_2} \bigcup_{j=0}^{h_2}(K + y_j + w_i)$. Again, by the same arguments as in the proof of Proposition 3.2.5, $C = \bigcup_{v \in L_U}(K + v)$, where $L_U = \{x_j + v_i \ : \ j \in \{0, 1, \dots, h_1\},\ i \in \{0, 1, \dots, t_1\}\} \backslash L_I \cup \{y_j + w_i \ : \ j \in \{0, 1, \dots, h_2\}, i \in \{0, 1, \dots, t_2\}\}$. □

3.2.2 Extended codes

Let C be a q-ary code of length n. The *extended code* of C, denoted by $\widehat{C}$, is defined as the set of codewords constructed by adding to each codeword of C a coordinate so that the sum of all coordinates is 0, that is,

$$\widehat{C} = \{(x_1, x_2, \dots, x_{n+1}) \in \mathbb{F}_q^{n+1} \ : \ (x_1, x_1, \dots, x_n) \in C \text{ with } \sum_{i=1}^{n+1} x_i = 0\}.$$

The $(n + 1)$th coordinate in the above construction is called *overall parity-check coordinate*, or just *parity coordinate* if C is a binary code.

Proposition 3.2.7 (Extended Code). *Let C be a q-ary code of length n with kernel K_C and t coset representatives given by the set $L = \{v_1, \dots, v_t\}$. Then, the extended q-ary code $\widehat{C}$ has kernel $\widehat{K_C}$ and t coset representatives given by the set $\{\hat{v}_1, \dots, \hat{v}_t\}$, where $\hat{v}_i$ is v_i after adding an overall parity-check coordinate for $i \in \{1, \dots, t\}$.*

Proof. Obviously, the extended code $\widehat{C}$ can be written as $\widehat{C} = \bigcup_{i=0}^{t}(\widehat{K_C} + \hat{v}_i)$, where $v_0 = \mathbf{0}$ and $\hat{v}_i$ is v_i after adding an overall parity-check coordinate for $i \in \{0, \dots, t\}$. Therefore, $\widehat{K_C} \subseteq K(\widehat{C})$, and we only need to prove that $K(\widehat{C}) \subseteq \widehat{K_C}$.

Let $\hat{k} = (k_1, k_2, \dots, k_n, -\sum_{j=1}^{n} k_j) \in K(\widehat{C}) \subseteq \widehat{C}$. For any $\lambda \in \mathbb{F}_q$ and $\hat{c} = (c_1, c_2, \dots, c_n, -\sum_{j=1}^{n} c_j) \in \widehat{C}$, we have that $\lambda\hat{k} + \hat{c} = (\lambda k_1 + c_1, \lambda k_2 + c_2, \dots, \lambda k_n + c_n, -\sum_{j=1}^{n}(\lambda k_j + c_j)) \in \widehat{C}$. Since $\widehat{C}$ is the extended code of C, $k = (k_1, k_2, \dots, k_n) \in C$, $c = (c_1, c_2, \dots, c_n) \in C$ and $\lambda k + c = (\lambda k_1 + c_1, \lambda k_2 + c_2, \dots, \lambda k_n + c_n) \in C$ for any $c \in C$ and $\lambda \in \mathbb{F}_q$, so $k \in K_C$, which means that $\hat{k} \in \widehat{K_C}$ and $K(\widehat{C}) \subseteq \widehat{K_C}$. □

Example 3.2.8. *Let C be the $(7, 24, 2)$ binary nonlinear code with kernel K generated by G and 2 coset representatives $v_1 = (0101011)$ and $v_2 = (1010100)$.*

$$G = \begin{pmatrix} 1001111 \\ 0100101 \\ 0011100 \end{pmatrix} \qquad \widehat{G} = \begin{pmatrix} 10011111 \\ 01001011 \\ 00111001 \end{pmatrix}$$

The extended code $\widehat{C}$ of C can be obtained by adding an overall parity-check coordinate to the kernel K and coset representatives v_1, v_2. The kernel of the extended code $\widehat{C}$ is the extended code of the kernel, $\widehat{K}$, which is generated by the matrix $\widehat{G}$; and the coset representatives of the extended code are $\widehat{v_1} = (01010110)$ and $\widehat{v_2} = (10101001)$. Therefore, $\widehat{C} = \widehat{K} \cup (\widehat{K} + \widehat{v_1}) \cup (\widehat{K} + \widehat{v_2})$. Note that the resulting code is a $(8, 24, 2)$ binary nonlinear code.

If C is an $(n, |C|, d)$ binary code, then the extended code $\widehat{C}$ contains only even weight codewords and is an $(n+1, |C|, \widehat{d})$ binary code, where $\widehat{d}$ equals d if d is even and equals $d+1$ if d is odd. It is known that this is true if C is linear [19], and it can be easily generalized into binary nonlinear codes. Note that if the distance between two codewords in C is even, after adding a parity coordinate, both codewords are at the same distance, and if the distance between them is odd, the distance increases by 1, after adding a parity coordinate.

In general, for an $(n, |C|, d)$ q-ary code C, denote by d_e the minimum distance between pairs of codewords of C having the same overall parity-check coordinate and by d_o the minimum distance between pairs of codewords of C having different overall parity-check coordinate. Therefore, $d = \min\{d_e, d_o\}$. If $d_e \leq d_o$, then $\widehat{C}$ has minimum distance $\widehat{d} = d_e = d$; otherwise $\widehat{d} = d_o + 1 = d + 1$. For q-ary linear codes this result is given for example in [19]. The generalization to q-ary nonlinear codes is straightforward considering the minimum distances instead of the minimum weights of codewords of C.

3.2.3 Punctured codes

Let C be a q-ary code of length n. The *punctured code* of C in the jth coordinate, denoted by C^j, is the code consisting of the codewords of C after deleting the jth coordinate.

Proposition 3.2.9 (Punctured Code). *Let C be an $(n, |C|, d)$ q-ary code with kernel K_C and t coset representatives given by the set $L = \{v_1, \ldots, v_t\}$.*

Let K^j be the punctured code of K_C in the jth coordinate. Then, the punctured code C^j in the jth coordinate is $C^j = \bigcup_{v \in L^j}(K^j + v)$, where $L^j \subseteq \{\mathbf{0}, v_1^j, \ldots, v_t^j\}$ and v_i^j is v_i after deleting the jth coordinate for $i \in \{1, \ldots, t\}$.

(i) If $d > 1$, C^j is an $(n-1, |C|, d')$ q-ary code, where $d' = d-1$ if C has two codewords at distance d which differ at the jth coordinate and $d' = d$ otherwise.

(ii) If $d = 1$, C^j is an $(n-1, |C|, 1)$ q-ary code if the codewords in C at distance 1 do not differ at the jth coordinate; otherwise, C^j is an $(n-1, |C^j|, d')$ q-ary code with $d' \geq 1$ and $|C^j| < |C|$.

Proof. Let v^j be the vector v after deleting the jth coordinate. Given any codeword $c \in C$, $c = k + v_i$ for some $k \in K_C$ and $i \in \{0, 1, \ldots, t\}$. Since $c^j = k^j + v_i^j$, $C^j \subseteq \bigcup_{v \in L^j}(K^j + v)$, where $L^j = \{\mathbf{0}, v_1^j, \ldots, v_t^j\}$.

If $d > 1$, or $d = 1$ but the codewords at distance 1 do not differ at the jth coordinate, then $|C^j| = |C|$. Since $C^j \subseteq \bigcup_{v \in L^j}(K^j + v)$ and $|C^j| = |C| = |\bigcup_{v \in L^j}(K^j + v)|$, we obtain that $C^j = \bigcup_{v \in L^j}(K^j + v)$. Moreover, it is clear that if $d > 1$, C^j is an $(n-1, |C|, d')$ q-ary code, where $d' = d - 1$ if C has two codewords at distance d which differ at the jth coordinate and $d' = d$ otherwise.

Finally, if there exist $c, u \in C$ such that they only differ at the jth coordinate, then $c - u = \lambda e_j$, where $\lambda \in \mathbb{F}_q \backslash \{0\}$ and e_j is the vector with 1 in the jth coordinate and zero elsewhere. Now, we have two possibilities. Firstly, if $\lambda e_j \in K_C$, then c and u are in the same coset, and the same happens with any such pair c, u since K_C is linear. In this case, K^j has dimension $\kappa - 1$, and the result follows with $L^j = \{\mathbf{0}, v_1^j, \ldots, v_t^j\}$. Secondly, if $\lambda e_j \notin K_C$, then c and u are in different cosets, that is, $c \notin K_C + u$. However, after deleting the jth coordinate, $K^j + c^j = K^j + u^j$ (or equivalently, $c^j \in K^j + u^j$) and the result follows with $L^j \subsetneq \{\mathbf{0}, v_1^j, \ldots, v_t^j\}$. Note that in both cases we have that $d' \geq 1$ and $|C^j| < |C|$. $\square$

Example 3.2.10. *Let C be the $(7, 24, 2)$ binary nonlinear code with kernel K generated by G and 2 coset representatives $v_1 = (0101011)$ and $v_2 = (1010100)$.*

$$G = \begin{pmatrix} 1001111 \\ 0100101 \\ 0011100 \end{pmatrix} \qquad G^5 = \begin{pmatrix} 100111 \\ 010001 \\ 001100 \end{pmatrix}$$

By puncturing the kernel K and coset representatives of C, we can obtain the punctured code C^5 as $C^5 = K^5 \cup (K^5 + v_1^5) \cup (K^5 + v_2^5)$, where the *punctured code of the kernel, K^5, is generated by the matrix G^5; and the coset representatives are $v_1^5 = (010111)$ and $v_2^5 = (101000)$. Note that the punctured code C^5 is a $(6, 24, 2)$ binary nonlinear code. In Table 3.2, we can see that we obtain the same codewords in C^5 as puncturing directly from the list of codewords of C.*

3.2.4 Shortened codes

Let C be a q-ary code of length n. The *shortened code* of C in the jth coordinate, denoted by C_0^j, is the code consisting of the codewords of C having 0 in the jth coordinate and deleting this coordinate.

Proposition 3.2.11 (Shortened Code)**.** *Let C be an $(n, |C|, d)$ q-ary code with kernel K_C and t coset representatives given by the set $L = \{v_1, \dots, v_t\}$. Let K_0^j be the shortened code of K_C in the jth coordinate. Then, the shortened code C_0^j in the jth coordinate is $C_0^j = \bigcup_{v\in L^j}(K_0^j + v)$, where $L^j = \{\mathbf{0}\} \cup \{v_i^j : i \in I\}$, $I = \{i \in \{1, \dots, t\}$: there exists $v_i' \in K_C + v_i$ such that has 0 in the jth coordinate$\}$ and v_i^j is v_i' after deleting the jth coordinate for $i \in I$. Moreover, C_0^j is an $(n-1, |C_0^j|, d')$ q-ary code with $d' \geq d$ and $|C_0^j| \leq |C|$.*

Proof. Let $C_0 \subseteq C$ and $K_0 \subseteq K_C$ be the subsets of C and K_C, respectively, containing the codewords having 0 in the jth coordinate.

If $K_0 \subsetneq K_C$, then $I = \{1, \dots, t\}$. Therefore, we have that $\bigcup_{v\in L^j}(K_0^j+v) \subseteq C_0^j$, where $L^j = \{\mathbf{0}, v_1^j, \dots, v_t^j\}$. Moreover, for any $c_0 \in C_0 \subseteq C$, since $c_0 \in C$, there exist $k \in K_C$ and $i \in \{1, \dots, t\}$ such that $c_0 = k + v_i'$. Since c_0 and v_i' have 0 in the jth coordinate, we have that $k \in K_0$. Therefore, after deleting the jth coordinate, $C_0^j = \bigcup_{v\in L^j}(K_0^j + v)$.

If $K_0 = K_C$, then any $k \in K_C$ has 0 in the jth coordinate. In this case, if v_i has a nonzero entry in the jth coordinate, does not exist any $v_i' \in K_C + v_i$ such that has 0 in the jth coordinate. Therefore, $C_0^j = \bigcup_{v\in L^j}(K_0^j + v)$, where $L^j = \{\mathbf{0}\} \cup \{v_i^j : i \in I\}$, $I = \{i \in \{1, \dots, t\}$: there exists $v_i' \in K_C + v_i$ such that has 0 in the jth coordinate$\}$ and v_i^j is v_i' after deleting the jth coordinate for $i \in I$. Note that, in this case, the number of cosets in C_0^j may be smaller than the number of cosets t in C.

Finally, it is straightforward to see that $d' \geq d$ and $|C_0^j| \leq |C|$. □

Example 3.2.12. *Let C be the $(7, 24, 2)$ binary nonlinear code with kernel K generated by G and 2 coset representatives $v_1 = (0101011)$ and $v_2 =$*

	C		C^5		
	(0000000),		(000000),		
	(1101010),		(110110),		
	(1001111),		(100111),		
$K =$	(1010011),	$\rightarrow$	(101011),	$\leftarrow$	K^5
	(1110110),		(111010),		
	(0111001),		(011101),		
	(0100101),		(010001),		
	(0011100).		(001100).		
	(0101011),		(010111),		
	(1000001),		(100001),		
	(1100100),		(110000),		
	(1111000),		(111100),		
$K + v_1 =$	(1011101),	$\rightarrow$	(101101),	$\leftarrow$	$K^5 + v_1^5$
	(0010010),		(001010),		
	(0001110),		(000110),		
	(0110111).		(011011).		
	(1010100),		(101000),		
	(0111110),		(011110),		
	(0011011),		(001111),		
	(0000111),		(000011),		
$K + v_2 =$	(0100010),	$\rightarrow$	(010010),	$\leftarrow$	$K^5 + v_2^5$
	(1101101),		(110101),		
	(1110001),		(111001),		
	(1001000).		(100100).		

Table 3.2: An example of puncturing a nonlinear code.

(1010100).

$$G = \begin{pmatrix} 1001111 \\ 0100101 \\ 0011100 \end{pmatrix} \qquad G_0^5 = \begin{pmatrix} 101011 \\ 011101 \end{pmatrix}$$

In order to obtain the shortened code C_0^5, we shorten the kernel K and an appropriate set of coset representatives of C. The shortened code of the kernel, K_0^5, is generated by the matrix G_0^5. Since $v_2 = (1010100)$ does not have a zero at the 5th coordinate, we choose another coset representative of the same coset, for example $v_2' = (0011011) \in K + v_2$. Then, a set of coset representatives for C_0^5 is $\{v_1^5, v_2^5\}$, where $v_1^5 = (010111)$ and $v_2^5 = (001111)$. Therefore, the shortened code $C_0^5 = K_0^5 \cup (K_0^5 + v_1^5) \cup (K_0^5 + v_1^5)$ is a $(6, 12, 2)$ binary nonlinear code.

3.2.5 Direct sum construction

Another way to construct new codes is to combine two codes together in a proper way. The most known such constructions are called *direct sum* and *Plotkin sum*. In this subsection, we focus on the direct sum construction and, in the next subsection, on the Plotkin sum construction.

For $i \in \{1, 2\}$, let C_i be a q-ary code of length n_i. The direct sum of C_1 and C_2 is the q-ary code $C_1 \oplus C_2 = \{(c_1|c_2) : c_1 \in C_1, c_2 \in C_2\}$.

Proposition 3.2.13 (Direct Sum). *Let C_1 and C_2 be two $(n_1, |C_1|, d_1)$ and $(n_2, |C_2|, d_2)$ q-ary codes with kernels K_{C_1} and K_{C_2}, and coset representatives $L_1 = \{v_1, \ldots, v_{t_1}\}$ and $L_2 = \{w_1, \ldots, w_{t_2}\}$, respectively. The direct sum $C_1 \oplus C_2$ is the $(n_1 + n_2, |C_1| \cdot |C_2|, \min\{d_1, d_2\})$ q-ary code with kernel $K_{C_1} \oplus K_{C_2}$, and $(t_1 + 1)(t_2 + 1) - 1$ coset representatives given by the set $\{(v|w) : v \in L_1 \cup \{\mathbf{0}\}, w \in L_2 \cup \{\mathbf{0}\}\} \backslash \{(\mathbf{0}, \mathbf{0})\}$.*

Proof. Firstly, we prove that $K_{C_1} \oplus K_{C_2}$ is the kernel of $C_1 \oplus C_2$. For $k_\oplus = (k_1|k_2) \in K_{C_1 \oplus C_2}$ and any $c_\oplus = (c_1|c_2) \in C_1 \oplus C_2$, where $k_1, c_1 \in C_1$ and $k_2, c_2 \in C_2$, we have $\lambda k_\oplus + c_\oplus = (\lambda k_1 + c_1 | \lambda k_2 + c_2) \in C_1 \oplus C_2$ for all $\lambda \in \mathbb{F}_q$. Since $\lambda k_1 + c_1 \in C_1$ and $\lambda k_2 + c_2 \in C_2$, we have that $k_1 \in K_{C_1}$ and $k_2 \in K_{C_2}$, so $K_{C_1 \oplus C_2} \subseteq K_{C_1} \oplus K_{C_2}$. On the other hand, for any $k_\oplus = (k_1|k_2) \in K_{C_1} \oplus K_{C_2}$ and $c_\oplus = (c_1|c_2) \in C_1 \oplus C_2$, since $k_1 \in K_{C_1}$ and $k_2 \in K_{C_2}$, we have $\lambda k_\oplus + c_\oplus = (\lambda k_1 + c_1 | \lambda k_2 + c_2) \in C_1 \oplus C_2$ for all $\lambda \in \mathbb{F}_q$, so $k_\oplus \in K_{C_1 \oplus C_2}$ and $K_{C_1} \oplus K_{C_2} \subseteq K_{C_1 \oplus C_2}$. Therefore, we have $K_{C_1 \oplus C_2} = K_{C_1} \oplus K_{C_2}$.

For any $c_\oplus \in C_1 \oplus C_2$, $c_\oplus = (k_1 + v | k_2 + w) = (k_1|k_2) + (v|w)$, where $k_1 \in K_{C_1}$, $k_2 \in K_{C_2}, v \in L_1 \cup \{\mathbf{0}\}$ and $w \in L_2 \cup \{\mathbf{0}\}$. Let $L_\oplus = \{(v|w) : v \in$

$L_1 \cup \{\mathbf{0}\}, w \in L_2 \cup \{\mathbf{0}\}\}$. Since $(k_1|k_2) \in K_{C_1} \oplus K_{C_2}$ and $(v|w) \in L_\oplus$, we have that $C_1 \oplus C_2 \subseteq \bigcup_{\ell_\oplus \in L_\oplus}((K_{C_1} \oplus K_{C_2}) + \ell_\oplus)$. On the other hand, for any $\ell_\oplus, \ell'_\oplus \in L_\oplus$ such that $\ell_\oplus \neq \ell'_\oplus$, we need to prove that $\ell_\oplus$ and $\ell'_\oplus$ belong to two different cosets. If two cosets are partly joint, there exists $k_\oplus \in K_{C_1} \oplus K_{C_2}$ such that $k_\oplus + \ell_\oplus = \ell'_\oplus$, which means that $(k_1|k_2) + (v|w) = (v'|w')$. Then $k_1 + v = v'$ and $k_2 + w = w'$, which is impossible because the cosets of C_1 and C_2 are disjoint and $\ell_\oplus \neq \ell'_\oplus$. Therefore, $|\bigcup_{\ell_\oplus \in L_\oplus}((K_{C_1} \oplus K_{C_2}) + \ell_\oplus)| = (t_1+1)(t_2+1) \cdot |K_{C_1} \oplus K_{C_2}| = (t_1+1)(t_2+1) \cdot q^{\kappa_1+\kappa_2} = |C_1| \cdot |C_2| = |C_1 \oplus C_2|$, where κ_1, κ_2 are the dimensions of K_{C_1}, K_{C_2}, respectively. Then, $C_1 \oplus C_2 = \bigcup_{\ell_\oplus \in L_\oplus}((K_{C_1} \oplus K_{C_2}) + \ell_\oplus)$.

Finally, in order to establish the minimum distance of $C_1 \oplus C_2$, let $c_\oplus = (c_1|c_2)$ and $c'_\oplus = (c'_1|c'_2)$, where $c_1, c'_1 \in C_1$, $c_2, c'_2 \in C_2$. Then $d(c_\oplus, c'_\oplus) = d((c_1|c_2), (c'_1|c'_2)) = d(c_1, c'_1) + d(c_2, c'_2)$. If $c_1 = c'_1$, $d(c_\oplus, c'_\oplus) = d(c_2, c'_2) \geq d_2$; if $c_2 = c'_2$, $d(c_\oplus, c'_\oplus) = d(c_1, c'_1) \geq d_1$; and if $c_1 \neq c'_1$, $c_2 \neq c'_2$, $d(c_\oplus, c'_\oplus) \geq d_1 + d_2$. Therefore, the minimum distance of $C_1 \oplus C_2$ is $\min\{d_1, d_2\}$. □

Example 3.2.14. *Let C_1 be the binary nonlinear code of length $n = 7$, minimum distance $d = 2$, kernel K_1 generated by G_1 and 2 coset representatives $v_1 = (0101011)$ and $v_2 = (1010100)$. Let C_2 be the binary nonlinear code of length $n = 5$, minimum distance $d = 1$, kernel K_2 generated by G_2 and 2 coset representatives $w_1 = (00100)$ and $w_2 = (00010)$.*

$$G_1 = \begin{pmatrix} 1001111 \\ 0100101 \\ 0011100 \end{pmatrix} \qquad G_2 = \begin{pmatrix} 10100 \\ 01101 \\ 00011 \end{pmatrix}$$

According to Proposition 3.2.13, the direct sum $C_1 \oplus C_2$ is a $(12, 576, 1)$ binary nonlinear code with kernel $K_1 \oplus K_2$ generated by $G_1 \oplus G_2$ and coset representatives given by L, where

$$G_1 \oplus G_2 = \left(\begin{array}{c|c} 1001111 & 00000 \\ 0100101 & 00000 \\ 0011100 & 00000 \\ \hline 0000000 & 10100 \\ 0000000 & 01101 \\ 0000000 & 00011 \end{array}\right) \quad \text{and} \quad L = \left(\begin{array}{c|c} 0101011 & 00000 \\ 0101011 & 00010 \\ 0101011 & 00100 \\ 1010100 & 00000 \\ 1010100 & 00010 \\ 1010100 & 00100 \\ 0000000 & 00010 \\ 0000000 & 00100 \end{array}\right).$$

3.2.6 Plotkin sum construction

For $i \in \{1,2\}$, let C_i be a q-ary code of length n. The Plotkin sum of C_1 and C_2 is the q-ary code $C_1 \oplus (C_1 + C_2) = \{(c_1|c_1 + c_2) : c_1 \in C_1, c_2 \in C_2\}$.

Proposition 3.2.15 (Plotkin Sum)**.** *[4] Let C_1 and C_2 be two $(n, |C_1|, d_1)$ and $(n, |C_2|, d_2)$ q-ary codes with kernels K_{C_1} and K_{C_2}, and coset representatives $L_1 = \{v_1, \ldots, v_{t_1}\}$ and $L_2 = \{w_1, \ldots, w_{t_2}\}$, respectively. The Plotkin sum $C_1 \oplus (C_1 + C_2)$ is the $(2n, |C_1| \cdot |C_2|, \min\{2d_1, d_2\})$ q-ary code with kernel $K_{C_1} \oplus (K_{C_1} + K_{C_2})$, and $(t_1 + 1)(t_2 + 1) - 1$ coset representatives given by the set $\{(v|v + w) : v \in L_1 \cup \{\mathbf{0}\}, w \in L_2 \cup \{\mathbf{0}\}\} \backslash \{(\mathbf{0}, \mathbf{0})\}$.*

Proof. Straightforward using the same arguments as in Proposition 3.2.13 [4]. □

Example 3.2.16. *Let C_1 be the binary nonlinear code of length $n = 7$, minimum distance $d = 2$, kernel K_1 generated by G_1 and 2 coset representatives $v_1 = (0101011)$ and $v_2 = (1010100)$. Let C_2 be the binary nonlinear code of length $n = 7$, minimum distance $d = 3$, kernel K_2 generated by G_2 and 2 coset representatives $w_1 = (1101110)$ and $w_2 = (1011000)$.*

$$G_1 = \begin{pmatrix} 1001111 \\ 0100101 \\ 0011100 \end{pmatrix} \qquad G_2 = \begin{pmatrix} 1010110 \\ 0110101 \end{pmatrix}$$

According to Proposition 3.2.15, the Plotkin sum $C_1 \oplus (C_1 + C_2)$ is a $(14, 288, 3)$ binary nonlinear code with kernel $K^{\oplus} = K_1 \oplus (K_1 + K_2)$ generated by $G^{\oplus}$ and coset representatives given by $L^{\oplus}$, where

$$G^{\oplus} = \left(\begin{array}{c|c} 1001111 & 1001111 \\ 0100101 & 0100101 \\ 0011100 & 0011100 \\ \hline 0000000 & 1010110 \\ 0000000 & 0110101 \end{array}\right) \quad \text{and} \quad L^{\oplus} = \left(\begin{array}{c|c} 0101011 & 0101011 \\ 0101011 & 1000101 \\ 0101011 & 1110011 \\ 0000000 & 1101110 \\ 0000000 & 1011000 \\ 1010100 & 1010100 \\ 1010100 & 0111010 \\ 1010100 & 0001100 \end{array}\right).$$

Note that we can obtain the kernel and coset representatives of an extended code directly from the kernel and coset representatives of the code. The same happens for the direct sum and Plotkin sum constructions. For all other constructions, we obtain a partial kernel and the corresponding coset representatives. Although we can not assure which are the final kernel and coset representatives in these cases, we can speed up the kernel computation by starting from a partial kernel and using the algorithms shown in Section 3.1.

Chapter 4

Minimum distance computation

In general, computing the minimum weight and minimum distance of a code is usually computationally difficult. Actually, it has been proven to be NP-hard [37]. For linear codes, the Brouwer-Zimmermann's algorithm is the best known enumerative algorithm to compute the minimum weight (see [39] or Section 2.7). For systematic nonlinear codes, an algorithm based on Gröbner bases to compute the distance distribution is described in [15]. However, in general, for nonlinear codes, as far as we know, there is not any algorithm to compute these parameters comparable to Brouwer-Zimmermann's algorithm for linear codes.

In this chapter, using the coset representation given in Section 3.1, we describe new algorithms to compute the minimum weight and minimum distance of an arbitrary q-ary nonlinear code. Specifically, we present new algorithms based on computing the minimum weight of linear subcodes, using the known Brouwer-Zimmermann's algorithm. Note that, for nonlinear codes, the minimum weight and minimum distance do not coincide like for linear code. Moreover, we study the performance of these algorithms by giving an estimation of the number of enumerated codewords needed in the computations. In this sense, we give the work factors of the new algorithms, based on the work factors given for the Brouwer-Zimmermann's algorithm (see [39] or Section 2.7). Finally, we also show the results of some tests, which have been performed in MAGMA version V2.18-3, running on a server with an Intel Xeon processor (clock speed 2.40GHz) and 32GB of memory. For a better time comparison among the different methods, we have used the same internal MAGMA functions.

4.1 Minimum distance of binary nonlinear codes

Given a binary linear code K and a vector $v \in \mathbb{F}_2^n \backslash K$, we call $K_v = K \cup (K+v)$ an *extended coset.* Since K is linear, K_v is also linear and $K_v = \langle K, v \rangle$.

Proposition 4.1.1. *Let* $C = \bigcup_{i=0}^{t} \big(K_C + v_i\big)$ *with* $t \geq 2$. *The minimum weight of* C *can be computed as* $\min\{wt(K_{v_i}) \ : \ i \in \{1, \dots, t\}\}$.

Proof. The linear codes K_{v_i} include exactly all codewords of K_C and $K_C + v_i$ for all $i \in \{1, \dots, t\}$. Therefore, $wt(C)$ can be computed by finding the smallest value among all minimum weights of the linear codes K_{v_i}. $\square$

Proposition 4.1.2. *Let* $C = \bigcup_{i=0}^{t} \big(K_C + v_i\big)$ *with* $t \geq 2$. *The minimum distance of* C *can be computed as* $\min\{wt(K_{v_i+v_j}) \ : \ i \in \{0, 1, \dots, t-1\}, \ j \in \{i+1, \dots, t\}\}$, *where* $v_0 = \mathbf{0}$.

Proof. In order to compute the minimum distance between any pair of codewords, c_1 and c_2, we consider different cases. If c_1, $c_2 \in K_C$, then $d_H(c_1, c_2) = wt_H(c_1 + c_2) = wt_H(k)$, where $k \in K_C$. If $c_1 \in K_C$ and $c_2 \in K_C + v_i$, then $d_H(c_1, c_2) = wt_H(c_1 + c_2) = wt_H(k + v_i)$, where $k \in K_C$. If c_1, $c_2 \in K_C + v_i$, then $d_H(c_1, c_2) = wt_H(k)$, where $k \in K_C$. Finally, if $c_1 \in K_C + v_i$ and $c_2 \in K_C + v_j$ with $i \neq j$, then $d_H(c_1, c_2) = wt_H(k + v_i + v_j)$, where $k \in K_C$. Therefore, the statement follows. $\square$

Using Propositions 4.1.1, 4.1.2 and applying the Brouwer-Zimmermann's algorithm to compute the minimum weight of a linear code, we can define Algorithms 5 (MinW) and 6 (MinD) to compute the minimum weight and minimum distance of a binary nonlinear code, respectively. Note that the complexity of these two algorithms depends strongly on the number of coset representatives t and the complexity of the Brouwer-Zimmermann's algorithm. For the minimum weight, we compute t times the minimum weight of a linear code K_v, and for the minimum distance, $\binom{t+1}{2}$ times. In order to study the efficiency of these algorithms, we compare them with the brute force method.

Example 4.1.3. *Let* K *be the binary linear code of length* $n = 31$, *dimension* 5, *and minimum distance* 16, *constructed as the dual of the binary Hamming code of length* $n = 31$. *Let* $C_{31} = \bigcup_{i=0}^{3} \big(K_{C_{31}} + v_i\big)$, *where the kernel* $K_{C_{31}} = K$, $v_0 = \mathbf{0}$, *and the coset representatives are*

$$v_1 = (0010001110011010011110001011110),$$
$$v_2 = (0101101010111100101110100111101),$$
$$v_3 = (0000011100011101101000111101011).$$

algorithm 5 Minimum weight computation (MinW)

```
Data: A binary code $C$ given by the kernel $K_C$ and coset representatives
      $L = \{v_1, \dots, v_t\}$.
Result: The minimum weight $wt(C)$.
begin
    $wt(C) \leftarrow \text{Length}(C)$
    for $i \in [1, \dots, t]$ do
        $K_{v_i} \leftarrow \langle K_C, v_i \rangle$
        $wt(C) \leftarrow \min(wt(K_{v_i}), wt(C))$
    return $wt(C)$
```

algorithm 6 Minimum distance computation (MinD)

```
Data: A binary code $C$ given by the kernel $K_C$ and coset representatives
      $L = \{v_1, \dots, v_t\}$.
Result: The minimum distance $d(C)$.
begin
    $d(C) \leftarrow \text{Length}(C)$
    for  $i \in [0, \dots, t-1]$ do
        for $j \in [i+1, \dots, t]$ do
            $K_{v_j - v_i} \leftarrow \langle K_C, v_j - v_i \rangle$
            $d(C) \leftarrow \min(wt(K_{v_j + v_i}), d(C))$
    return $d(C)$
```

It is easy to check that the minimum weight of C_{31} is $wt(C_{31}) = 10$ and its minimum distance $d(C_{31}) = 8$. The time of computing $wt(C_{31})$ using brute force and Algorithm 5 (MinW) are 1.8×10^{-4} and 6×10^{-4} seconds, respectively. Note that sometimes a brute force calculation can be a faster way to obtain the minimum weight. On the other hand, the time of computing $d(C_{31})$ using brute force and Algorithm 6 (MinD) are 8.4×10^{-3} and 1.26×10^{-3} seconds, respectively, so it is much faster to use Algorithm 6 (MinD) than brute force.

Propositions 4.1.4 and 4.1.5 give us the work factors of computing the minimum weight and minimum distance by using the new Algorithms 5 (MinW) and 6 (MinD), respectively. Recall that a binary nonlinear code of length n with a kernel of dimension κ and t coset representatives has $M = 2^\kappa(t+1)$ codewords. Therefore, as we have already mentioned in Section 2.7, it is easy to see that the work factor for computing $wt(C)$ and $d(C)$ using brute force is, respectively,

$$n2^\kappa(t+1) \qquad \text{and} \qquad n\binom{2^\kappa(t+1)}{2}. \tag{4.1}$$

Proposition 4.1.4. *Let C be a binary nonlinear code of length n with kernel of dimension κ and coset representatives $L = \{v_1, \dots, v_t\}$. The work factor for computing $wt(C)$ using Algorithm 5 (MinW) is*

$$\sum_{i=1}^{t} \Big((n-\kappa-1)\lceil n/(\kappa+1) \rceil \sum_{r=1}^{\bar{r}_i} \binom{\kappa+1}{r} \Big) \tag{4.2}$$

where $\bar{r}_i$ is the smallest integer such that

$$\lfloor n/(\kappa+1) \rfloor (\bar{r}_i + 1) + max(0, \bar{r}_i + 1 - (\kappa + 1 - n \bmod (\kappa+1))) \geq wt(K_{v_i}).$$

Proof. By Proposition 4.1.1, the work factor of computing $wt(C)$ is equal to the sum of the work factors of computing $wt(K_{v_i})$ for all $i \in \{1, \dots, t\}$. Then, the result follows by Proposition 2.7.5, since the dimension of any linear code K_{v_i} is $\kappa + 1$. □

Proposition 4.1.5. *Let C be a binary nonlinear code of length n with kernel of dimension κ and coset representatives $L = \{v_1, \dots, v_t\}$. The work factor for computing $d(C)$ using Algorithm 6 (MinD) is*

$$\sum_{i=0}^{t-1} \Big(\sum_{j=i+1}^{t} \Big((n-\kappa-1)\lceil n/(\kappa+1) \rceil \sum_{r=1}^{\bar{r}_{i,j}} \binom{\kappa+1}{r} \Big) \Big) \tag{4.3}$$

where $\bar{r}_{i,j}$ is the smallest integer such that

$$\lfloor n/(\kappa+1)\rfloor(\bar{r}_{i,j}+1)+max(0,\bar{r}_{i,j}+1-(\kappa+1-n \bmod (\kappa+1))) \geq wt(K_{v_j-v_i}).$$

Proof. By Proposition 4.1.2, to compute $d(C)$ we need to compute the minimum weight of the binary linear codes $K_{v_i+v_j}$ for all $i,j \in \{0,1,\ldots,t\}$. The work factor of computing $d(C)$ is equal to the sum of the work factors of computing $wt(K_{v_i+v_j})$ for all $i,j \in \{0,1,\ldots,t\}$. By Proposition 2.7.5, since the dimension of any $K_{v_i+v_j}$ is $\kappa+1$ the result follows. □

Example 4.1.6. *Let us consider random $(100, 2^{15}\cdot 31)$ binary codes C, with kernels of dimension $\kappa \in \{8,\ldots,15\}$, and random $(100, 2^{7}\cdot 31)$ binary codes C, with kernels of dimension $\kappa \in \{3,\ldots,7\}$. Figure 4.1 shows the work factors given by Proposition 4.1.4 and (4.1), and the real time cost, for computing $wt(C)$ using Algorithm 5 (MinW) and brute force, respectively. Equivalently, Figure 4.2 shows the work factors given by Proposition 4.1.5 and (4.1), and the real time cost, for computing $d(C)$ using Algorithm 6 (MinD) and brute force, respectively.*

It can be seen from these figures that the work factors and real time cost follow the same trend. Moreover, keeping the same length and number of codewords, the time cost of using Algorithms 5 (MinW) and 6 (MinD) decreases sharply while the dimension of the kernel increases (or equivalently, while the number of cosets decreases). Note that when κ is large, Algorithms 5 (MinW) and 6 (MinD) save a lot of time. More specifically, Algorithm 5 (MinW) when $\kappa = 15$ and Algorithm 6 (MinD) when $\kappa = 7$ use only $1/31$ and $1/21$ time compared with brute force, respectively.

From Propositions 4.1.4 and 4.1.5, it can be seen that the work factor for computing $wt(C)$ and $d(C)$ of a binary code C relies on the parameters $\bar{r}_i$ and $\bar{r}_{i,j}$, which depend on $wt(K_{v_i})$ and $wt(K_{v_i+v_j})$, respectively, and they may be different for any i,j. Therefore, it is impossible to estimate the work factor if only the values n, κ and t of the binary code C are given. However, we can consider an upper bound of the work factor, and from that be able to estimate easily the work factor for computing the minimum weight and minimum distance. Since for any extended coset K_v we have that $wt(K_v) \leq wt(K_C)$, we can obtain an upper bound for the previous given work factors by replacing $wt(K_v)$ with $wt(K_C)$. Then, we obtain the following propositions.

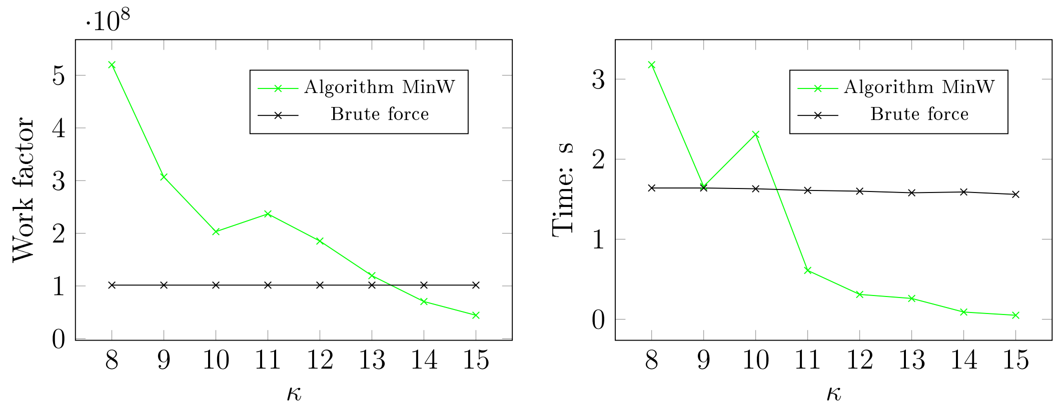


Figure 4.1: Work factor and time comparison on computing $wt(C)$ using Algorithm 5 (MinW) and brute force, for $(100, 2^{15} \cdot 31)$ binary codes, with kernels of dimension $\kappa \in \{8, \dots, 15\}$.

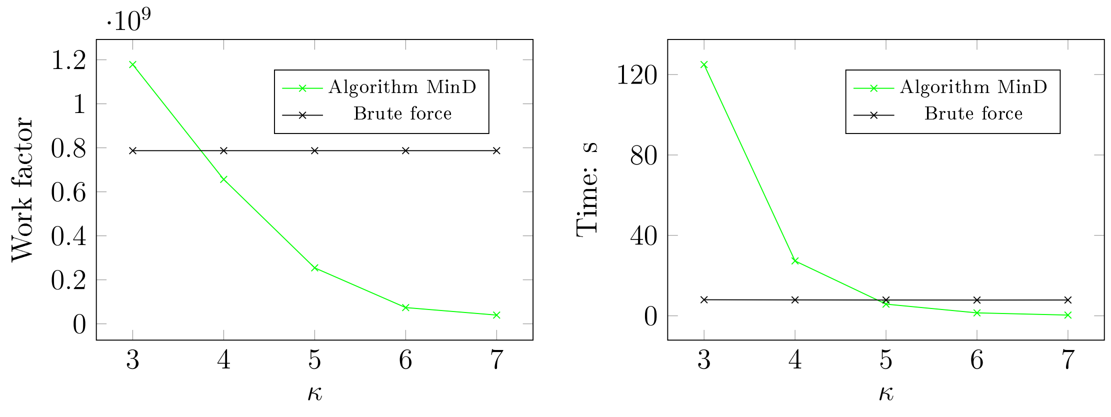


Figure 4.2: Work factor and time comparison on computing $d(C)$ using Algorithm 6 (MinD) and brute force, for $(100, 2^7 \cdot 31)$ binary codes, with kernels of dimension $\kappa \in \{3, \dots, 7\}$.

Proposition 4.1.7. *Let C be a binary nonlinear code of length n with kernel K_C of dimension κ and coset representatives $L = \{v_1, \ldots, v_t\}$. An upper bound for the work factor of computing $wt(C)$ using Algorithm 5 (MinW) is given by*

$$t(n-\kappa-1)\lceil n/(\kappa+1)\rceil \sum_{r=1}^{\bar{r}} \binom{\kappa+1}{r} \tag{4.4}$$

where $\bar{r}$ is the smallest integer such that

$$\lfloor n/(\kappa+1)\rfloor(\bar{r}+1) + max(0, \bar{r}+1-(\kappa+1-n \bmod (\kappa+1))) \geq wt(K_C).$$

Proof. Given n and κ, $f(r) = \lfloor n/(\kappa+1)\rfloor(r+1) + \max(0, r+1-(\kappa+1-n \bmod (\kappa+1)))$ is an increasing function. Let $\bar{r}_i$ be the smallest integer r such that $f(r) \geq wt(K_{v_i})$, that is, defined as in Proposition 4.1.4. Let $\bar{r}$ be the smallest integer r such that $f(r) \geq wt(K_C)$. Since $wt(K_{v_i}) \leq wt(K_C)$, we have that $\bar{r}_i \leq \bar{r}$. Therefore, the result follows by Proposition 4.1.4. □

Proposition 4.1.8. *Let C be a binary nonlinear code of length n with kernel K_C of dimension κ and coset representatives $L = \{v_1, \ldots, v_t\}$. An upper bound for the work factor of computing $d(C)$ using Algorithm 6 (MinD) is given by*

$$\binom{t+1}{2}(n-\kappa-1)\lceil n/(\kappa+1)\rceil \sum_{r=1}^{\bar{r}} \binom{\kappa+1}{r} \tag{4.5}$$

where $\bar{r}$ is the smallest integer such that

$$\lfloor n/(\kappa+1)\rfloor(\bar{r}+1) + max(0, \bar{r}+1-(\kappa+1-n \bmod (\kappa+1))) \geq wt(K_C).$$

Proof. Straightforward using Proposition 4.1.5 and the same arguments as in Proposition 4.1.7. □

Example 4.1.9. *Considering the same binary codes as in Example 4.1.6, Figures 4.3 and 4.4 show the differences between the work factors and their upper bounds for computing the minimum weight and minimum distance. Note that the upper bound of work factor is quite close to the work factor and is much easier to estimate, since we just need $wt(K_C)$ along with the values of n, κ and t of C.*

From Algorithms 5 (MinW) and 6 (MinD), it is easy to see that the weight of some codewords in the kernel K_C is computed several times, specifically, once for each $K_{v_i+v_j} = K_C \cup (K_C + v_i + v_j)$, where $i, j \in \{0, 1, \ldots, t\}$ and

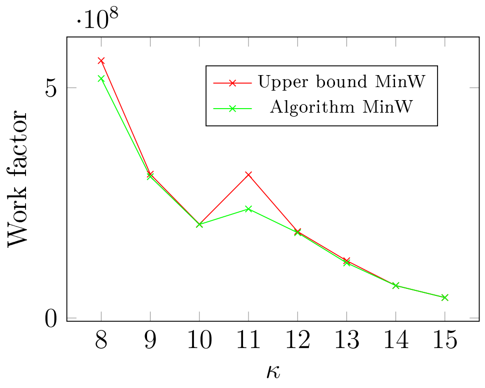


Figure 4.3: Work factor and work factor upper bound on computing $wt(C)$ for $(100, 2^{15} \cdot 31)$ binary codes.

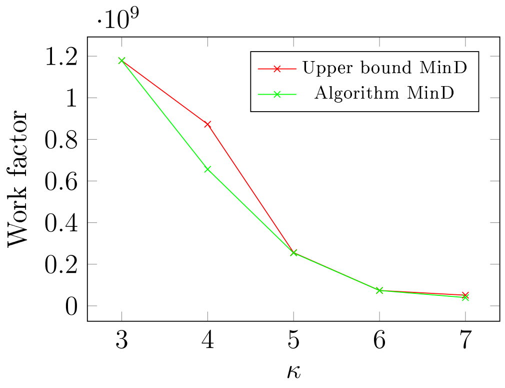


Figure 4.4: Work factor and work factor upper bound on computing $d(C)$ for $(100, 2^7 \cdot 31)$ binary codes.

$i < j$. However, we will show that we can make a little adjustment to these algorithms, in order to avoid this repetition.

In Brouwer-Zimmermann's algorithm, the enumerating process is divided into several steps. In the rth step, it enumerates all linear combinations of r rows of the generator matrix of $K_{v_i+v_j}$ of dimension $\kappa + 1$, examines the minimum weight of each combination and compares it with the lower bound. In order to avoid enumerate some codewords several times, we can modify the previous algorithms and enumerate only the codewords in each coset $K_C + v_i + v_j$. Then, in the rth step, we enumerate all linear combinations of r rows of the generator matrix of K_C of dimension κ and compute the weight of each combination adding the vector $v_i + v_j$. The codewords in the kernel are considered by adding the all-zero vector to the set of coset representatives. After this adjustment, the work factor using the improved Algorithms 7 and 8, which are referred as Algorithm IMinW and IMinD, respectively, is reduced as it is shown in the following propositions.

Proposition 4.1.10. *Let C be a binary nonlinear code of length n with kernel of dimension κ and coset representatives $L = \{v_1, \dots, v_t\}$. The work factor for computing $wt(C)$ using improved Algorithm 7 (IMinW) is*

$$\sum_{i=1}^{t+1} \left((n-\kappa) \lceil n/\kappa \rceil \sum_{r=1}^{\bar{r}_i} \binom{\kappa}{r} \right) \tag{4.6}$$

where $\bar{r}_i$ is the smallest integer such that

$$\lfloor n/\kappa \rfloor (\bar{r}_i + 1) + max(0, \bar{r}_i + 1 - (\kappa - n \bmod \kappa)) \geq wt(K_{v_i}).$$

algorithm 7 Improved minimum weight (IMinW).

```
Data: A binary code $C$ given by the kernel $K_C$ and coset representatives
      $L = \{v_1, \dots, v_t\}$.
Result: The minimum weight $wt(C)$.
begin
    Let $G_1, \dots, G_h$ be a sequence of systematic generator matrices of $K_C$ with
    rank deficits $\delta_1, \dots, \delta_h$, respectively.
    $L' \leftarrow L \cup \{\mathbf{0}\}$
    for $u \in L'$ do
        for $j \in [1, \dots, h]$ do
            Let $u_j \in K_C + u$ having 0 in the information set of $G_j$.
        for $r \in [1, \dots, \kappa]$ do
            $wt_{low} \leftarrow \sum_{j=1}^{h} \max\{0, r + 1 - \delta_j\}$
            for $j \in [1, \dots, h]$ do
                $wt_{up}^{u} \leftarrow \min\{wt_H(mG_j + u_j) \mid m \in \mathbb{F}_2^{\kappa},\ wt_H(m) = r\}$
                if $wt_{up}^{u} \leq wt_{low}$ then
                    break
    $wt(C) \leftarrow \min\{wt_{up}^{u} \mid u \in L'\}$
    return $wt(C)$
```

algorithm 8 Improved minimum distance (IMinD).

```
Data: A binary code $C$ given by the kernel $K_C$ and coset representatives
      $L = \{v_1, \ldots, v_t\}$.
Result: The minimum distance $d(C)$.
begin
    Let $G_1, \ldots, G_h$ be a sequence of systematic generator matrices of $K_C$ with
    rank deficits $\delta_1, \ldots, \delta_h$, respectively.
    $L' \leftarrow \{v_j - v_i \mid 0 \leq i < j \leq t\}$
    for $u \in L'$ do
        for $j \in [1, \ldots, h]$ do
            Let $u_j \in K_C + u$ having 0 in the information set of $G_j$.
        for $r \in [1, \ldots, \kappa]$ do
            $wt_{low} \leftarrow \sum_{j=1}^{h} \max\{0, r + 1 - \delta_j\}$
            for $j \in [1, \ldots, h]$ do
                $wt_{up}^{u} \leftarrow \min\{wt_H(mG_j + u_j) \mid m \in \mathbb{F}_2^{\kappa},\ wt_H(m) = r\}$
                if $wt_{up}^{u} \leq wt_{low}$ then
                    break
    $d(C) \leftarrow \min\{wt_{up}^{u} \mid u \in L'\}$
    return $d(C)$.
```

Proposition 4.1.11. *Let C be a binary nonlinear code of length n with kernel of dimension κ and coset representatives $L = \{v_1, \dots, v_t\}$. The work factor for computing $d(C)$ using improved Algorithm 8 (IMinD) is*

$$\sum_{i=0}^{t-1}\Big(\sum_{j=i+1}^{t}\Big((n-\kappa)\lceil n/\kappa\rceil\sum_{r=1}^{\bar{r}_{i,j}}\binom{\kappa}{r}\Big)\Big) + (n-\kappa)\lceil n/\kappa\rceil\sum_{r=1}^{\bar{r}_{0,0}}\binom{\kappa}{r} \tag{4.7}$$

where $\bar{r}_{i,j}$ is the smallest integer such that

$$\lfloor n/\kappa\rfloor(\bar{r}_{i,j}+1) + \max(0, \bar{r}_{i,j}+1-(\kappa - n \bmod \kappa)) \geq wt(K_{v_i+v_j}).$$

Note that $\bar{r}_{0,0} = \bar{r}$ given in Proposition 2.7.5. As before, we can also establish an upper bound for the work factors given by Propositions 4.1.10 and 4.1.11 by using the same arguments as in Propositions 4.1.7 and 4.1.8. Using these upper bounds, again it is possible to estimate the work factor for computing $wt(C)$ and $d(C)$ from the parameters n, κ, t and $wt(K_C)$ of a binary nonlinear code C.

Proposition 4.1.12. *Let C be a binary nonlinear code of length n with kernel K_C of dimension κ and coset representatives $L = \{v_1, \dots, v_t\}$. An upper bound for the work factor of computing $wt(C)$ using improved Algorithm 7 (IMinW) is given by*

$$(t+1)(n-\kappa)\lceil n/\kappa\rceil\sum_{r=1}^{\bar{r}}\binom{\kappa}{r} \tag{4.8}$$

where $\bar{r}$ is the smallest integer such that

$$\lfloor n/\kappa\rfloor(\bar{r}+1) + max(0, \bar{r}+1-(\kappa - n \bmod \kappa)) \geq wt(K_C).$$

Proposition 4.1.13. *Let C be a binary nonlinear code of length n with kernel K_C of dimension κ and coset representatives $L = \{v_1, \dots, v_t\}$. An upper bound for the work factor of computing $d(C)$ using improved Algorithm 8 (IMinD) is given by*

$$\Big(\binom{t+1}{2}+1\Big)(n-\kappa)\lceil n/\kappa\rceil\sum_{r=1}^{\bar{r}}\binom{\kappa}{r} \tag{4.9}$$

where $\bar{r}$ is the smallest integer such that

$$\lfloor n/\kappa\rfloor(\bar{r}+1) + max(0, \bar{r}+1-(\kappa - n \bmod \kappa)) \geq wt(K_C).$$

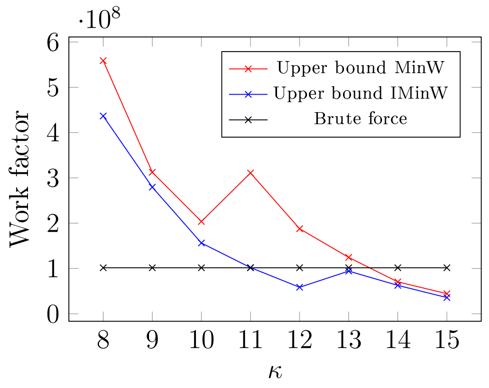


Figure 4.5: Work factor upper bounds on computing $wt(C)$ for $(100, 2^{15} \cdot 31)$ binary codes.

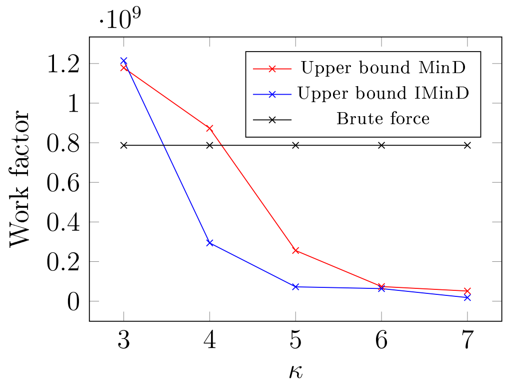


Figure 4.6: Work factor upper bounds on computing $d(C)$ for $(100, 2^7 \cdot 31)$ binary codes.

Example 4.1.14. *Figures 4.5 and 4.6 show these upper bounds for the work factors for both algorithms presented in the dissertation and brute force, using* $(100, 2^{15}\cdot 31)$ *and* $(100, 2^7\cdot 31)$ *binary codes, respectively. Through these examples, we can see the improvement on Algorithms 7 (IMinW) and 8 (IMinD).*

Note that the results on these upper bounds for the work factors allow to establish from which parameters of the given code, it is better to use the new presented algorithms instead of the brute force method.

4.2 Minimum distance of q-ary nonlinear codes

The results for binary nonlinear code can be generalized into q-ary nonlinear codes with a few adjustments.

For binary nonlinear codes $C = \bigcup_{i=0}^{t}(K_C + v_i)$, since K_C is linear, the subcodes $K_C \cup (K_C + v_i)$ are linear for all $i \in \{1, \ldots, t\}$. However, for q-ary nonlinear codes with $q > 2$, $K_C \cup (K_C + v_i)$ is not a linear code, so we need another strategy to apply the Brouwer-Zimmermann's algorithm.

Given a q-ary linear code K of dimension κ and a vector $v \in \mathbb{F}_q^n \backslash K$, the linear span $K_v = \langle K, v\rangle = \bigcup_{\lambda\in\mathbb{F}_q}(K + \lambda v)$ of dimension $\kappa + 1$ is called an *extended coset.*

Proposition 4.2.1. *Let K be a q-ary linear code and $v \in \mathbb{F}_q^n\backslash K$. The minimum weight of the extended coset K_v is equal to the minimum weight of $K \cup (K + v)$.*

Proof. Let e be a codeword of minimum weight in K_v. Then, $e = k + \lambda v$, where $k \in K$ and $\lambda \in \mathbb{F}_q$. Let $\lambda' = 1/\lambda$ if $\lambda \in \mathbb{F}_q \backslash \{\mathbf{0}\}$. Note that $wt_H(\lambda' e) = wt_H(e)$ for all $\lambda' \in \mathbb{F}_q \backslash \{\mathbf{0}\}$.

If $wt(K \cup (K + v)) = wt(K)$, then $wt(K + v) \geq wt(K)$. Since $\lambda' e = \lambda' k + v \in K + v$, $wt_H(\lambda' e) \geq wt(K)$, which is the same as $wt(K_v) \geq wt(K)$. Since $K \subseteq K_v$, $wt(K_v) \leq wt(K)$, so $wt(K_v) = wt(K) = wt(K \cup (K + v))$.

If $wt(K \cup (K + v)) < wt(K)$, $e' \in K + v$, where e' is a codeword of minimum weight in $K \cup (K + v)$. Since $K \cup (K + v) \subseteq K_v$, $wt(K_v) \leq wt(K \cup (K + v))$. Then, since $\lambda' e = \lambda' k + v \in K + v$, $wt_H(\lambda' e) \leq wt_H(e')$. Finally, $wt_H(e') = wt(K + v)$, so $wt_H(\lambda' e) = wt_H(e')$, and the statement follows. □

Proposition 4.2.2. *Let $C = \bigcup_{i=0}^{t} \left(K + v_i\right)$ be a q-ary nonlinear code with kernel K and coset representatives $L = \{v_1, \ldots, v_t\}$. The minimum weight of C can be computed as* $\min\{wt(K_{v_i}) \ : \ i \in \{1, \ldots, t\}\}$.

Proof. According to Proposition 4.2.1, $wt(K_{v_i}) = wt(K \cup (K + v_i))$ for all $i \in \{1, \ldots, t\}$. Moreover, note that the linear code K_{v_i} includes all codewords of $K \cup (K + v_i)$. Therefore, $wt(C)$ can be computed by finding the smallest value among all minimum weights of the linear codes K_{v_i}. □

Proposition 4.2.3. *Let $C = \bigcup_{i=0}^{t} \left(K + v_i\right)$ be a q-ary nonlinear code with kernel K and coset representatives $L = \{v_1, \ldots, v_t\}$. The minimum distance of C can be computed as* $\min\{wt(K_{v_j - v_i}) \ : \ i \in \{0, 1, \ldots, t-1\},\ j \in \{i+1, \ldots, t\}\}$*, where* $v_0 = \mathbf{0}$.

Proof. In order to compute the minimum distance between any pair of codewords, c_1 and c_2, we consider different cases. If $c_1, c_2 \in K$, then $d_H(c_1, c_2) = wt_H(c_2 - c_1) = wt_H(k)$, where $k \in K$. If $c_1 \in K$ and $c_2 \in K + v_i$, then $d_H(c_1, c_2) = wt_H(c_2 - c_1) = wt_H(k + v_i)$, where $k \in K$. If $c_1, c_2 \in K + v_i$, then $d_H(c_1, c_2) = wt_H(k)$, where $k \in K$. Finally, if $c_1 \in K + v_i$ and $c_2 \in K + v_j$ with $i \neq j$, then $d_H(c_1, c_2) = wt_H(k + v_j - v_i)$, where $k \in K$. Therefore, the statement follows. □

Recall that a q-ary nonlinear code of length n with a kernel of dimension κ and t coset representatives has $M = q^{\kappa}(t + 1)$ codewords. Therefore, it is easy to see that the work factor for computing $wt(C)$ and $d(C)$ using a brute force algorithm is, respectively,

$$\log_2(q) n q^{\kappa}(t + 1) \qquad \text{and} \qquad \log_2(q) n \binom{q^{\kappa}(t + 1)}{2}. \tag{4.10}$$

Using the representation of q-ary nonlinear codes given in Section 3.1.2, Propositions 4.2.2, 4.2.3 and the known Brouwer-Zimmermann's algorithm to compute the minimum weight of a q-ary linear code, we can see that Algorithms 5 (MinW) and 6 (MinD) can be generalized into q-ary nonlinear codes without any change.

Propositions 4.2.4 and 4.2.5 give us the work factors of computing the minimum weight and minimum distance by using Algorithms 5 (MinW) and 6 (MinD) applied to q-ary nonlinear codes, respectively.

Proposition 4.2.4. *Let C be a q-ary nonlinear code of length n with kernel of dimension κ and coset representatives $L = \{v_1, \dots, v_t\}$. The work factor for computing $wt(C)$ using Algorithm 5 (MinW) is*

$$\sum_{i=1}^{t} \Big(\log_2(q)(n-\kappa-1)\lceil n/(\kappa+1)\rceil \sum_{r=1}^{\bar{r}_i} \binom{\kappa+1}{r}(q-1)^{r-1}\Big) \qquad (4.11)$$

where $\bar{r}_i$ is the smallest integer such that

$$\lfloor n/(\kappa+1)\rfloor(\bar{r}_i+1) + \max(0, \bar{r}_i + 1 - (\kappa+1-n \bmod (\kappa+1))) \geq wt(K_{v_i}).$$

Proof. By Proposition 4.2.2, the work factor of computing $wt(C)$ is equal to the sum of the work factors of computing $wt(K_{v_i})$ for all $i \in \{1, \dots, t\}$. Then, the result follows by Proposition 2.7.5, since the dimension of any q-ary linear code K_{v_i} is $\kappa + 1$. □

Proposition 4.2.5. *Let C be a q-ary nonlinear code of length n with kernel of dimension κ and coset representatives $L = \{v_1, \dots, v_t\}$. The work factor for computing $d(C)$ using Algorithm 6 (MinD) is*

$$\sum_{i=0}^{t-1} \Big(\sum_{j=i+1}^{t} \Big(\log_2(q)(n-\kappa-1)\lceil n/(\kappa+1)\rceil \sum_{r=1}^{\bar{r}_{i,j}} \binom{\kappa+1}{r}(q-1)^{r-1}\Big)\Big) \quad (4.12)$$

where $\bar{r}_{i,j}$ is the smallest integer such that

$$\lfloor n/(\kappa+1)\rfloor(\bar{r}_{i,j}+1) + \max(0, \bar{r}_{i,j} + 1 - (\kappa+1-n \bmod (\kappa+1))) \geq wt(K_{v_j-v_i}).$$

Proof. By Proposition 4.2.3, to compute $d(C)$ we need to compute the minimum weight of the q-ary linear codes $K_{v_j-v_i}$ for all $i, j \in \{0, 1, \dots, t\}$. The work factor of computing $d(C)$ is equal to the sum of the work factors of computing $wt(K_{v_j-v_i})$ for all $i, j \in \{0, 1, \dots, t\}$. By Proposition 2.7.5, since the dimension of any $K_{v_j-v_i}$ is $\kappa + 1$ the result follows. □

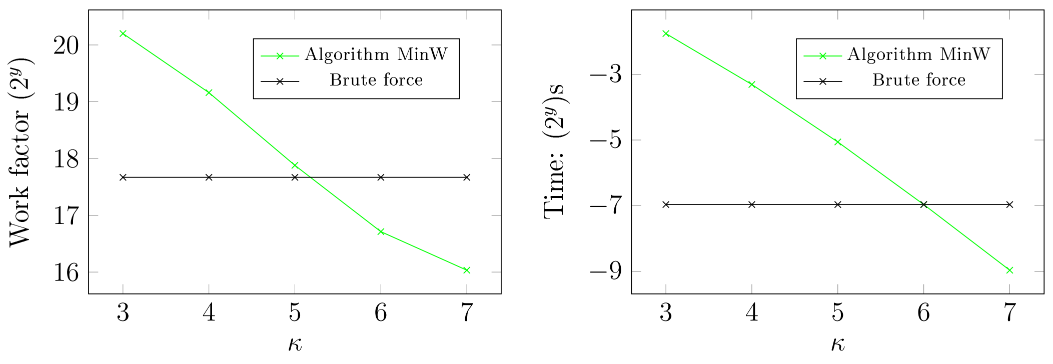


Figure 4.7: Work factor and time comparison on computing $wt(C)$ using Algorithm 5 (MinW) and brute force, for $(30, 3^7 \cdot 4)$ ternary codes, with kernels of dimension $\kappa \in \{3, \dots, 7\}$.

Example 4.2.6. *Let us consider random $(30, 3^7 \cdot 4)$ ternary codes C, with kernels of dimension $\kappa \in \{3, \dots, 7\}$. Figure 4.7 shows the work factors given by Proposition 4.2.4 and (4.10), and the real time cost, for computing $wt(C)$ using Algorithm 5 (MinW) and brute force, respectively. Equivalently, Figure 4.8 shows the work factors given by Proposition 4.2.5 and (4.10), and the real time cost, for computing $d(C)$ using Algorithm 6 (MinD) and brute force, respectively. In all figures, the work factors are expressed in logarithmic scale.*

It can be seen from these figures that the work factors and real time cost follow the same trend. Moreover, keeping the same length and number of codewords, the time cost of using Algorithms 5 (MinW) and 6 (MinD) decreases sharply while the dimension of the kernel increases (or equivalently, while the number of cosets decreases). Note that when κ is large, Algorithms 5 (MinW) and 6 (MinD) save a lot of time.

Similarly as for binary codes, from Propositions 4.2.4 and 4.2.4, it can be seen that the work factor for computing $wt(C)$ and $d(C)$ of a q-ary code C relies on the parameters $\bar{r}_i$ and $\bar{r}_{i,j}$, which depend on $wt(K_{v_i})$ and $wt(K_{v_j - v_i})$, respectively, and they may be different for any i, j. Since for any extended coset K_v we have that $wt(K_v) \leq wt(K_C)$, we can obtain an upper bound for the previous given work factors by replacing $wt(K_v)$ with $wt(K_C)$. That is, we have also the following propositions.

Proposition 4.2.7. *Let C be a q-ary nonlinear code of length n with kernel K_C of dimension κ and t coset representatives. An upper bound for the work*

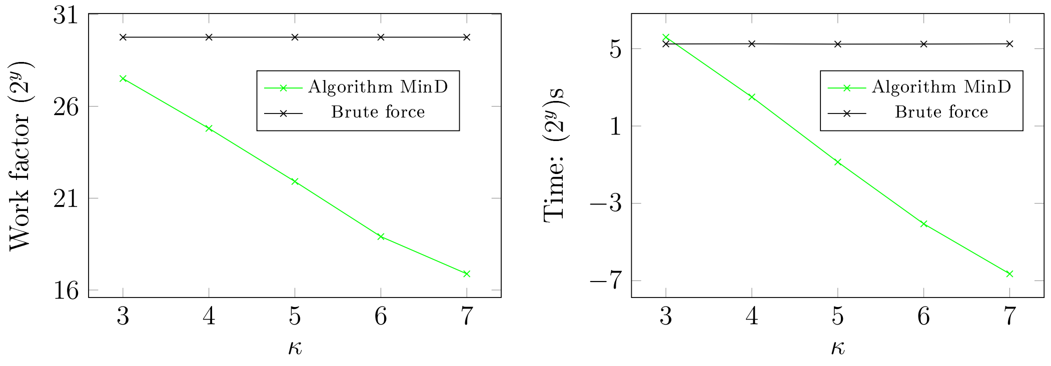


Figure 4.8: Work factor and time comparison on computing $d(C)$ using Algorithm 6 (MinD) and brute force, for $(30, 3^7 \cdot 4)$ ternary codes, with kernels of dimension $\kappa \in \{3, \dots, 7\}$.

factor of computing $wt(C)$ using Algorithm 5 (MinW) is given by

$$t \cdot \log_2(q)(n-\kappa-1)\lceil n/(\kappa+1)\rceil \sum_{r=1}^{\bar{r}} \binom{\kappa+1}{r}(q-1)^{r-1} \tag{4.13}$$

where $\bar{r}$ is the smallest integer such that

$$\lfloor n/(\kappa+1)\rfloor(\bar{r}+1) + \max(0, \bar{r}+1-(\kappa+1-n \bmod (\kappa+1))) \geq wt(K_C).$$

Proof. Given n and κ, $f(r) = \lfloor n/(\kappa+1)\rfloor(r+1) + \max(0, r+1-(\kappa+1-n \bmod (\kappa+1)))$ is an increasing function. Let $\bar{r}_i$ be the smallest integer r such that $f(r) \geq wt(K_{v_i})$, that is, defined as in Proposition 4.2.4. Let $\bar{r}$ be the smallest integer r such that $f(r) \geq wt(K_C)$. Since $wt(K_{v_i}) \leq wt(K_C)$, we have that $\bar{r}_i \leq \bar{r}$. Therefore, the result follows by Proposition 4.2.4. □

Proposition 4.2.8. *Let C be a q-ary nonlinear code of length n with kernel K_C of dimension κ and t coset representatives. An upper bound for the work factor of computing $d(C)$ using Algorithm 6 (MinD) is given by*

$$\binom{t+1}{2} \cdot \log_2(q)(n-\kappa-1)\lceil n/(\kappa+1)\rceil \sum_{r=1}^{\bar{r}} \binom{\kappa+1}{r}(q-1)^{r-1} \tag{4.14}$$

where $\bar{r}$ is the smallest integer such that

$$\lfloor n/(\kappa+1)\rfloor(\bar{r}+1) + \max(0, \bar{r}+1-(\kappa+1-n \bmod (\kappa+1))) \geq wt(K_C).$$

Proof. Straightforward using Proposition 4.2.5 and the same arguments as in Proposition 4.2.7. □

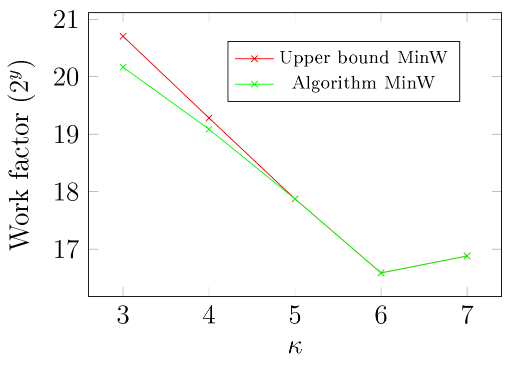


Figure 4.9: Work factor and work factor upper bound on computing $wt(C)$ for $(30, 3^7 \cdot 4)$ ternary codes.

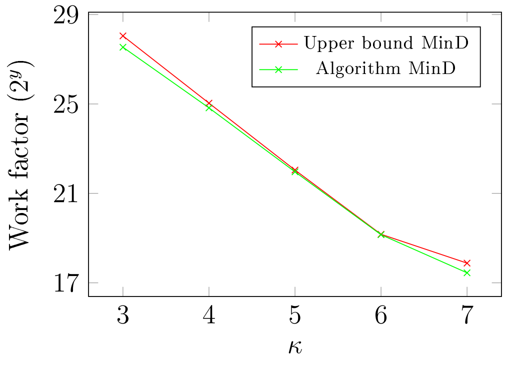


Figure 4.10: Work factor and work factor upper bound on computing $d(C)$ for $(30, 3^7 \cdot 4)$ ternary codes.

Example 4.2.9. *Considering the same random ternary codes C as in Example 4.2.6, Figures 4.9 and 4.10 show the differences between the work factors and their upper bounds for computing the minimum weight and minimum distance. Note that the upper bound of work factor is quite close to the work factor and is much easier to estimate, since we just need $wt(K_C)$ along with the values of n, κ and t of C. In both figures, the work factors are expressed in logarithmic scale.*

Like for binary codes, from Algorithms 5 (MinW) and 6 (MinD) applied to q-ary nonlinear codes, it can be seen that the weight of some codewords in the kernel K_C is computed several times, specifically, once for each $K_{v_j - v_i}$, where $i, j \in \{0, 1, \dots, t\}$ and $i < j$. Moreover, we need to compute the weight of extra vectors which belong to $K_C + \lambda(v_j - v_i)$, where $\lambda \in \mathbb{F}_q \backslash \{0, 1\}$. The same improvements as we considered for binary codes in Algorithms 7 (IMinW) and 8 (IMinD), can be considered for computing the minimum weight and minimum distance of q-ary codes. Therefore, again, we can see that Algorithms 7 (IMinW) and 8 (IMinD) can be generalized into q-ary nonlinear codes without any change.

After this adjustment, the work factor using the improved Algorithms 7 and 8 applied to q-ary nonlinear codes, which are also referred as Algorithms IMinW and IMinD, respectively, can be reduced as it is shown in the following propositions.

Proposition 4.2.10. *Let C be a q-ary nonlinear code of length n with kernel of dimension κ and coset representatives $L = \{v_1, \dots, v_t\}$. The work factor* *for computing $wt(C)$ using improved Algorithm 7 (IMinW) is*

$$\sum_{i=1}^{t+1}\Big(\log_2(q)(n-\kappa)\lceil n/\kappa\rceil\sum_{r=1}^{\bar{r}_i}\binom{\kappa}{r}(q-1)^{r-1}\Big) \tag{4.15}$$

where $\bar{r}_i$ is the smallest integer such that

$$\lfloor n/\kappa\rfloor(\bar{r}_i+1)+\max(0,\bar{r}_i+1-(\kappa-n \bmod \kappa))\geq wt(K_{v_i}).$$

Proposition 4.2.11. *Let C be a q-ary nonlinear code of length n with kernel of dimension κ and coset representatives $L=\{v_1,\dots,v_t\}$. The work factor for computing $d(C)$ using improved Algorithm 8 (IMinD) is*

$$\begin{aligned}\sum_{i=0}^{t-1}\Big(\sum_{j=i+1}^{t}\Big(\log_2(q)(n-\kappa)\lceil n/\kappa\rceil\sum_{r=1}^{\bar{r}_{i,j}}\binom{\kappa}{r}(q-1)^{r-1}\Big)\Big)\\ +\log_2(q)(n-\kappa)\lceil n/\kappa\rceil\sum_{r=1}^{\bar{r}_{0,0}}\binom{\kappa}{r}(q-1)^{r-1}\end{aligned} \tag{4.16}$$

where $\bar{r}_{i,j}$ is the smallest integer such that

$$\lfloor n/\kappa\rfloor(\bar{r}_{i,j}+1)+\max(0,\bar{r}_{i,j}+1-(\kappa-n \bmod \kappa))\geq wt(K_{v_j-v_i}).$$

Note that $\bar{r}_{0,0}=\bar{r}$ given in Proposition 2.7.5. As before, we can also establish an upper bound for the work factors given by Propositions 4.2.10 and 4.2.11 by using the same arguments as in Propositions 4.2.7 and 4.2.8. Using these upper bounds, again it is possible to estimate the work factor for computing $wt(C)$ and $d(C)$ from the parameters n, κ, t and $wt(K_C)$ of a q-ary nonlinear code C.

Proposition 4.2.12. *Let C be a q-ary nonlinear code of length n with kernel K_C of dimension κ and t coset representatives. An upper bound for the work factor of computing $wt(C)$ using improved Algorithm 7 (IMinW) is given by*

$$(t+1)\cdot\log_2(q)(n-\kappa)\lceil n/\kappa\rceil\sum_{r=1}^{\bar{r}}\binom{\kappa}{r}(q-1)^{r-1} \tag{4.17}$$

where $\bar{r}$ is the smallest integer such that

$$\lfloor n/\kappa\rfloor(\bar{r}+1)+\max(0,\bar{r}+1-(\kappa-n \bmod \kappa))\geq wt(K_C).$$

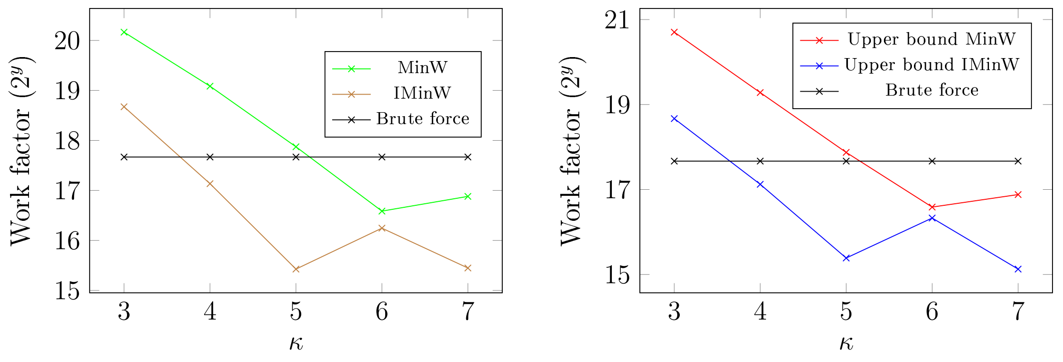


Figure 4.11: Work factors and upper bounds on computing $wt(C)$ for $(30, 3^7 \cdot 4)$ ternary codes.

Proposition 4.2.13. *Let C be a q-ary nonlinear code of length n with kernel K_C of dimension κ and t coset representatives. An upper bound for the work factor of computing $d(C)$ using improved Algorithm 8 (IMinD) is given by*

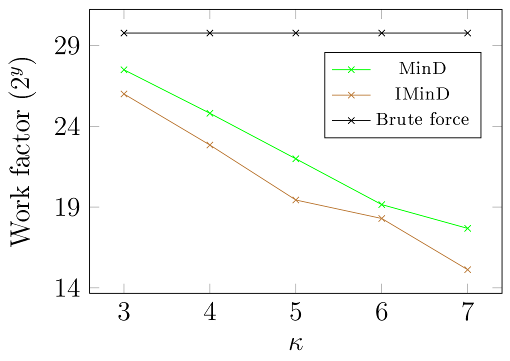
$$\left(\binom{t+1}{2}+1\right) \cdot \log_2(q)(n-\kappa)\lceil n/\kappa \rceil \sum_{r=1}^{\bar{r}} \binom{\kappa}{r}(q-1)^{r-1} \tag{4.18}$$

where $\bar{r}$ is the smallest integer such that

$$\lfloor n/\kappa \rfloor(\bar{r}+1) + \max(0, \bar{r}+1-(\kappa - n \bmod \kappa)) \geq wt(K_C).$$

Example 4.2.14. *Figures 4.11 and 4.12 show the work factors (and the work factors upper bounds) for both algorithms presented in the dissertation and brute force, using $(30, 3^7 \cdot 4)$ ternary codes. Through these examples, we can see the improvement on Algorithms 7 (IMinW) and 8 (IMinD). In all figures, the work factors are expressed in logarithmic scale.*

Note that the results on these upper bounds for the work factors allow to establish from which parameters of the given code, it is better to use the new presented algorithms instead of the brute force method.



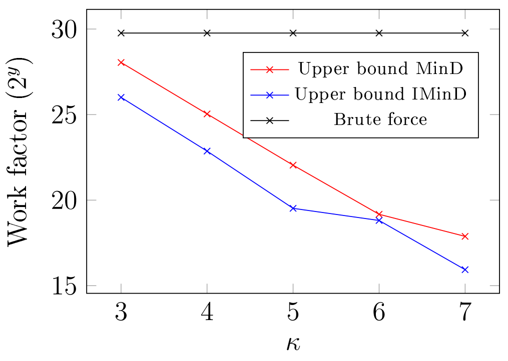


Figure 4.12: Work factors and upper bounds on computing $d(C)$ for $(30, 3^7 \cdot 4)$ ternary codes.

Chapter 5

Application to decoding

In this chapter, we show how to apply the previous algorithms to decode linear and nonlinear codes. Specifically, we first focus on the binary case, that is, we show how to decode binary linear and nonlinear codes using the algorithms described in Chapter 4 to compute the minimum weight of a linear code or a coset of a linear code. Then, we generalize the description of these new decoding algorithms from binary codes to q-ary codes, distinguishing between q-ary linear codes and q-ary nonlinear codes.

For linear codes, the most known general decoding method is the syndrome decoding, where a syndrome table is computed before starting the decoding process. For linear codes, we compare the new methods of decoding using the minimum weight algorithms with the syndrome decoding, although it is a text book algorithm and rarely used in application; and for nonlinear codes in general, as far as we know they are the first general decoding methods.

5.1 Decoding binary linear codes

Let K be an $[n, \kappa, d]$ binary linear code. The general decoding algorithm for linear codes is the syndrome decoding (see Section 2.4). We recall that using this method, before the decoding process starts, it is necessary to compute a syndrome table pairing each syndrome $s \in \mathbb{F}_2^{n-\kappa}$ with an error vector e of minimum weight in the coset associated to that syndrome. Although creating the syndrome table is a one-time task, which is carried out before decoding the received vectors, sometimes it can be difficult to create and store it. Moreover, if it contains many elements, it can also be difficult to find the corresponding error vector from a given syndrome. In these cases, other methods can be more efficient.

Extended coset decoding for binary linear codes

In general, as we have seen in Section 2.4, to decode a received vector $u = c + e$, where $c \in K$ and e is an error vector, we need to find a codeword $c' \in K$ such that

$$d_H(c, u) = d_H(c', u) = \min\{d_H(\bar{c}, u) \mid \bar{c} \in K\},$$

and decode u as c'. Since $d_H(\bar{c}, u) = wt_H(u - \bar{c})$, this is the same as finding a vector $e' = u - c'$ of minimum weight $wt_H(e') = wt_H(e)$ in the coset $u - K$. Moreover, since K is linear, $-K = K$, so the decoding process is equivalent to find a vector of minimum weight in $K + u$. Note that, in general, it is difficult to find a vector of minimum weight in a coset $K + u$ of a linear code K.

In order to find a vector of minimum weight in a coset $K + u$ of a binary linear code K, we can consider the linear extended coset $K_u = K \cup (K + u)$ and apply the Brouwer-Zimmermann's algorithm for binary linear codes. If the number of errors is less than the minimum weight d of K, which means that $d_H(c, u) = wt_H(e) < d$, then an error vector e' such that $wt_H(e') = wt_H(e)$ can be found as a codeword of minimum weight, which is called *minimum word*, in the linear extended coset K_u.

Proposition 5.1.1. *Let K be a binary linear code with minimum weight d. For a received vector $u = c + e \notin K$, where $c \in K$, let $K_u = K \cup (K + u)$. If $wt_H(e) < d$, then u can be decoded as $c' = u - e' \in K$, where e' is a vector of minimum weight in K_u, so $wt_H(e) = wt_H(e')$. Note that if $wt_H(e) \leq \lfloor \frac{d-1}{2} \rfloor$, then $e' = e$ and $c' = c$.*

Proof. For a received vector $u = c + e \in \mathbb{F}_2^n \backslash K$, where $c \in K$, in order to decode it, we look for a vector e' of minimum weight such that $c' = u - e' \in K$. This is equivalent to find a vector e' of minimum weight in the coset containing u, which is $K + u$.

Considering the linear extended coset $K_u = K \cup (K + u)$, if the minimum weight of the coset $K + u$ is less than d, we can find a vector e' of minimum weight in the coset $K + u$ such that $wt_H(e') = wt_H(e) < d$ by finding a vector of minimum weight in the linear extended coset K_u. Then, the received vector u can be decoded as $c' = u - e' \in K$. $\square$

In this way, we can decode a received vector as long as less than d errors have been added to the transmitted codeword. When d or more than d errors occurs during the transmission, a vector of minimum weight in K_u could come from K, and then an error vector e' can not be found by Proposition 5.1.1. Therefore, this method, called *extended coset decoding*, provides a complete decoding but only up to $d-1$ errors. Note that if $\rho(K) \leq d-1$, that is when K is a maximal code, we actually obtain a complete decoding.

The extended coset decoding consists on finding a minimum word of the linear extended coset K_u, which is similar to computing the minimum weight of K_u. Thus, we can apply the Brouwer-Zimmermann's algorithm. However, in order to obtain a minimum word using this algorithm, the columns of the generator matrices corresponding to the information sets can not be removed, as it happens to simplify the computation of the weight of a codeword while the minimum weight is computed. In addition, to get the original minimum word, it maybe necessary to reverse the column permutation done to the generator matrices, but this is negligible comparing to the enumeration process. Finally, note that before finding a minimum word, the minimum weight of the linear extended coset K_u is unknown. Therefore, in order to estimate the work factor, we can use the minimum weight $d = wt(K)$ instead of $wt(K_u)$, and obtain an upper bound of the work factor, since $d \geq wt(K_u)$.

Therefore, in conclusion, using the extended coset decoding, a work factor upper bound of decoding a received vector u encoded using an $[n, \kappa, d]$ binary linear code K is

$$n\lceil n/(\kappa+1)\rceil \sum_{r=1}^{\bar{r}} \binom{\kappa+1}{r}, \qquad (5.1)$$

where $\bar{r}$ is the smallest integer such that

$$\lfloor n/(\kappa+1)\rfloor(\bar{r}+1) + \max(0, \bar{r}+1-(\kappa+1-n \bmod (\kappa+1))) \geq d.$$

Example 5.1.2. *Let K be the $[31, 5, 16]$ simplex code [24]. Figure 5.1 shows the time in seconds to decode random received vectors by using the extended coset decoding and comparing it with the syndrome decoding implemented in* Magma*. Note that $\rho(K) = 15 = d-1$, so K is a maximal code and both decoding methods perform a complete decoding. According to the implementation in* Magma*, the syndrome decoding uses* 2836 *MB of memory, and the extended coset decoding method uses a negligible amount of memory.*

Apparently, the extended coset decoding has a big advantage if the number of received vectors to be decoded is small. Since the syndrome table

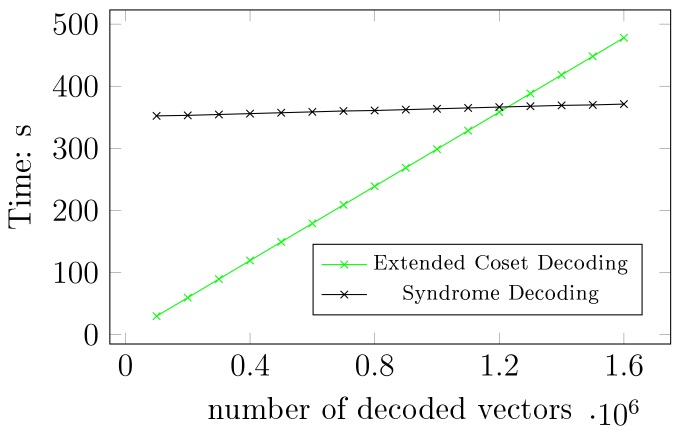


Figure 5.1: Time for decoding using the $[31, 5, 16]$ binary simplex code.

needs to be computed at the first decode procedure, it costs much time on preparation. However, the biggest advantage of the extended coset decoding is on the memory usage. For the syndrome decoding, it is necessary to store a syndrome table in the memory to be used in the decoding process. If this syndrome table is too big to be stored, which will happen when the codimension of the code is moderately large, it will be impossible to decode by syndrome decoding. By contrast, the extended coset decoding based on computing the minimum weight of a linear code does not need to store anything significant, but the very few data needed for the enumeration, which makes the decoding useful for codes with a big error correcting capability, or with a large possible codimension.

Example 5.1.3. *Let K be the $[63, 6, 32]$ simplex code [24]. As the code in Example 5.1.2, $\rho(K) = 31 = d - 1$, so it is a maximal code and both decoding methods allow to perform a complete decoding. By using the syndrome decoding,* MAGMA *returns "Runtime error in 'Decode': Code has too many cosets". However, in this case, we can still perform a complete decoding by using the extended coset decoding method, which takes* 11.83 *seconds to decode* 5000 *random received vectors and uses a negligible amount of memory.*

Coset decoding for binary linear codes

Using the improvement from Algorithm 5 (MinW) to Algorithm 7 (IMinW), shown in Section 4.1, which allows to compute directly the minimum weight of a coset of a linear code, we can describe another decoding method, called *coset decoding.*

For a binary linear code K, we decode a received vector $u = c + e$, where $c \in K$ and e is an error vector, as $c = u - e'$, where e' is a vector of minimum weight in the coset $K + u$. A vector e' of minimum weight in $K + u$ can be found by using an algorithm similar to Algorithm 7 (IMinW), but returning a minimum word of the coset $K + u$ instead of its minimum weight.

Using the coset decoding, the work factor of decoding a received vector u encoded using an $[n, \kappa, d]$ binary linear code K is similar to computing the minimum weight of $K + u$. Again, in order to estimate the work factor, we can use the minimum weight $d = wt(K)$ instead of $wt(K_u)$, and obtain an upper bound of the work factor, since $d \geq wt(K_u)$. Therefore, a work factor upper bound is

$$n\lceil n/\kappa \rceil \sum_{r=1}^{\bar{r}} \binom{\kappa}{r}, \tag{5.2}$$

where $\bar{r}$ is the smallest integer such that

$$\lfloor n/\kappa \rfloor (\bar{r} + 1) + \max(0, \bar{r} + 1 - (\kappa - n \bmod \kappa)) \geq d.$$

Example 5.1.4. *In order to see how both decoding methods perform, we compare their corresponding work factor upper bounds. In all these tests, only the parameters of the codes are considered, which means that the codes have not been constructed, and some of them with certain parameters may not exist.*

We compare the performance of both decoding methods for different $[n, \kappa, d]$ binary linear codes. The advantage of the coset decoding is obvious according to Figures 5.2, 5.3 and 5.4. Note that by increasing the dimension of the code in Figure 5.2, we are in fact increasing the size of the code. In all figures, the work factors are expressed in logarithmic scale.

5.2 Decoding binary nonlinear codes

Both decoding methods, the extended coset decoding and coset decoding, seen for binary linear codes in the previous section, can be generalized to decode nonlinear codes using their coset representation given in Section 3.1.

Extended coset decoding for binary nonlinear codes

Let C be an (n, M, d) binary code, $C = \bigcup_{i=0}^{t}(K_C + v_i)$, with kernel K_C of dimension κ, t coset representatives given by the set $L = \{v_1, \ldots, v_t\}$, and

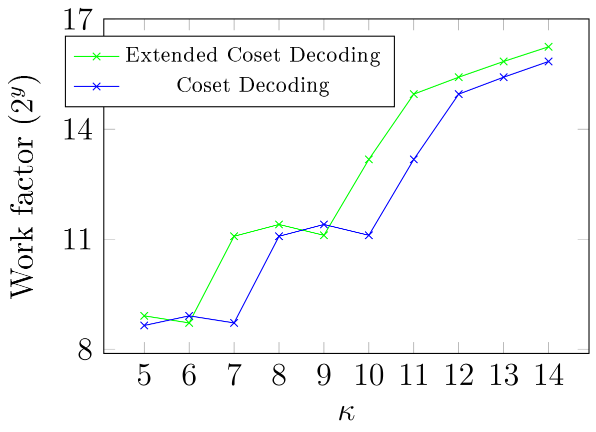


Figure 5.2: Decoding $[20, \kappa, 5]$ binary linear codes, where $\kappa \in \{5, \dots, 14\}$.

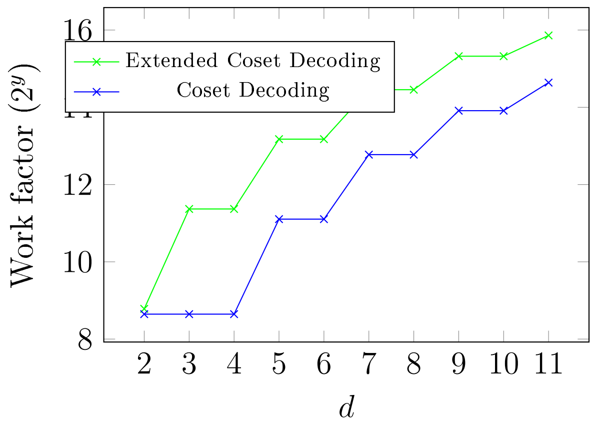


Figure 5.3: Decoding $[20, 10, d]$ binary linear codes, where $d \in \{2, \dots, 11\}$.

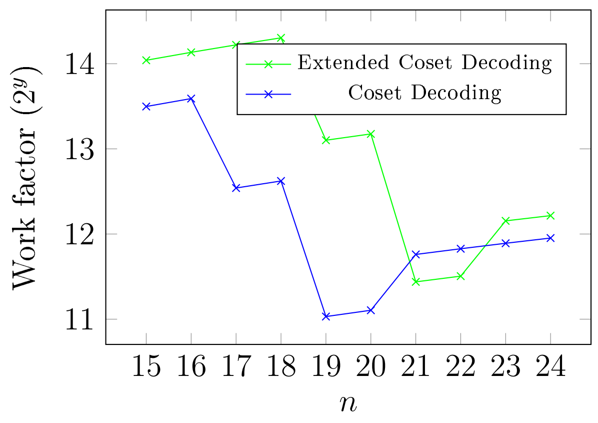


Figure 5.4: Decoding $[n, 10, 5]$ binary linear codes, where $n \in \{15, \ldots, 24\}$.

$v_0 = \mathbf{0}$. In general, according to the nearest neighbor decoding seen in Section 2.4, to decode a received vector $u = c + e$, where $c \in C$ and e is an error vector, we need to find a codeword $c' \in C$ which has the minimum distance to the received vector u. This is the same as finding a vector $e' = u - c'$ of minimum weight in $u - C$, or equivalently in $C + u$, since C is binary. Note that the set $C + u$ can be written as $C + u = \bigcup_{i=0}^{t}(K_C + v_i + u)$.

In order to find a vector of minimum weight in the binary set $C + u = \bigcup_{i=0}^{t}(K_C + v_i + u)$, we can consider the linear extended cosets $K_{v_i+u} = K_C \cup (K_C + v_i + u)$ for all $i \in \{0, \ldots, t\}$, define $C_u = \bigcup_{i=0}^{t} K_{v_i+u}$, and apply the Brouwer-Zimmermann's algorithm for binary linear codes. If the number of errors is less than the minimum weight of K_C, that is $d_H(c, u) = wt_H(e) < wt(K_C)$, then an error vector e' such that $wt_H(e') = wt_H(e)$ can be found as a minimum word in C_u. Such codeword e' can be obtained by applying the Brouwer-Zimmermann's algorithm to each one of the linear extended cosets K_{v_i+u} to get a codeword of minimum weight, denoted by e'_i for all $i \in \{0, \ldots, t\}$, and choosing the one having the minimum weight among them. The following proposition summarizes this decoding process for nonlinear codes, also called *extended coset decoding*.

Proposition 5.2.1. *Let C be a binary nonlinear code with minimum distance d, kernel K_C and coset representatives $\{v_1, \ldots, v_t\}$. For a received vector $u = c + e \notin C$, where $c \in C$, let $C_u = \bigcup_{i=0}^{t} K_{v_i+u}$. If $wt_H(e) < d$, then u can be decoded as $c' = u - e' \in C$, where e' is a vector of minimum weight in C_u, so $wt_H(e) = wt_H(e')$. Note that if $wt_H(e) \leq \lfloor \frac{d-1}{2} \rfloor$, then $e' = e$ and $c' = c$.*

Proof. For a received vector $u \in \mathbb{F}_2^n$, in order to decode it, we look for a vector

$e' \in \mathbb{F}_2^n$ of minimum weight such that $u - e' \in C$. This is equivalent to find a vector e' of minimum weight in $C+u$. We have that $C+u = \bigcup_{i=0}^{t}(K_C+v_i+u)$. Let e'_i be a vector of minimum weight in the linear extended coset K_{v_i+u} for all $i \in \{0,\ldots,t\}$. If $wt_H(e) < d$, then we can take e' as the vector of minimum weight in $\{e'_i \mid i \in \{0,\ldots,t\}\}$, since it is also a vector of minimum weight in $C+u$ such that $wt_H(e) = wt_H(e')$. Then, the received vector u can be decoded as $c' = u - e' \in C$. □

Using the extended coset decoding, the work factor of decoding a received vector u encoded using a binary nonlinear code C is similar to computing the minimum weight of $C_u = \bigcup_{i=0}^{t} K_{v_i+u}$, so it is

$$\sum_{i=0}^{t}\left(n\lceil n/(\kappa+1)\rceil \sum_{r=1}^{\bar{r}_i}\binom{\kappa+1}{r}\right), \tag{5.3}$$

where $\bar{r}_i$ is the smallest integer such that

$$\lfloor n/(\kappa+1)\rfloor(\bar{r}_i+1)+\max(0,\bar{r}_i+1-(\kappa+1-n \bmod (\kappa+1))) \geq wt(K_{v_i+u}).$$

Taking into account that $wt(K_{v_i+u}) \leq wt(K_C)$ for all $i \in \{0,\ldots,t\}$, an upper bound of the previous work factor is

$$(t+1)n\lceil n/(\kappa+1)\rceil \sum_{r=1}^{\bar{r}}\binom{\kappa+1}{r}, \tag{5.4}$$

where $\bar{r}$ is the smallest integer such that

$$\lfloor n/(\kappa+1)\rfloor(\bar{r}+1)+\max(0,\bar{r}+1-(\kappa+1-n \bmod (\kappa+1))) \geq wt(K_C).$$

Note that the performance of the extended coset decoding for nonlinear codes highly depends on the number of coset representatives.

Example 5.2.2. *Let C_κ be a $(31, 2^9\cdot 5, 5)$ binary nonlinear code with a kernel of dimension κ, for $\kappa \in \{5,\ldots,9\}$, and number of cosets t listed in the following table. The following table also shows the time in seconds to decode* 5000 *random received vectors using the extended coset decoding for each one of the codes C_κ, $\kappa \in \{5,\ldots,9\}$.*

κ	*5*	*6*	*7*	*8*	*9*
Number of cosets: t	*79*	*39*	*19*	*9*	*4*
Time: s	*63.30*	*30.35*	*18.53*	*12.57*	*10.82*

Coset decoding for binary nonlinear codes

As for binary linear codes, using the improvement from Algorithm 5 (MinW) to Algorithm 7 (IMinW), shown in Section 4.1, which allows to compute directly the minimum weight of a coset of a linear code, we can also describe another method, the coset decoding for binary nonlinear codes. In this case, for a binary nonlinear code C, a vector e' of minimum weight in $C + u = \bigcup_{i=0}^{t}(K_C + v_i + u)$ is obtained as the vector of minimum weight in $\{e_i \mid i \in \{0, \dots, t\}\}$, where e_i is a vector of minimum weight in $K_C + v_i + u$, computed using this improvement of the algorithm, but returning a minimum word instead of the minimum weight.

Using the coset decoding, the work factor of decoding a received vector u encoded using a binary nonlinear code C is similar to computing the minimum weight of $C + u = \bigcup_{i=0}^{t}(K_C + v_i + u)$, so it is

$$\sum_{i=0}^{t} \left(n \lceil n/\kappa \rceil \sum_{r=1}^{\bar{r}_i} \binom{\kappa}{r} \right), \tag{5.5}$$

where $\bar{r}_i$ is the smallest integer such that

$$\lfloor n/\kappa \rfloor (\bar{r}_i + 1) + \max(0, \bar{r}_i + 1 - (\kappa - n \bmod \kappa)) \geq wt(K_{v_i+u}).$$

Again, taking into account that $wt(K_{v_i+u}) \leq wt(K_C)$ for all $i \in \{0, \dots, t\}$, an upper bound of the previous work factor is

$$(t+1) n \lceil n/\kappa \rceil \sum_{r=1}^{\bar{r}} \binom{\kappa}{r}, \tag{5.6}$$

where $\bar{r}$ is the smallest integer such that

$$\lfloor n/\kappa \rfloor (\bar{r} + 1) + \max(0, \bar{r} + 1 - (\kappa - n \bmod \kappa)) \geq wt(K_C).$$

Example 5.2.3. *As in Example 5.1.4, in order to see how both decoding methods perform, we compare their corresponding work factor upper bounds. We compare the performance of both decoding methods for different $(n, 2^{14} \cdot 3, d)$ binary nonlinear codes with a kernel of dimension κ. The advantage of the coset decoding is also obvious according to Figure 5.5, 5.6 and 5.7. Note that, unlike Example 5.1.4, by changing the dimension of the kernel in Figure 5.5, we are only changing the linearity of the code, but the size of the code remains the same. In all figures, the work factors are expressed in logarithmic scale.*

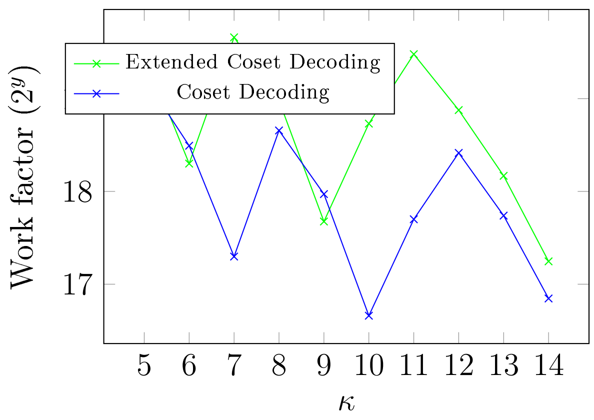


Figure 5.5: Decoding $(20, 2^{14} \cdot 3, 5)$ binary nonlinear codes with kernels of dimension $\kappa \in \{5, \ldots, 14\}$.

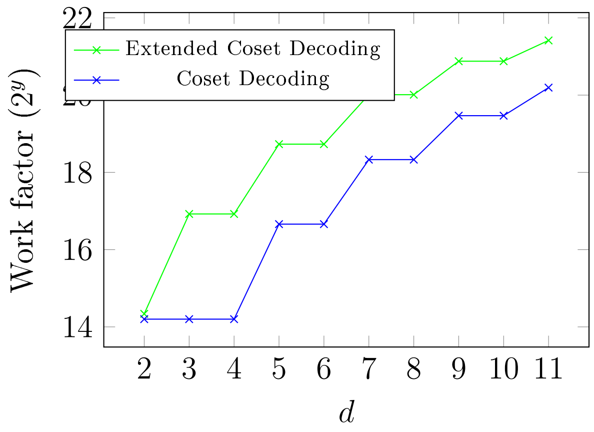


Figure 5.6: Decoding $(20, 2^{14} \cdot 3, d)$ binary nonlinear codes, where $d \in \{2, \ldots, 11\}$, and with kernels of dimension $\kappa = 10$.

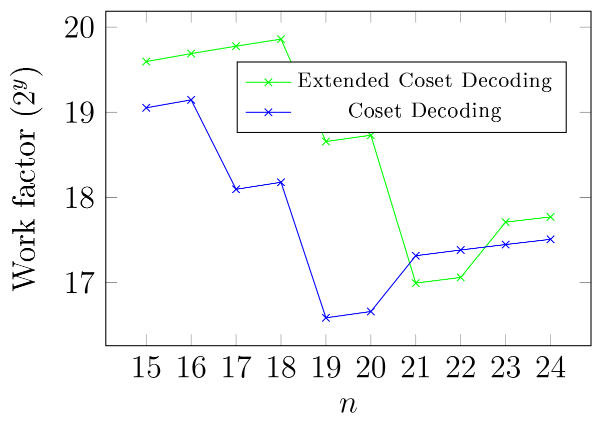


Figure 5.7: Decoding $(n, 2^{14} \cdot 3, 5)$ binary nonlinear codes, where $n \in \{15, \dots, 24\}$, and with kernels of dimension $\kappa = 10$.

5.3 Decoding q-ary linear codes

The decoding methods seen for binary linear codes in Section 5.1 can be generalized into q-ary linear codes.

Extended coset decoding for q-ary linear codes

Let K be an $[n, \kappa, d]$ q-ary linear code. Let $v \in \mathbb{F}_q^n \backslash K$. Recall that the linear span $K_v = \langle K, v \rangle = \bigcup_{\lambda \in \mathbb{F}_q} (K + \lambda v)$ of dimension $\kappa + 1$ is called an *extended coset* of K respect to v.

Proposition 5.3.1. *Let $K_v = \bigcup_{\lambda \in \mathbb{F}_q} (K + \lambda v)$ be an extended coset of K respect to a vector $v \in \mathbb{F}_q^n \backslash K$. If $d(K_v) < d(K)$, where $d(K_v)$ is the minmum distance of K_v and $d(K)$ is the minimnum distance of K, there is at least one vector $e_\lambda \in K + \lambda v$ such that $wt_H(e_\lambda) = d(K_v)$, for all $\lambda \in \mathbb{F}_q \backslash \{\mathbf{0}\}$.*

Proof. If $d(K_v) < d(K)$, the vector e of minimum weight in K_v satisfies that $wt_H(e) = d(K_v)$ and $e \notin K$. Assume that $e = k + \lambda_1 v \in K + \lambda_1 v$, where $k \in K$ and $\lambda_1 \in \mathbb{F}_q \backslash \{\mathbf{0}\}$. For any $\lambda \in \mathbb{F}_q \backslash \{\mathbf{0}\}$, there is $\lambda_2 = \lambda / \lambda_1$ such that $e_\lambda = \lambda_2 e = \lambda_2 k + \lambda_1 \lambda_2 v = \lambda_2 k + \lambda v \in K + \lambda v$. Since $wt_H(e_\lambda) = wt_H(\lambda_2 e) = wt_H(e)$, the result follows. □

As mentioned in Section 5.1, according to the nearest neighbor decoding, in order to decode a received vector $u = c + e$, where $c \in K$ and e is an error vector, we need to find a codeword $c' \in K$ having the minimum distance

algorithm 9 Extended coset decoding of q-ary linear codes.

```
Data: A $q$-ary linear code $K$ and a received vector $u$.
Result: The decoded codeword $c' \in K$.
begin
    $K_u \leftarrow \langle K, u \rangle$
    $e' \leftarrow \text{MinWord}(K_u)$
    for $\lambda \in \mathbb{F}_q$ do
        if $\lambda e' - u \in K$ then
            $c' \leftarrow u - \lambda e'$
            break
    return $c'$.
```

to the received vector u, which is equivalent to find a vector $e' = u - c'$ of minimum weight in $u - K$. Since K is linear, $K = -K$, so as for linear codes, the decoding process is equivalent to find a minimum word in $K + u$.

Again, as for binary linear codes, in order to find a vector of minimum weight in a coset $K + u$ of a linear code K, we can consider the linear extended coset $K_u = \langle K, u \rangle = \bigcup_{\lambda \in \mathbb{F}_q}(K + \lambda u)$, of K respect to u, and apply the Brouwer-Zimmermann's algorithm for linear codes. If the number of errors is less than $d(K) = d$, that is, $wt_H(e) < d$, according to Proposition 5.3.1, we can find an error vector $e' \in K + u$ such that $wt_H(e') = wt_H(e)$. This decoding process, also called *extended coset decoding*, is explained in Proposition 5.3.2 and described in Algorithm 9.

Proposition 5.3.2. *Let K be a q-ary linear code with minimum distance d. For a received vector $u = c + e \notin K$, where $c \in K$, let $K_u = \bigcup_{\lambda \in \mathbb{F}_q}(K + \lambda u)$. If $wt_H(e) < d$, then u can be decoded as $c' = u - e' \in K$, where e' is a vector of minimum weight in K_u such that $e' \in K + u$, so $wt_H(e) = wt_H(e')$. Note that if $wt_H(e) \leq \lfloor \frac{d-1}{2} \rfloor$, then $e' = e$ and $c' = c$.*

Proof. For a received vector $u = c + e \in \mathbb{F}_q^n \backslash K$, where $c \in K$, in order to decode it, we look for a vector e' of minimum weight such that $c' = u - e' \in K$. This is equivalent to find a vector e' of minimum weight in the coset containing u, which is $K + u$.

Let e'' be a vector of minimum weight in the linear extended coset $K_u = \bigcup_{\lambda \in \mathbb{F}_q}(K + \lambda u)$. Since the minimum weight of the coset $K + u$ is less than d, by Proposition 5.3.1, there is a $\lambda \in \mathbb{F}_q \backslash \{\mathbf{0}\}$ such that $e' = \lambda e'' \in K + u$. We have that $d(K_u) = wt_H(\lambda e'') = wt_H(e') = wt_H(e) < d$. Then, the received vector u can be decoded as $c' = u - e' \in K$. □

Example 5.3.3. *Let K be a random $[20, 5, 3]$ ternary linear code. Figure 5.8 shows the time in seconds to decode random received vectors by using the extended coset decoding and comparing it with the syndrome decoding implemented in* MAGMA. *According to the implementation in* MAGMA, *the syndrome decoding uses* 1620 *MB of memory, and the extended coset decoding method uses a negligible amount of memory.*

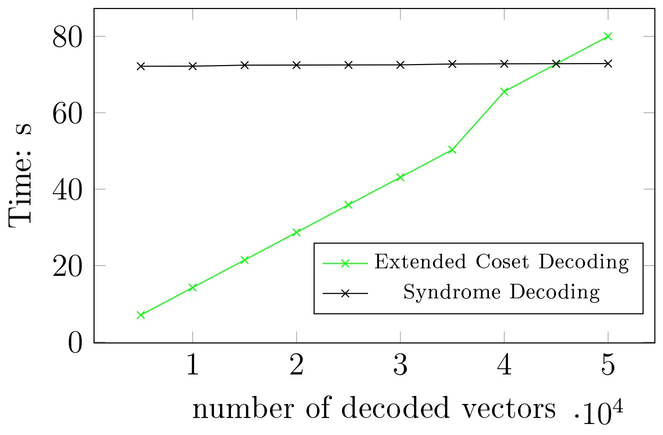


Figure 5.8: Time for decoding using a random $[20, 5, 3]$ ternary linear code.

Example 5.3.4. *Let C_n be a random $[n, 5]$ ternary linear code, for $n \in \{15, \dots, 20\}$. Figure 5.9 shows the time in seconds to decode* 5000 *random received vectors by using the extended coset decoding and comparing it with the syndrome decoding implemented in* MAGMA. *It is clear that as the length grows, the time cost grows significantly by using the syndrome decoding, so the extended coset decoding is more efficient for linear codes with bigger length.*

Using the extended coset decoding, the work factor of decoding a received vector u encoded using an $[n, \kappa, d]$ q-ary linear code K is similar to computing the minimum weight of K_u. Since $d \geq wt(K_u)$, we can obtain an upper bound of the work factor, which is

$$\log_2(q) n \lceil n/(\kappa+1) \rceil \sum_{r=1}^{\bar{r}} \binom{\kappa+1}{r} (q-1)^{r-1}, \tag{5.7}$$

where $\bar{r}$ is the smallest integer such that

$$\lfloor n/(\kappa+1) \rfloor (\bar{r}+1) + \max(0, \bar{r}+1-(\kappa+1-n \bmod (\kappa+1))) \geq d.$$

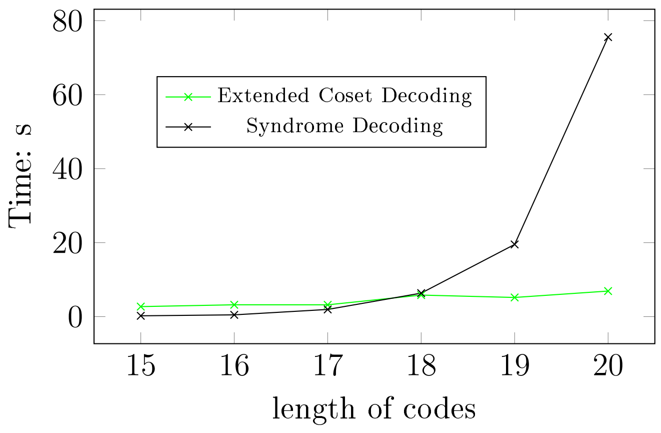


Figure 5.9: Time for decoding 5000 vectors using random $[n, 5]$ ternary codes for $n \in \{15, \ldots, 20\}$.

Coset decoding for q-ary linear codes

Using the coset decoding, as it is described in Section 5.1 for binary linear codes, we can find directly an error vector in the coset $K + u$. In this case, the work factor of decoding a received vector u encoded using an $[n, \kappa, d]$ q-ary linear code K is similar to computing the minimum weight of $K + u$. Again, since $d \geq wt(K_u)$, we can obtain an upper bound of the work factor, which is

$$\log_2(q) n \lceil n/\kappa \rceil \sum_{r=1}^{\bar{r}} \binom{\kappa}{r} (q-1)^{r-1}, \tag{5.8}$$

where $\bar{r}$ is the smallest integer such that

$$\lfloor n/\kappa \rfloor (\bar{r} + 1) + \max(0, \bar{r} + 1 - (\kappa - n \bmod \kappa)) \geq d.$$

Example 5.3.5. *As in Example 5.1.4, in order to see how both decoding methods perform, we compare their corresponding work factor upper bounds. Specifically, we compare the performance of both decoding methods for different* $[n, \kappa, d]$ *ternary linear codes, and different* $[20, 10, 5]$ *q-ary linear codes. Again, the advantage of the coset decoding is obvious according to Figures 5.10, 5.11, 5.12 and 5.13. Note that by changing the cardinality q of the field in Figure 5.13, the size of the code $M = q^{10}$ varies since it depends on q. In all figures, the work factors are expressed in logarithmic scale.*

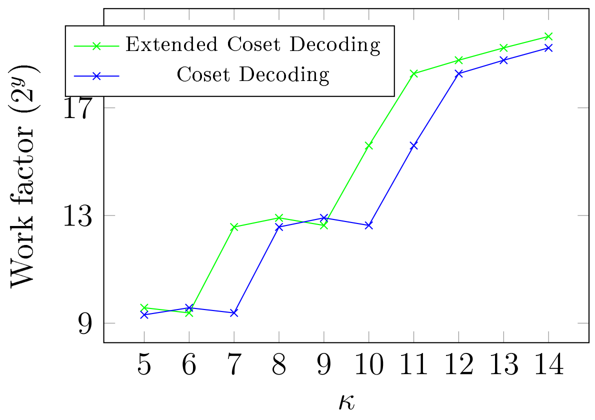


Figure 5.10: Decoding $[20, \kappa, 5]$ ternary linear codes, where $\kappa \in \{5, \ldots, 14\}$.

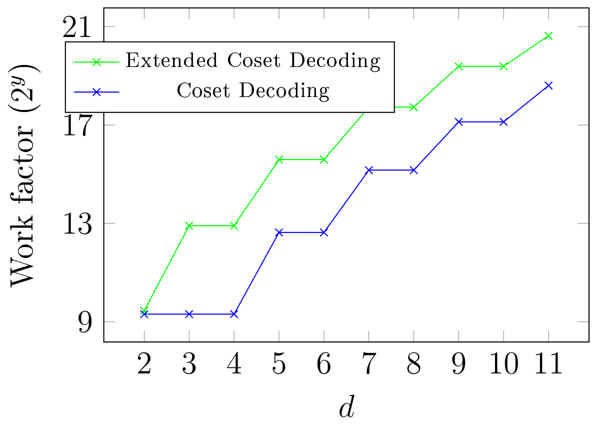


Figure 5.11: Decoding $[20, 10, d]$ ternary linear codes, where $d \in \{2, \ldots, 11\}$.

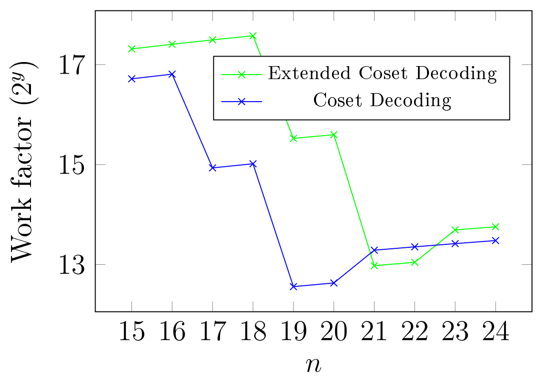


Figure 5.12: Decoding $[n, 10, 5]$ ternary linear codes, where $n \in \{15, \ldots, 24\}$.

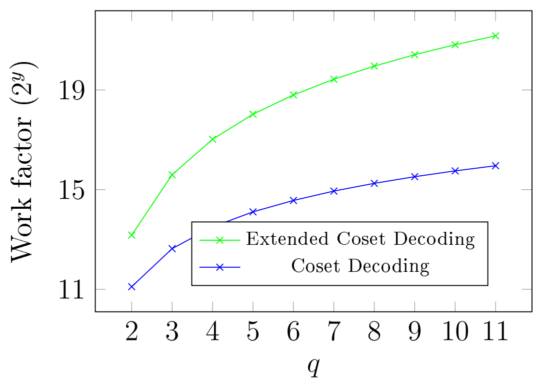


Figure 5.13: Decoding $[20, 10, 5]$ q-ary linear codes for $q \in \{2, \ldots, 11\}$.

5.4 Decoding q-ary nonlinear codes

As for the binary linear case, the decoding methods seen for binary nonlinear codes in Section 5.2 can also be generalized into q-ary nonlinear codes.

Extended coset decoding for q-ary nonlinear codes

Let C be an (n, M, d) q-ary code, $C = \bigcup_{i=0}^{t}(K_C + v_i)$, with kernel K_C of dimension κ, t coset representatives given by the set $L = \{v_1, \ldots, v_t\}$, and $v_0 = \mathbf{0}$. As we have mentioned before, according to the nearest neighbor decoding, in order to decode a received vector $u = c + e$, where $c \in C$ and e is an error vector, we need to find a codeword $c' \in C$ having the minimum distance to the received vector u, which is equivalent to find a vector $e' = u - c'$ of minimum weight in $u - C$, as for binary nonlinear codes. Note that the set $u - C$ can be written as $u - C = \bigcup_{i=0}^{t}(K_C + u - v_i)$, since $-K_C = K_C$.

Again, as for binary nonlinear codes, in order to find a vector of minimum weight in the set $u - C = \bigcup_{i=0}^{t}(K_C + u - v_i)$, we can consider the linear extended cosets $K_{u-v_i} = \langle K_C, u - v_i\rangle = \bigcup_{\lambda\in\mathbb{F}_q}(K_C + \lambda(u - v_i))$ for all $i \in \{0, \ldots, t\}$, define $C_u = \bigcup_{i=0}^{t} K_{u-v_i}$, and apply the Brouwer-Zimmermann's algorithm for linear codes. If the number of errors is less than the minimum weight of K_C, that is $d_H(c, u) = wt_H(e) < wt(K_C)$, then an error vector e' such that $wt_H(e') = wt_H(e)$ can be found as a codeword of minimum weight in C_u such that $e' \in u - C = \bigcup_{i=0}^{t}(K_C + u - v_i)$. Such codeword e' can be obtained by applying the Brouwer-Zimmermann's algorithm to each one of the linear extended cosets K_{u-v_i} to get first a codeword e''_i of minimum weight, for all $i \in \{0, \ldots, t\}$. Then, a codeword $e'_i \in K_C + u - v_i$ such that $wt_H(e''_i) = wt_H(e'_i)$ can be generated from e''_i, by Proposition 5.3.1. Finally, e' is chosen as a vector of minimum weight in $\{e'_i \mid i \in \{0, \ldots, t\}\}$. This decoding process for nonlinear codes, also called *extended coset decoding*, is explained in Proposition 5.4.1 and described in Algorithm 10.

Proposition 5.4.1. *Let C be a q-ary nonlinear code with minimum distance d, kernel K_C and coset representatives $\{v_1, \ldots, v_t\}$. For a received vector $u = c + e \notin C$, where $c \in C$, let $C_u = \bigcup_{i=0}^{t} K_{u-v_i}$. If $wt_H(e) < d$, then u can be decoded as $c' = u - e' \in C$, where e' is a vector of minimum weight in C_u such that $e' \in u - C = \bigcup_{i=0}^{t}(K_C + u - v_i)$, so $wt_H(e) = wt_H(e')$. Note that if $wt_H(e) \leq \lfloor \frac{d-1}{2} \rfloor$, then $e' = e$ and $c' = c$.*

Proof. For a received vector $u \in \mathbb{F}_q^n$, in order to decode it, we look for a vector $e' \in \mathbb{F}_q^n$ of minimum weight such that $u - e' \in C$. This is equivalent to find a

algorithm 10 Extended coset decoding of q-ary nonlinear codes.

```
Data: A $q$-ary nonlinear code $C$ given by the kernel $K_C$ and coset representatives $L = \{v_1, \dots, v_t\}$, and a received vector $u$.
Result: The decoded codeword $c' \in C$.
begin
    minWeight $\leftarrow$ Length($C$)
    for $v_i \in L \cup \{\mathbf{0}\}$ do
        $K_{u-v_i} \leftarrow \langle K_C, u - v_i \rangle$
        $e' \leftarrow$ MinWord($K_{u-v_i}$)
        if $wt_H(e') <$ minWeight then
            minWeight $\leftarrow wt_H(e')$
            for $\lambda \in \mathbb{F}_q$ do
                if $\lambda e' - (u - v_i) \in K_C$ then
                    $c' \leftarrow u - \lambda e'$
                    break
    return $c'$
```

vector e' of minimum weight in $u-C$. We have that $u-C = \bigcup_{i=0}^{t}(u-K_C-v_i)$. Since K_C is linear, $-K_C = K_C$. Therefore, $u-C = \bigcup_{i=0}^{t}(K_C+u-v_i)$. Let e'_i be a vector of minimum weight in the linear extended coset K_{u-v_i} such that $e'_i \in K_C+u+v_i$ for all $i \in \{0, \dots, t\}$. The vector e'_i exists by Proposition 5.3.1 and because $wt_H(e) < d$. Finally, we can take e' as the vector of minimum weight in $\{e'_i \mid i \in \{0, \dots, t\}\}$, since it is also a vector of minimum weight in $u - C$ such that $wt_H(e) = wt_H(e')$. Then, the received vector u can be decoded as $c' = u - e' \in C$. □

Example 5.4.2. *Let C_κ be a $(20, 3^{10}, 3)$ ternary nonlinear code with a kernel of dimension κ, for $\kappa \in \{5, \dots, 9\}$, and number of cosets t listed in the following table. The following table also shows the time in seconds to decode* 5000 *random received vectors using the extended coset decoding for each one of the codes C_κ, $\kappa \in \{5, \dots, 9\}$.*

κ	5	6	7	8	9
Number of cosets: t	*242*	*80*	*26*	*8*	*2*
Time: s	*280.680*	*96.830*	*33.700*	*14.800*	*8.420*

Using the extended coset decoding, the work factor of decoding a received vector u encoded using a q-ary nonlinear code C is similar to computing the minimum weight of $C_u = \bigcup_{i=0}^{t} K_{u-v_i}$, so it is

$$\sum_{i=0}^{t} \Big(\log_2(q) n \lceil n/(\kappa+1) \rceil \sum_{r=1}^{\bar{r}_i} \binom{\kappa+1}{r} (q-1)^{r-1} \Big), \tag{5.9}$$

where $\bar{r}_i$ is the smallest integer such that

$$\lfloor n/(\kappa+1) \rfloor (\bar{r}_i + 1) + \max(0, \bar{r}_i + 1 - (\kappa + 1 - n \bmod (\kappa+1))) \geq wt(K_{u-v_i}).$$

Taking into account that $wt(K_{u-v_i}) \leq wt(K_C)$ for all $i \in \{0, \ldots, t\}$, an upper bound of the previous work factor is

$$(t+1) \log_2(q) n \lceil n/(\kappa+1) \rceil \sum_{r=1}^{\bar{r}} \binom{\kappa+1}{r} (q-1)^{r-1}, \tag{5.10}$$

where $\bar{r}$ is the smallest integer such that

$$\lfloor n/(\kappa+1) \rfloor (\bar{r} + 1) + \max(0, \bar{r} + 1 - (\kappa + 1 - n \bmod (\kappa+1))) \geq wt(K_C).$$

Coset decoding for q-ary nonlinear codes

Using the coset decoding, as it is described in Section 5.2 for binary nonlinear codes, we can find directly an error vector in $u - C$. In this case, the work factor of decoding a received vector u encoded using a q-ary nonlinear code C is similar to computing the minimum weight of $u - C = \bigcup_{i=0}^{t} (K_C + u - v_i)$, so it is

$$\sum_{i=0}^{t} \Big(\log_2(q) n \lceil n/\kappa \rceil \sum_{r=1}^{\bar{r}_i} \binom{\kappa}{r} (q-1)^{r-1} \Big), \tag{5.11}$$

where $\bar{r}_i$ is the smallest integer such that

$$\lfloor n/\kappa \rfloor (\bar{r}_i + 1) + \max(0, \bar{r}_i + 1 - (\kappa - n \bmod \kappa)) \geq wt(K_{u-v_i}).$$

Again, taking into account that $wt(K_{u-v_i}) \leq wt(K_C)$ for all $i \in \{0, \ldots, t\}$, an upper bound of the previous work factor is

$$(t+1) \log_2(q) n \lceil n/\kappa \rceil \sum_{r=1}^{\bar{r}} \binom{\kappa}{r} (q-1)^{r-1}, \tag{5.12}$$

where $\bar{r}$ is the smallest integer such that

$$\lfloor n/\kappa \rfloor (\bar{r} + 1) + \max(0, \bar{r} + 1 - (\kappa - n \bmod \kappa)) \geq wt(K_C).$$

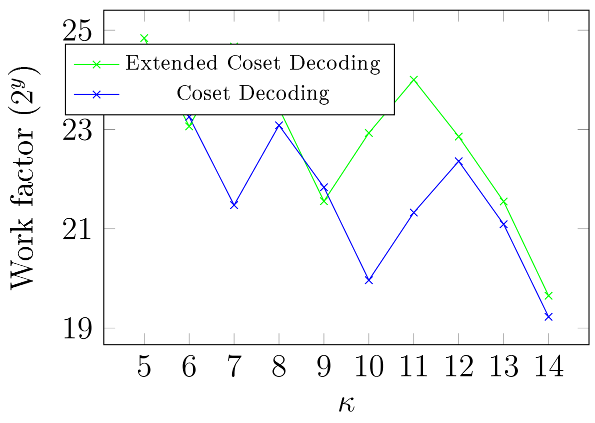


Figure 5.14: Decoding $(20, 3^{14} \cdot 2, 5)$ ternary nonlinear codes with kernels of dimension $\kappa \in \{5, \dots, 14\}$.

Example 5.4.3. *As in Example 5.1.4, in order to see how both decoding methods perform, we compare their corresponding work factor upper bounds. Specifically, we compare the performance of both decoding methods for different $(n, 3^{14} \cdot 2, d)$ ternary nonlinear codes with kernels of dimension κ, and different $(20, q^5 \cdot 60, 3)$ q-ary nonlinear codes with kernels of dimension $\kappa = 5$. Again, the advantage of the coset decoding is obvious according to Figures 5.14, 5.15, 5.16 and 5.17. Note that, unlike Example 5.1.4, by changing the dimension of the kernel in Figure 5.14, we are only changing the linearity of the code, but the size of the code remains the same. Also note that by changing the cardinality q of the field in Figure 5.17, the size of the code $M = q^5 \cdot 60$ varies since it depends on q. In all figures, the work factors are expressed in logarithmic scale.*

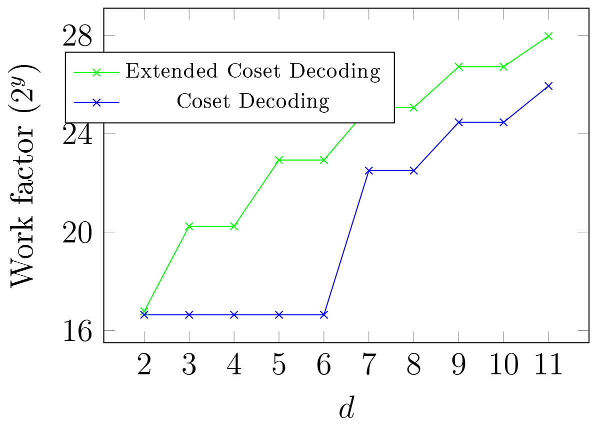


Figure 5.15: Decoding $(20, 3^{14} \cdot 2, d)$ ternary nonlinear codes, where $d \in \{2, \ldots, 11\}$, and with kernels of dimension $\kappa = 10$.

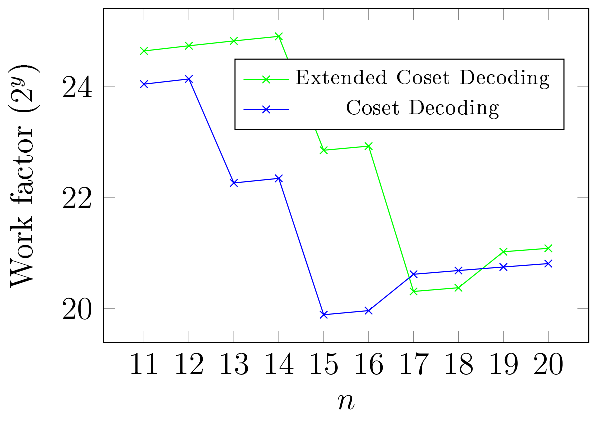


Figure 5.16: Decoding $(n, 3^{14} \cdot 2, 3)$ ternary nonlinear codes, where $n \in \{11, \ldots, 20\}$, and with kernels of dimension $\kappa = 5$.

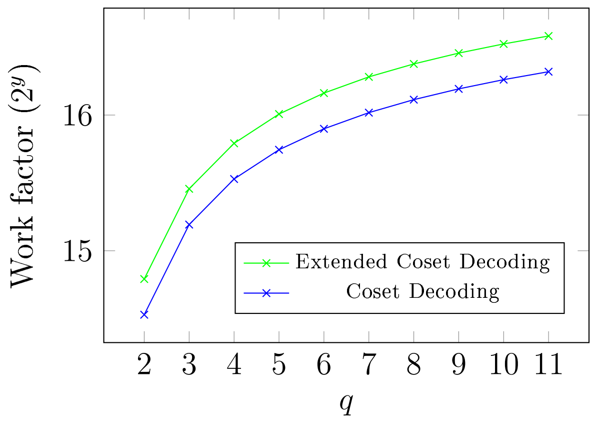


Figure 5.17: Decoding $(20, q^5 \cdot 60, 3)$ q-ary nonlinear codes with kernels of dimension $\kappa = 5$, for $q \in \{2, \ldots, 11\}$.

Chapter 6

MAGMA package implementation

MAGMA is a software system designed to solve computationally hard problems in algebra, number theory, geometry and combinatorics [9]. MAGMA currently supports the basic facilities for codes over finite fields and codes over integer residue rings and Galois rings, all that are linear codes.

The *Combinatoric, Coding and Security Group* (CCSG) [34] has been working on extending the MAGMA system implementing new packages to work efficiently with $\mathbb{Z}_2\mathbb{Z}_4$-linear codes, completely regular codes, and nonlinear codes in general [27, 5]. This dissertation is focused on the latter research area, that is, on extending the functionality of a new package to work with nonlinear codes in general.

In this chapter, we describe the main characteristics of the package implementation, related to the representation of binary nonlinear codes, constructions, minimum distance computation and decoding. The last version of the package for binary codes can be downloaded from the CCSG web site `http://ccsg.uab.cat` together with a manual describing all functions. For q-ary codes, only some functions have been implemented and the package is far to be finished.

6.1 Enumeration of vectors

All algorithms based on Brouwer-Zimmermann's algorithm for linear codes make extensively use of the enumeration of vectors in $\mathbb{F}_2^n$ ordered by their Hamming weight. Thus, we must generate the 2^n vectors $v_0, v_1, v_2, \ldots$ of $\mathbb{F}_2^n$ in such a way that $wt_H(v_i) \leq wt_H(v_{i+1})$. In order to do that, we need to generate step by step the sequence of all combinations of t elements from a set of n natural numbers. Each combination of t elements represents a vector $v \in \mathbb{F}_2^n$ such that $wt_H(v) = t$, given by its support. This algorithm implemented in the package, called resolving-door algorithm, is based in a research paper of W.H. Payne and explained in the book of D.E. Knuth [22]. Moreover, given a combination L, we need to know the position of L within the sequence of all combinations of size less than or equal to t.

$\boxed{\texttt{InitializeComb(}n, t\texttt{)}}$

Initialize a $t+1$ $(1 < t < n)$ sequence of natural numbers in increasing order. The $t+1$ position is the sentinel. Returns the sequence $L = [0, 1, \ldots, t-1, n]$.

$\boxed{\texttt{NextComb(}L, n, t, R3\texttt{)}}$

Compute the next combination from a current combination L of parameters n and t. $R3$ is `false` for the first step and `true` for the following. This is an iterative implementation of the resolving-door algorithm.

$\boxed{\texttt{PositionComb(}L\texttt{)}}$

Given a combination L, return the position of L in the sequence of combinations obtained from the resolving-door algorithm. The combination $c_1c_2\cdots c_t$ is visited by the revolving-door algorithm exactly in the position $\sum_{i=1}^{t}(-1)^i(-1)^{t+1}\binom{c_i+1}{i}) + 1$ if $t+1$ is even, and $\sum_{i=1}^{t}(-1)^i(-1)^{t+1}\binom{c_i+1}{i})$ if $t+1$ is odd.

6.2 Kernel computation

Another set of functions in the package is the one related to the kernel computation. The kernel and coset representatives of a code are computed whenever we construct a code from the list of its codewords, or we construct a new code from existing ones. For example, the package contains the following functions:

$\boxed{\texttt{BinaryCode(}L\texttt{)}}$

Create a binary code C given by its superdual, where L is a sequence of elements of $\mathbb{Z}_2^n$, a subspace of $\mathbb{Z}_2^n$, a $m \times n$ matrix A over $\mathbb{Z}_2$, a binary linear code, a quaternary linear code or a $\mathbb{Z}_2\mathbb{Z}_4$-additive code. The kernel and coset representatives are computed using Algorithm 2, described in Section 3.1, in the cases where it is necessary.

This constructor appends five attributes to the code category:

- `Length`: The length n of the binary code.
- `Kernel`: The kernel of the binary code as a binary linear subcode.
- `CosetLeaders`: The sequence of coset leaders $[v_1, \dots, v_t]$.
- `MinimumDistance`: The minimum (Hamming) distance.
- `IsLinear`: It is `true` if and only if C is a binary linear code.

`BinaryCode(`$L, K, Lids$`)`

Create a binary code C given by its superdual, where L is a sequence of elements of $\mathbb{Z}_2^n$, a partial kernel K, and the corresponding sequence of partial coset representatives *Lids*. In this case, the final kernel and coset representatives are computed using Algorithm 2 and Proposition 3.1.1, shown in Section 3.1.

This constructor appends five attributes to the code category:

- `Length`: The length n of the binary code.
- `Kernel`: The kernel of the binary code as a binary linear subcode.
- `CosetLeaders`: The sequence of coset leaders $[v_1, \dots, v_t]$.
- `MinimumDistance`: The minimum (Hamming) distance.
- `IsLinear`: It is `true` if and only if C is a binary linear code.

Example 6.2.1. *We can define a binary code by giving a sequence of elements of a vector space $V = \mathbb{Z}_2^n$, or a matrix over $\mathbb{Z}_2$. Note that the `BinaryCode` function returns the super dual of a binary code C and the codewords of C can be generated as the union of the cosets of its kernel.*

```
> V  := VectorSpace(GF(2),4);
> L  := [V!0,V![1,0,0,0],V![0,1,0,1],V![1,1,1,1]];
> C1 := BinaryCode(L);
> C1;
[7, 4, 2] Linear Code over GF(2)
Generator matrix:
[1 0 0 0 1 0 1]
[0 1 0 0 0 1 1]
[0 0 1 0 0 0 1]
```

```
[0 0 0 1 0 1 1]
> IsBinaryLinearCode(C1);
false

> A  := Matrix(L);
> C2 := BinaryCode(A);
> C2;
[7, 4, 2] Linear Code over GF(2)
Generator matrix:
[1 0 0 0 1 0 1]
[0 1 0 0 0 1 1]
[0 0 1 0 0 0 1]
[0 0 0 1 0 1 1]
> IsBinaryEqual(C1, C2);
true
```

A binary linear code C can be generated as a binary code with this constructor. Note that in this case the ***BinaryCode*** *function returns the dual of C.*

```
> C := LinearCode(sub<V|[[0,0,1,1],[1,0,1,1]]>);
> D := BinaryCode(sub<V|[[0,0,1,1],[1,0,1,1]]>);
> D;
[4, 2, 1] Linear Code over GF(2)
Generator matrix:
[0 1 0 0]
[0 0 1 1]
> IsBinaryLinearCode(D);
true
> Dual(C) eq D;
true
```

6.3 Construction of binary codes

Using the above functions to compute the kernel and coset representatives, and the results given in Section 3.2, we have implemented new functions to construct new binary nonlinear codes from existing ones. Specifically, the package contains the following functions:

$\boxed{\texttt{BinaryIntersection}(C, D)}$

Create the intersection of two binary codes C and D, both of the same length. The binary codes C and D are given by their kernels and coset representatives. Proposition 3.2.5 is used in the implementation.

$\boxed{\texttt{BinaryUnion}(C, D)}$

Create the union of two binary codes C and D, both of the same length. The binary codes C and D are given by their kernels and coset representatives. Proposition 3.2.6 is used in the implementation.

Example 6.3.1. *We verify some simple results from the union and intersection of binary codes.*

```
> C1 := BinaryRandomCode(5,4);
> C2 := BinaryRandomCode(5,4);
> C1UnionC2 := BinaryUnion(C1,C2);
> C1InterC2 := BinaryIntersection(C1,C2);
> BinaryCardinal(C1) + BinaryCardinal(C2) eq
> BinaryCardinal(C1UnionC2) + BinaryCardinal(C1InterC2);
true

> U := BinaryUniverseCode(5);
> IsBinaryEqual(BinaryUnion(C1,U),U);
true
> IsBinaryEqual(BinaryIntersection(C1,U),C1);
true

> Z := BinaryZeroCode(5);
> IsBinaryEqual(BinaryUnion(C1,Z),C1);
true
> IsBinaryEqual(BinaryIntersection(C1,Z),Z);
true
```

$\boxed{\texttt{BinaryExtendedCode}(C)}$

Given a binary code C form a new binary code C' from C by adding the appropriate extra coordinate to each vector of C such that the sum of the coordinates of the extended vector is zero. Proposition 3.2.7 is used in the implementation.

$\boxed{\texttt{BinaryPuncturedCode}(C, i)}$

Given a binary code C of length n and an integer i, $1 \leq i \leq n$, construct a new binary code C' by deleting the i-th coordinate from each codeword of C. Proposition 3.2.9 is used in the implementation.

$\boxed{\texttt{BinaryShortenedCode}(C, i)}$

Given a binary code C of length n and an integer i, $1 \leq i \leq n$, construct a new binary code from C by selecting only those codewords of C having a zero as their i-th component and deleting the i-th component from these codewords. Thus, the resulting code will have length $n - 1$. Proposition 3.2.11 is used in the implementation.

$\boxed{\texttt{BinaryDirectSum}(C, D)}$

Given binary codes C and D, construct the direct sum of C and D. The direct sum is a binary code that consists of all vectors of the form (u, v), where $u \in C$ and $v \in D$. Proposition 3.2.13 is used in the implementation.

$\boxed{\texttt{BinaryPlotkinSum}(C, D)}$

Given binary codes C and D both of the same length, construct the Plotkin sum of C and D. The Plotkin sum is a binary code that consists of all vectors of the form $(u, u+v)$, where $u \in C$ and $v \in D$. Proposition 3.2.15 is used in the implementation.

Example 6.3.2. *We combine binary codes in different ways and look at the length of the new binary codes.*

```
> C1 := BinaryRandomCode(5,4);
> C2 := BinaryRandomCode(7,3);
> BinaryLength(C1);
5
> BinaryLength(C2);
7

> C3 := BinaryDirectSum(C1,C2);
> BinaryLength(C3);
12
> C4 := BinaryDirectSum([C1,C2,C3]);
> BinaryLength(C4);
24
> C5 := BinaryExtendCode(C2);
> BinaryLength(C5);
8
> C6 := BinaryPunctureCode(C2,4);
> BinaryLength(C6);
6
> C7 := BinaryShortenCode(C2,{4,5});
> BinaryLength(C7);
5
> C8 := BinaryPlotkinSum(C2,C2);
> BinaryLength(C8);
14
```

6.4 Minimum distance of binary codes

Using the algorithms described in Section 4.1, we have implemented new functions to compute the minimum weight and minimum distance of binary nonlinear codes.

`BinaryMinimumWeight(`C` : parameter)`

`AlgMethod` MonStgElt *Default* : "MinWeightCoset"

Compute the minimum weight of a binary code C given by the kernel and coset representatives. The accessible algorithms are: "MinWeightExtendCoset" (Algorithm 5) and "MinWeightCoset" (Algorithm 7). The "MinWeightCoset" algorithm is used by default, and the "MinWeightExtendCoset" will be used if the parameter `AlgMethod` is assigned the value "MinWeightExtendCoset".

Note that the "MinWeightExtendCoset" algorithm can be faster than the "MinWeightCoset" algorithm, since it uses functions already integrated into the Magma system.

`BinaryMinimumDistance(`C` : parameter)`

`AlgMethod` MonStgElt *Default* : "MinWeightCoset"

Compute the minimum distance of a binary code C given by the kernel and coset representatives. The accessible algorithms are: "MinDistanceExtendCoset" (Algorithm 6) and "MinDistanceCoset" (Algorithm 8). The "MinDistanceCoset" algorithm is used by default, and the "MinDistanceExtendCoset" will be used if the parameter `AlgMethod` is assigned the value "MinDistanceExtendCoset".

Note that the “MinDistanceExtendCoset” algorithm can be faster than the “MinDistanceCoset” algorithm, since it uses functions already integrated into the MAGMA system.

Example 6.4.1.

```
> V := VectorSpace(GF(2),31);
> C_kernel := SimplexCode(5);
> C_leaders := [
    V![ 0,0,1,0,0,0,1,1,1,0,0,1,1,0,1,0,0,1,1,1,1,0,0,0,1,0,1,1,1,1,0],
    V![ 0,1,0,1,1,0,1,0,1,0,1,1,1,1,0,0,1,0,1,1,1,0,1,0,0,1,1,1,1,0,1],
    V![ 0,0,0,0,0,1,1,1,0,0,0,1,1,1,0,1,1,0,1,0,0,0,1,1,1,1,0,1,0,1,1]
];
> C := BinaryCode(C_kernel, C_leaders);
> w := BinaryMinimumWeight(C);
> w;
10;
> d := BinaryMinimumDistance(C);
> d;
8
```

6.5 Decoding binary codes

Using the algorithms described in Section 5.1 and Section 5.2, we have implemented new functions to simulate the decoding process for binary codes.

`BinaryCosetDecode(`C`, `d`, `u` : parameter)`

`AlgMethod`	MONSTGELT	*Default* : “MinWeightCoset”
`MinWeigthKernel`	RNGINTELT	*Default* : -

Given a binary code C, its minimum distance d and a vector u from the ambient space V of C, attempt to decode u with respect to C. The accessible algorithms are: “MinWeightExtendCoset” (Algorithm 5) and “MinWeight-Coset” (Algorithm 7). The “MinWeightCoset” algorithm is used by default, and the “MinWeightExtendCoset" will be used if the parameter `AlgMethod` is assigned the value `"MinWeightExtendCoset"`. If the decoding algorithm succeeds in computing a vector u' as the decoded version of u, then the function returns `true` and u'. If the decoding algorithm does not succeed in decoding u, then the function returns `false` and the zero vector.

The "MinWeightExtendCoset" algorithm consists of consider the linear code $C_u = C \cup (C+u)$ when C is linear, or the linear codes $K_0 = K \cup (K+u)$, $K_1 = K \cup (K+v_1+u)$, …, $K_t = K \cup (K+v_t+u)$, where K is the kernel of C and $C = \bigcup_{i=0}^{t}(K+v_i)$, when C is nonlinear. If C is linear and the minimum weight of C_u is less than d, then $u' = u + e$, where e is a word of minimum weight of C_u; otherwise, the decoding algorithm returns `false`. On the other hand, if C is nonlinear and the minimum weight of $\cup_{i=0}^{t} K_i$ is less than the minimum weight of K, then $u' = u + e$, where e is a word of minimum weight of $\cup_{i=0}^{t} K_i$; otherwise, the decoding algorithm returns `false`. If the parameter `MinWeightKernel` is not assigned, then the minimum weight of K is computed.

The "MinWeightCoset" algorithm consists of just consider the coset $C+u$ when C is linear, or the cosets $K+u$, $K+v_1+u$, …, $K+v_t+u$, where K is the kernel of C and $C = \bigcup_{i=0}^{t}(K+v_i)$, when C is nonlinear. In this case, the decoding algorithm always will return `true` and $u' = u + e$, where e is a word of minimum weight of $C+u$ or $\cup_{i=0}^{t}(K+v_i+u)$ depending on whether C is linear or not, respectively.

Note that the "MinWeightExtendCoset" algorithm can be faster than the "MinWeightCoset" algorithm, since it uses functions already integrated into the MAGMA system. However, it may correct less errors than the default method.

Propositions 5.1.1 and 5.2.1, and Algorithms 9 and 10 for binary codes, are used in the implementation.

Example 6.5.1. *We create a binary nonlinear code C and a vector c of C and then perturb c to a new vector u. We then decode u to find c again.*

```
> V := VectorSpace(GF(2),31);
> C_kernel := SimplexCode(5);
> C_leaders := [
    V![ 0,0,1,0,0,0,1,1,1,0,0,1,1,0,1,0,0,1,1,1,1,0,0,0,1,0,1,1,1,1,0],
    V![ 0,1,0,1,1,0,1,0,1,0,1,1,1,1,0,0,1,0,1,1,1,0,1,0,0,1,1,1,1,0,1],
    V![ 0,0,0,0,0,1,1,1,0,0,0,1,1,1,0,1,1,0,1,0,0,0,1,1,1,1,0,1,0,1,1]
];
> C := BinaryCode(C_kernel, C_leaders);
> d := BinaryMinimumDistance(C);
> d;
8
```

```
> c := V ! [0,1,1,1,0,1,1,0,0,1,0,0,0,1,0,0,0,1,1,0,0,1,1,1,0,0,1,1,1,0,0];
> IsInBinaryCode(C, c);
true
> u := c;
> u[5] := u[5] + 1;
> u[12] := u[12] + 1;
> c;
(0 1 1 1 0 1 1 0 0 1 0 0 0 1 0 0 0 1 1 0 0 1 1 1 0 0 1 1 1 0 0)
> u;
(0 1 1 1 1 1 1 0 0 1 0 1 0 1 0 0 0 1 1 0 0 1 1 1 0 0 1 1 1 0 0)
> c-u;
(0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)

> isDecoded, cDecoded := BinaryCosetDecode(C, d, u);
> isDecoded;
true
> cDecoded;
(0 1 1 1 0 1 1 0 0 1 0 0 0 1 0 0 0 1 1 0 0 1 1 1 0 0 1 1 1 0 0)
> cDecoded eq c;
true
```

6.6 Package for q-ary nonlinear codes

Using the algorithms described in Section 5.3 and Section 5.4, we have implemented new functions to simulate the decoding process for q-ary codes.

`QaryDecode(`C`, `u`)`

Decode the received vector u respect to the q-ary code C using extended coset decoding. Decoding linear q-ary codes and decoding nonlinear q-ary codes are united in one function. During the decoding process, the minimum weight of some cosets are computed, currently for q-ary codes, only MinWeightExtendCoset are implemented. Algorithms 9 and 10 are used in the implementation.

For q-ary nonlinear codes, only some functions have been implemented. For example, the improved versions of algorithms IMinW, IMinD, CosetDecode for q-ary codes have not been implemented yet. Next package versions will include these improvements and new results.

Chapter 7

Conclusions

Coding theory was introduced with the publication of the paper "A mathematical theory of communication" by C. Shannon in 1948. In the last 60 years, this theory has grown as a mathematical discipline with applications to sciences and engineering. Perhaps the most eminent of Shannon's results was the concept that every communication channel had a speed limit, measured in binary digits per second. Mathematically, this means that it is impossible to obtain an error free communication above this limit but, below the Shannon limit, it is possible to transmit information without errors. In order to do this, we need to encode the information, by introducing redundancy into the digital representation. Coding theory deals with the design of good error-correcting codes for the reliable transmission of information across noisy channels, according to Shannon's laws.

Most of the codes used in coding theory are linear. However, it is known that some nonlinear codes have more codewords than any linear one with the same parameters, length and minimum distance, so they are better than any linear code. Using a nonlinear code is not as easy as using a linear one. The lack of linearity implies that it is not possible to represent them in a compact way by using a generator o parity-check matrix, and the process of encoding and decoding can not be performed in an efficient way. The main goal of this dissertation is to discuss a general representation of nonlinear codes, the kernel/coset representation, suitable to represent a nonlinear code without storing all its codewords. We have seen how to construct nonlinear codes from other ones using this representation, how to compute their minimum weight and minimum distance in a more efficient way than by brute force, and how to apply the minimum distance computation to describe a general decoding method for nonlinear codes.

7.1 Main results

In this dissertation, an algorithm to obtain the kernel/coset representation of a nonlinear code is described and improved. The complexity of computing the kernel and coset representatives of a nonlinear code is determined by the dimension of its kernel, or say, how far it is from being linear.

Although for codes with large size, the computation of the kernel and coset representatives is not efficient enough, some known constructions allow us to construct new nonlinear codes from old ones by manipulating directly their kernel and coset representatives. More specifically, properties such as equality, inclusion, intersection and union between nonlinear codes are given in terms of this representation. Also, some well known code constructions (extended code, punctured code, shortened code, direct sum, Plotkin sum) are described by manipulating directly the kernel and coset representatives of the constituent nonlinear codes.

When the kernel/coset representation of a nonlinear code is already computed, the nonlinear code can be stored more efficiently. Moreover, some new algorithms designed to compute the minimum weight and minimum distance of a nonlinear code represented using this structure can be applied. These algorithms are based on computing the minimum weight of a union of some linear codes which include all codewords of the nonlinear code. Furthermore, these algorithms are improved by modifying the enumeration process of Brouwer-Zimmermann's algorithm in such a way that unnecessary vectors, which are not codewords of the nonlinear code, are excluded.

As an application of the algorithm to compute the minimum weight of a nonlinear code, a new algorithm to decode linear codes is given and compared with the traditional syndrome decoding. Instead of computing a syndrome table, the new decoding method can decode a received vector by finding the minimum word of a coset of the linear code. This decoding method can also be used for nonlinear codes, and it is the only decoding general method for nonlinear code as far as we know.

Some of the algorithms given in the dissertation are evaluated by real test and theoretical results. All the test are performed in the MAGMA system and the algorithms are written into standard MAGMA functions, which will be released within a new package to work with nonlinear codes. This new package will be useful to support further research on this area.

7.2 Further research and development

By using the kernel/coset representation, we generalized some well-known results and algorithms suitable for linear codes into nonlinear codes over $\mathbb{F}_q$. Further research on this topic can be the following:

- To design new efficient algorithms to compute the weight distribution for q-ary nonlinear codes, considering the kernel/coset representation.
- To design new efficient algorithms to compute the covering radius for q-ary nonlinear codes, considering the kernel/coset representation.
- To adapt the probabilistic methods to compute the minimum distance for linear codes into the computation for q-ary nonlinear codes.
- To study the application of the general algorithms for computing the minimum distance to some families of nonlinear codes, such as $\mathbb{Z}_4$-linear codes, or $\mathbb{Z}_2\mathbb{Z}_4$-linear codes; and to establish which method is more suitable depending maybe on the parameters of the code.
- To find new optimal nonlinear codes for some given parameters, by using the new MAGMA package on binary nonlinear codes.

Based in the results given in this dissertation, and in order to evaluate most of the algorithms described, we have developed a new MAGMA package to deal with binary nonlinear codes. The last version of the package can be downloaded from the CCSG web site `http://ccsg.uab.cat` together with a manual describing all functions. For q-ary codes, so far, we have only implemented some functions. Further development on this topic can be the following:

- To extend the functionality of the package for binary nonlinear codes, according to new algorithms developed for computing other parameters, such as the weight distribution and covering radius.
- To improve some of the functions already implemented in the package in order to be more efficient.
- To generalize the implementation of the package to deal with binary nonlinear codes, in order to deal with any q-ary nonlinear code.
- To generate a database of nonlinear perfect codes storing the codes using the kernel/coset representation.

- To generate a database of optimal nonlinear codes storing the codes using the kernel/coset representation.

Bibliography

[1] H. Bauer, B. Ganter and F. Hergert, "Algebraic techniques for non-linear codes", *Combinatorica*, vol. 3, no. 1, pp. 21-33, 1983.

[2] A. Betten, H. Fripertinger, A. Kerber, A. Wassermann, and K.-H. Zimmermann, "*Codierungstheorie: Konstruktion und Anwendung linearer Codes*", Berlin: Springer, 1998, ISBN: 9783540645020.

[3] G. Birkhoff and S. Mac Lane, "*A survey of modern algebra*", New York:Macmillan Publishing Co., Inc., 1977, ISBN: 0023100702.

[4] J. Borges and C. Fernández, "Plotkin construction: rank and kernel", [arXiv:0707.3878v1], July 26, 2007.

[5] J. Borges, C. Fernández, J. Pujol, J. Rifà, and M. Villanueva, "$\mathbb{Z}_2\mathbb{Z}_4$-linear codes. A MAGMA package," Universitat Autònoma de Barcelona, 2007. `http://www.ccsg.uab.cat`

[6] J. Borges, C. Fernández, J. Pujol, J. Rifà and M. Villanueva, "$\mathbb{Z}_2\mathbb{Z}_4$-linear codes: generator matrices and duality", *Designs, Codes and Cryptography*, vol. 54, no. 2, pp. 167-179, 2010.

[7] M. Borges-Quintana, M.A. Borges-Trenard, I. Márquez-Corbella, and E. Martínez-Moro, "Computing coset leaders and leader codewords of binary codes", [arXiv:1211.5568v1], November 26, 2012.

[8] R. Bose, *Information theory, coding and cryptography*, 2nd ed, Tata McGraw-Hill Education, 2008, ISBN: 0070669015.

[9] W. Bosma, J.J. Cannon, C. Fieker and A. Steel (eds.), *Handbook of* MAGMA *functions*, Edition 2.16, 5017 pages, 2010. `http://magma.maths.usyd.edu.au/magma/`

[10] I. Bouyukliev and V. Bakoev, "A method for efficiently computing the number of codewords of fixed weights in linear codes", *Discrete Applied Mathematics*, vol. 156, no. 15, pp. 2986-3004, August 2008.

[11] A. Brouwer, Table of general binary codes, `http://www.win.tue.nl/~aeb/codes/binary-1.html`

[12] L. Childs, *A concrete introduction to higher algebra*, 3rd ed. New York: Springer, 2009, ISBN 9780387745275.

[13] A. Foster, "A polynomial-time probabilistic algorithm for the minimum distance of an arbitrary linear error-correcting code", Mathematics Honors Report, Spring 2003-2004.

[14] M. Grassl, "Searching for linear codes with large minimum distance", in *Discovering Mathematics with Magma*, W. Bosma and J. Cannon (Eds.). *Algorithms and Computation in Mathematics*, volume 19, Berlin: Springer, pp. 287-313, 2006.

[15] E. Guerrini, E. Orsini and M. Sala, "Computing the distance distribution of systematic non-linear codes", [arXiv: 0909.1626], 9 September 2009.

[16] A.R. Hammons, P.V. Kumar, A.R. Calderbank, N.J.A. Sloane and P. Solé, "The $\mathbb{Z}_4$-linearity of Kerdock, Preparata, Goethals and related codes", *IEEE Trans. Inf. Theory*, vol. 40, no. 2, pp. 301-319, 1994.

[17] O. Heden, "On perfect p-ary codes of length $p+1$", *Designs, Codes and Cryptography*, vol. 46, no 1, pp. 45-56, 2008.

[18] D.G. Hoffman, D.A. Leonard, C.C. Lindner, K.T. Phelps, C.A. Rodger, J.R. Wall, *Coding theory: the essentials*, New York: Marcel Dekker, Inc. 1991, ISBN: 0824786114.

[19] W.C. Huffman and V. Pless, *Fundamentals of error-correcting codes*, New York: Cambridge University Press. 2003, ISBN: 1139439502.

[20] A. Al Jabri, "A statistical decoding algorithm for general linear block codes", in *Cryptography and Coding*, B. Honary (Ed.). LNCS 2260, pp. 1-8. Berlin: Springer, 2001.

[21] A.M. Kerdock, "A class of low-rate nonlinear binary codes", *Information and Control*, vol. 20, no. 2, pp. 182-187, March 1972.

[22] D.E. Knuth, *The art of computer programming, volume 4A: Combinatorial algorithms Part 1*, New Jersey: Addison-Wesley, 2011, ISBN: 0201038048.

[23] J.S. Leon "A probabilistic algorithm for computing minimum weights of large error-correcting codes", *IEEE trans. on information theory*, vol. 34, no. 5, pp. 1354-1359, September 1988.

[24] F.J. MacWilliams and N.J.A. Sloane, *The theory of error-correcting codes*, Amsterdam: North-Holland Publishing Company. 1977, ISBN: 0444851933.

[25] M. Mohri and M. Morii, "On computing the minimum distance of linear codes", *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)*, vol. 83, no. 11, pp. 32-42, November 2000.

[26] P.R.J. Östergård and O. Pottonen, "The perfect binary one-error-correcting codes of length 15: part I-classification", *IEEE Trans. Inf. Theory*, vol. 55, no. 10, pp. 4657-4660, October 2009.

[27] J. Pernas, J. Pujol, and M. Villanueva, "Codes Over $\mathbb{Z}_4$. A MAGMA package", Universitat Autònoma de Barcelona, 2011. `http://www.ccsg.uab.cat`

[28] K.T. Phelps, J. Rifà, and M. Villanueva, "Kernels and p-kernels of p^r-ary 1-perfect codes", *Designs, Codes and Cryptography*, vol. 37, no. 2, pp 243-261, November 2005.

[29] F.P. Preparata, "A class of optimum nonlinear double-error-correcting codes", *Information and Control*, vol. 13, no. 4, pp. 378-400, October 1968.

[30] J. Pujol, J. Rifà, and F.I. Solov'eva, "Quaternary Plotkin constructions and quaternary Reed-Muller codes", *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, AAECC-17, vol. 4851, pp 148-157, December 2007.

[31] J. Pujol, M. Villanueva and F. Zeng, "Minimum distance of binary nonlinear codes", in *18th International Conference on Applications of Computer Algebra (ACA 2012)*, Sofia, Bulgaria, pp. 25-28, June 2012.

[32] J. Pujol, M. Villanueva and F. Zeng, "On the minimum distance of q-ary nonlinear codes", in *Canadian Discrete and Algorithmic Mathematics Conference (CanaDAM 2013)*, St. John's, Canada, pp. 10-13, June 2013.

[33] J. Pujol, M. Villanueva and F. Zeng, "Representation, constructions and minimum distance computation of binary nonlinear codes", in *19th International Conference on Applications of Computer Algebra (ACA 2013)*, Málaga, Spain, pp. 142-143, July 2013.

[34] The Combinatorics, Coding and Security Group. Universitat Autònoma de Barcelona. `http://ccg.uab.cat`

[35] M. Villanueva, F. Zeng, J. Pujol, "Nonlinear q-ary codes: constructions and minimum distance computation", in *XIV Encuentro de Álgebra Computacional y Aplicaciones (EACA 2014)*, Barcelona, Spain, June 18-20, 2014.

[36] M. Villanueva, F. Zeng, J. Pujol, "Efficient representation of binary nonlinear codes: constructions and minimum distance computation", submitted to *Designs, Codes and Cryptography*, 2014.

[37] A. Vardy, "The intractability of computing the minimum distance of a code", *IEEE Trans. Inf. Theory*, vol. 43, no. 6, pp. 1757-1773, November 1997.

[38] Z.-X. Wan, *Quaternary codes*, Singapore: World Scientific Pub. Co. Inc., December 1997, ISBN: 9810232748.

[39] G. White, "Enumeration-based algorithms in coding theory", PhD Thesis, University of Sydney, 2006.

[40] G. White and M. Grassl, "A new minimum weight algorithm for additive codes", *ISIT 2006*, pp. 1119-1123, Seattle, USA, July 9-14, 2006.

[41] K.-H. Zimmermann, "Integral Hecke modules, integral generalized Reed-Muller codes, and linear codes", in *Berichte des Forschungsschwerpunktes Informations und Kommunikationstechnik*, vol. 96, Techn. Univ. Hamburg-Harburg, 1996.

Fanxuan Zeng
Cerdanyola del Vallès, June 2014