Ampliació d'una llibreria en MAGMA: Propietats i distància mínima de codis q-aris

Erik Vicent Ferrer

Resum— Aquest projecte té com a objectiu principal contribuir en el desenvolupament d'una llibreria en el sistema Magma per treballar amb codis *q-aris* no lineals de forma eficient. El que es pretén és analitzar i millorar l'eficiència de funcions ja implementades dins el propi paquet, i implementar-ne de noves eficientment. D'aquesta manera l'objectiu principal es divideix en 5 parts. La primera es basa en l'estudi de les propietats bàsiques dels codis lineals i no lineals a nivell teòric. La segona en l'aprenentatge del sistema Magma amb la seva llibreria per a codis binaris no lineals ja implementada. La tercera part d'aquest objectiu principal consisteix en la validació i l'estudi del rendiment de funcions que construeixen codis *q-aris,* ja siguin lineals o no lineals, comparant diferents algorismes de construcció utilitzant la dimensió del *kernel*, els representants dels *cosets* d'un codi i el *kernel* parcial d'aquest, establint un algorisme general segons els resultats extrets. La quarta part es basa en la implementació de diverses funcions del paquet de codis *q-aris* que més endavant es presenten. Finalment, la cinquena part consisteix en la implementació de les funcions del càlcul del pes mínim i la distància mínima dels codis per a la llibreria de forma eficient tenint present novament la comparació de diferents mètodes a l'hora de treure'n un resultat concret. Tot aquest treball estarà correctament validat a partir de tests prèviament proposats.

Paraules clau—Distància mínima, pes mínim, codis *q-aris*, codis lineals, codis no lineals, sistema Magma.

Abstract—This project aims to contribute to the main development of the Magma system, specifically the library for nonlinear *q-ary* codes efficiently. The aim is to analyze and improve the efficiency of functions already implemented within the packet and deploy new features optimally. Thus the main objective is divided into five parts. The first one is based on the study of the properties of linear and nonlinear codes in a theoretical level. The second part consists in learning how Magma works with the binary codes packet already implemented. The third part of this objective consists in the validation and studying of the performance of functions that build *q-ary* codes, whether they are linear or nonlinear, and compare different construction methods using the dimension of the *kernel*, the *coset representatives* and the *partial kernel*, to define a general algorithm according those performance tests. The fourth part consists in the implementation of various functions for the *q-ary* codes packet. Finally, the last part consists in the implementation of new functions capable to calculate the minimum weight and the minimum distance of a *q-ary* code with the help of performance tests and establishing the most efficient way to treat the *q-ary* codes. All this work will be successfully validated from tests previously proposed.

Index Terms—Minimum distance, minimum weight, *q-ary* codes, linear codes, nonlinear codes, Magma system.

1 Introducció

QUAN un emissor i un receptor estableixen un canal de comunicació per a realitzar una transmissió d'informació digital la presència de soroll és possible. L'objectiu de la teoria de codis és l'estudi del disseny de codis correctors per a una transmissió fiable de la informació a través d'un canal amb soroll. Aquest canal pot ser binari, ternari o, tal i com es tractarà en aquesta memòria, *q-ari*, segons el nombre d'elements diferents que poden passar pel canal. Per tant, aquest valor q també defineix el nombre d'elements que tindrà l'alfabet sobre el que construirem el codi. A causa del soroll, existeix la necessitat de transformar qualsevol missatge a enviar en una seqüència de símbols de l'alfabet *q-ari* acceptat pel canal i afegir redundància, a través del ja mencionat codi amb capacitat per corregir tots aquells errors generats pel soroll.

Sigui F_q^n el conjunt de vectors de longitud n amb coeficients en un cos finit F_q de q elements. La distància (de Hamming) entre dos vectors u, v de longitud n és el nombre de posicions en que els dos vectors són diferents. El pes (de Hamming) d'un vector u és el nombre de posicions diferent de zero d'u. Un codi q-ari $C(n, M, d)$ és un subconjunt de F_q^n amb M elements anomenats paraules-codi, on d és la distància mínima, que es defineix com la menor de les distàncies entre dues paraules-codi diferents i es pot expressar amb la fórmula:

$$d = mín\{d(u,v) | u, v \in C, u \neq v\},$$

on $d(u, v)$ és la distància entre u i v. El pes mínim d'un codi és el menor dels pesos de les paraules-codi diferents de zero.

La capacitat correctora, t, d'un codi és el nombre de símbols de l'alfabet *q-ari* erronis que es poden corregir per a cada paraula-codi enviada en una transmissió. Es calcula mitjançant la fórmula:

$$t = \lfloor (d-1)/2 \rfloor.$$

- *E-mail de contacte: Erik.Vicent.94@gmail.com*
- *Menció realitzada: Tecnologies de la Informació*
- *Treball tutoritzat per: Mercè Villanueva i Jaume Pujol (Departament d'Enginyeria de la Informació i de les Comunicacions)*
- *Curs 2016/17*

Per tant, el càlcul de la distància mínima d és necessari per conèixer la capacitat correctora t d'un codi *q-ari* [10]. La tècnica d'afegir elements redundants és en la que es basa el procés de correcció d'errors al expressar una seqüencia d'elements sobre un alfabet de manera que qualsevol error que sigui introduït, pugui ser detectat i corregit. És un procés que inclou la codificació, transmissió i descodificació de la seqüencia d'elements. Per a dur a terme aquest procés es fan servir codis blocs, alguns dels quals tenen propietats de l'àlgebra lineal. Concretament, seran tractats dos tipus de codis correctors en aquesta memòria: els lineals i els no lineals.

Els codis lineals compleixen propietats de l'àlgebra lineal de manera que faciliten en major grau el procés de codificació i descodificació. Un codi *q-ari* $C(n, M, d)$, on $M = q^k$, serà lineal si i només si C és un subespai de dimensió k de F_q^n. En aquest cas també ens podrem referir a C com un codi $C[n, k, d]$. Com a tals són codis que es poden representar a partir d'una matriu generadora, que és qualsevol matriu G de mida $k\ x\ n$ on les seves files formen una base del subespai vectorial determinat per C [10].

Per altra banda, tenim els codis no lineals, els quals ens centrarem més en aquesta memòria, i que malgrat no tenir, en general, les mateixes bones propietats per codificar i descodificar com els lineals, els no lineals tenen com a punt a favor el fet que alguns dels millors codis resulten ser no lineals [8, pàg. 53], [9], [10]. El motiu pel qual, en part, no són tan bons com els lineals és perquè la seva representació no es tant compacta. El millor mètode per representar un codi no lineal de forma reduïda es basaria en la representació *kernel/cosets*, on el *kernel* (en anglès, o nucli en català) es defineix com el conjunt de tots els vectors que deixen el codi invariant al fer translacions. Concretament, el *kernel* d'un codi C es calcula com a $K(C) = \{x \in F_q^n \mid \lambda x + C = C, \forall \lambda \in F_q\}$ [11]. El *kernel* és sempre un subespai vectorial de F_q^n. A més, a partir del *kernel* podem dividir el codi en *cosets* o traslladats de manera que podem representar el codi amb una base del *kernel* i un element de cada *coset*, $c_0, c_1, \dots, c_t$, anomenats *representants* dels *cosets*. O sigui, un codi C es pot expressar com la unió de *cosets* del *kernel* $K(C)$ de la següent manera:

$$C = \bigcup_{i=0}^{t} \Big(K(C) + c_i \Big) \quad (1)$$

Un cop vistos els codis no lineals i tot i que la seva codificació i descodificació pot portar més dificultats que en els lineals, la seva taxa de transmissió és millor [8, pàg. 53], [9], [10]. Aquest motiu, juntament amb el poc ús donat a aquests codis en sistemes com Maple [17], Sage [18], GAP [15] o Magma [16], va propiciar que des del CCSG (*Combinatorics, Coding and Security Group*) del dEIC (Departament d'Enginyeria de la Informació i de les Comunicacions) es decidís crear un paquet per treballar amb codis *q-aris* no lineals estenent les funcionalitats existents per a codis lineals [13]. Es va iniciar la creació d'aquest paquet en el sistema Magma, ja que és un dels millors softwares dissenyats per resoldre problemes computacionals difícils en àlgebra i sobretot en teoria de codis, a més del fet que des del CCSG ja s'havien realitzat altres paquets amb Magma [1], [2], [12], [20].

1.1 Estat de l'art

Aquest treball s'ha basat en l'estudi de codis no lineals *q-aris* així com l'obtenció de mètodes més eficients de construcció d'aquests a partir de diversos algorismes. Per assolir aquesta fita aquest treball s'ha recolzat en la premissa de que alguns dels millors codis existents són no lineals, tal i com s'explica en els llibres *Fundamentals of error-correcting codes* [8, pàg 53], *New upper bounds on binary linear codes and a Z4-code with a better-than-linear Gray image* [9], i també en el llibre de F. J. MacWilliams: *The theory of error-correcting codes* [10].

Des del CCSG ja fa uns anys que es va proposar la realització d'una sèrie de paquets per al sistema Magma de cara a l'ampliació de les seves llibreries per afegir les funcionalitats relacionades amb l'ús d'aquests codis no lineals. Dins els paquets ja desenvolupats, hi ha el de codis binaris [12], el qual es centra en tots aquells algorismes així com les seves funcionalitats lligades únicament amb els codis binaris. Més endavant, i relacionat amb aquest treball, es va iniciar el desenvolupament d'un altre paquet, el de codis *q-aris* [13], el qual està en procés i en el qual aquest projecte pretén fer-ne la seva aportació de cara a l'expansió del paquet i per tant, de la llibreria del Magma.

1.2 Objectius del treball

L'objectiu principal d'aquest treball és contribuir en el desenvolupament d'una llibreria en Magma per treballar amb codis *q-aris* no lineals de forma eficient [13]. Concretament es pretén analitzar i millorar l'eficiència de funcions ja implementades dins el propi paquet, i implementar noves funcions de forma eficient. D'aquesta manera l'objectiu principal es dividirà en 5 apartats:

1. Estudiar les propietats bàsiques dels codis lineals i no lineals: mida, pes mínim, distància mínima, *kernel*, *rang*, representació amb *cosets* del *kernel.*
2. Conèixer el llenguatge de programació del sistema Magma i la llibreria de codis binaris no lineals ja implementada.
3. Validar i estudiar el rendiment de funcions generals per a la construcció de codis *q-aris* utilitzant el *kernel* i distingint entre el cas binari i no binari.
4. Implementar noves funcions del paquet: els invariants d'un codi *q-ari*, el *rang* i el *kernel*, l'espai dual, i la generació de codis aleatoris.
5. Implementar, validar i estudiar el rendiment de funcions destinades al càlcul del pes mínim i la distància mínima.

En primer lloc s'han estudiat les propietats bàsiques dels codis lineals i no lineals tals com la mida, el pes mínim, la distància mínima, el *kernel*, el *rang*, i la representació amb traslladats del *kernel* per tal de poder entendre la temàtica del treball des d'un punt de vista teòric.

Paral·lelament i d'acord amb el segon apartat, hi ha

hagut un procés d'adaptació per entendre el funcionament del sistema Magma amb totes les seves particularitats [3]. A més, hi ha hagut un treball de documentació per entendre el funcionament del paquet de codis binaris i algunes funcions ja implementades del paquet de codis *q-aris* necessàries pel desenvolupament d'aquest projecte.

D'acord amb el tercer punt, s'ha validat i estudiat el rendiment d'aquelles funcions del paquet de codis *q-aris* encarregades de la construcció de codis utilitzant la representació *kernel/cosets*. A partir dels resultats de l'estudi de rendiment s'ha determinat un algoritme eficient per a la construcció de codis *q-aris* tenint en compte un mostreig de codis que es presentaran en la Taula 1 de l'Apartat 3, juntament amb el desenvolupament d'aquest punt.

D'acord amb el quart punt, s'ha ampliat la funcionalitat del paquet. Concretament, s'han implementat diverses funcions corresponents a codis trivials *q-aris*, càlcul del *rang*, el *kernel*, l'espai dual, la generació de codis aleatoris i també funcions corresponents a l'apartat d'invariants. Totes aquestes funcions seran degudament presentades en l'Apartat 4 d'aquest document.

Per últim, amb el cinquè punt dels objectius d'aquest treball, similar al punt 3, s'ha tractat la implementació de les funcions destinades al càlcul del pes mínim i la distància mínima per, posteriorment, validar-les i estudiar-ne el seu rendiment. S'han analitzat dos algoritmes diferents capaços de calcular aquests dos paràmetres de la forma més ràpida possible.

Aquest document es divideix en els següents apartats: primer de tot es parla sobre la metodologia emprada; en segon lloc s'explica com s'ha dut a terme el desenvolupament dels punts 1, 2 i 3; a continuació en l'Apartat 4 s'inclou la descripció de les noves funcions implementades; després es presenta el desenvolupament i validació del punt 5 amb tota la feina realitzada en l'aspecte d'anàlisi de resultats; i finalment es mostren les conclusions extretes a partir de la realització d'aquest treball, els agraïments, les referències bibliogràfiques utilitzades i els apèndixs.

2 Metodología

Per encarar aquest projecte s'han emprat dos tipus de metodologies diferents, per un costat hi ha hagut un enfocament més teòric i per l'altre un component més pràctic.

En aquest projecte s'han tractat conceptes sobre la teoria de codis i concretament sobre els codis lineals i no lineals que han requerit un estudi teòric previ. Per això primerament s'ha enfocat el treball a un nivell més conceptual per tractar d'entendre tots aquells conceptes matemàtics que hi apareixen. El motiu principal que motiva a utilitzar aquesta metodologia inicial no és altre que seguir els principis de la metodologia matemàtica on, en primer lloc, és clau tenir clars els conceptes relacionats amb l'estudi que un vagi a realitzar. Per tant, cal documentar-se bé previ a encarar la part pràctica del projecte.

El principal objectiu és el d'incrementar la funcionalitat del paquet de codis *q-aris* no lineals de Magma, per tant la part experimental s'ha basat en la implementació de forma eficient de diferents algorismes així com la corresponent interpretació dels resultats obtinguts.

Aquest desenvolupament ha seguit com a metodologia experimental el *Test Driven Development* (TDD) [14], [19]. Aquesta metodologia consisteix en codificar les funcions de manera que primer s'ha tingut en compte pensar casos importants que validin el sistema per després codificar les solucions basades en aquests "casos d'ús". És per això que ha estat convenient utilitzar-la en la realització d'aquest treball degut a que, a partir de l'estudi teòric previ, es va decidir quins eren els casos més importants a tenir en compte i després es van generar les funcions per arribar a la solució pensada, tot per finalment poder avaluar els resultats obtinguts.

L'entorn de treball ha estat diversificat. Una gran part de la feina s'ha realitzat en un dels despatxos del departament cedit per a la recerca d'aquest projecte. Més endavant i gràcies a dues eines com són els programes PuTTY i WinSCP, l'entorn de treball ha pogut ser des de casa de manera que ha sigut possible realitzar sessions de treball connectant remotament l'ordinador personal amb els servidors disponibles del departament. Aquests servidors disposen de totes les eines necessàries per desenvolupar i codificar cada funció amb Magma. A més, ha sigut de vital importància l'ajuda, consells i dubtes resolts per part de la Mercè Villanueva i d'en Jaume Pujol, amb els que hi han hagut reunions setmanals per avaluar el progrés realitzat. Finalment, de cara a l'adquisició de coneixements relacionats amb el llenguatge Magma també s'ha requerit l'ús del manual general del Magma [3].

3 Desenvolupament i anàlisis de les funcions de construcció de codis

Al començar aquest projecte, el primer que es va fer va ser un treball de cerca d'informació per poder estudiar les propietats bàsiques dels codis lineals i no lineals, i també conèixer el llenguatge de programació Magma i la llibreria amb la que es treballaria. Les primeres tasques es van basar en la consulta de llibres sobre teoria de codis [8], tutories d'introducció al temari de la mà de la Mercè Villanueva i en Jaume Pujol, consultar tesis i treballs anteriors relacionats amb el tema [6], [7], [20], mirar tutorials i documentació de Magma [4], [5], [16], i finalment, consultar la documentació sobre la llibreria per a codis no lineals binaris i *q-aris* del grup de recerca [12], [13].

A continuació, en l'Apartat 3.1, s'introdueix la representació de codis *q-aris* anomenada *kernel/cosets* i s'expliquen les diferents funcions disponibles per a la construcció de codis *q-aris,* així com els codis generats per a la seva validació. En l'Apartat 3.2, es descriu l'estudi de rendiment realitzat per a una d'aquestes funcions i finalment en l'Apartat 3.3 s'inclouen les observacions de l'estudi i es presenta l'algoritme final.

3.1 Introducció al paquet de codis *q-aris*

La finalitat d'aquest apartat és explicar la representació dels codis *q-aris* que es fa servir en aquest paquet, fent servir la metodologia *kernel/cosets* així com les diferents funcions disponibles per a la construcció de codis *q-aris*.

Tal i com s'ha mencionat en la introducció, el Magma és un dels millors softwares dissenyats per resoldre problemes computacionals difícils en teoria de codis. Aquest va ser un dels motius per escollir-lo per al desenvolupament del paquet de codis binaris [12] i *q-aris* [13] des del grup CCSG. Amb aquests paquets el que es pretén és expandir el sistema Magma amb funcions per a codis no lineals. Dins d'aquest projecte es vol contribuir en l'expansió del paquet de codis *q-aris* implementant noves funcions i millorant el rendiment d'algunes ja implementades.

En el primer apartat del manual del paquet de codis *q-aris* apareixen funcions per a la construcció de codis. Concretament, hi ha disponibles les tres funcions següents: *QaryCode(L), QaryCode(L, K) i QaryCode(K, Rep)* [13], [20]. Els seus pseudocodis apareixen a l'Apèndix A1. Aquestes tres funcions tenen com a objectiu comú construir un codi *q-ari* representat utilitzant el *kernel* i els *cosets* corresponents, per tal de que ocupi menys espai. Segons la informació disponible del codi *q-ari* a emmagatzemar o representar, és més convenient utilitzar una funció o una altra. Les tres funcions aplicades a un mateix codi *q-ari* haurien de donar un resultat equivalent.

La funció *QaryCode(L)* admet com a paràmetre una seqüència *L* de totes les paraules del codi. Aquesta funció s'encarrega de calcular el *kernel* del codi i determinar uns representants dels *cosets*. Aquests representants són paraules-codi que per elles soles representen un conjunt de paraules-codi, és a dir, cada representant representa un *coset*, i que, combinades amb els vectors del *kernel*, donen lloc a totes les paraules-codi pertanyents al seu *coset*.

En el cas de la funció *QaryCode(L, K)* entren en joc dos paràmetres. Per un costat tindríem la seqüència *L* de totes les paraules del codi, i a més a més, li passarem un *kernel K* el qual pot ser parcial o final (l'usuari pot no saber-ho d'entrada), de manera que a priori hem de pensar que aquesta funció sempre serà més ràpida que *QaryCode(L)* ja que estem estalviant part dels càlculs a fer al proporcionar una part o directament el *kernel* final del codi.

Finalment tenim la funció *QaryCode(K, Rep)*. En aquesta funció seguim tenint l'objectiu de reduir el codi original però aquest cop representa que estem un pas endavant respecte les altres. En comptes de passar el codi sencer estem passant-li un codi més proper a codi final (que pot acabar sent el codi final o no, de nou, l'usuari no té perquè saber-ho), i la idea és que la funció intenti reduir-lo el màxim possible o directament donar-lo per bo. Aquesta funció pot potencialment ser la més ràpida de les tres.

Per validar aquestes tres funcions, s'han codificat una sèrie de tests que han comprovat, mitjançant 29 codis diferents, que el càlcul de construcció i reducció de codis es feia correctament. Aquests 29 codis s'han utilitzat al llarg del treball tant per validar funcions com per estudiar-ne el seu rendiment. A continuació es poden observar les propietats d'aquests 29 codis:

TAULA 1
LLISTAT DE CODIS *Q-ARIS* I LES SEVES PROPIETATS

(GF(q)=alfabet; n=mida; M=nº paraules-codi; d=distància mínima; k=dimensió kernel; Rep= nº representants cosets)							
Codi	**GF(q)**	**n**	**M**	**d**	**k**	**Rep**	**Linearitat**
CHamq2n15ker11	GF(2)	15	2048	3	11	0	Si
CHamq2n16ker1	GF(2)	16	2048	4	1	1023	No
CHamq2n16ker4	GF(2)	16	2048	4	4	127	No
CHamq2n16ker4t	GF(2)	16	2048	4	4	127	No
Chamq2n16ker9	GF(2)	16	2048	4	1	3	No
CKerq2n256ker9	GF(2)	256	65536	120	9	127	No
Cq2n10ker0	GF(2)	10	100	1	0	99	No
Cq2n20ker0	GF(2)	20	127	4	0	126	No
Cq2n20ker5	GF(2)	20	128	4	5	3	No
Cq3n13ker8a	GF(3)	13	59049	3	8	8	No
Cq3n13ker8b	GF(3)	13	59049	3	8	8	No
Cq3n13ker8c	GF(3)	13	59049	3	8	8	No
Cq3n13ker9a	GF(3)	13	59049	3	9	2	No
Cq3n13ker9b	GF(3)	13	59049	3	9	2	No
Cq3n13ker9c	GF(3)	13	59049	3	9	2	No
Cq3n13ker10	GF(3)	13	59049	3	10	0	Si
CHadq4n4ker2	GF(4)	4	16	3	2	0	Si
CHadq4n8ker1	GF(4)	8	32	6	1	7	No
CHadq4n12ker1	GF(4)	12	48	9	1	11	No
CHadq4n16ker1a	GF(4)	16	64	12	1	15	No
CHadq4n16ker1b	GF(4)	16	64	12	1	15	No
CHadq4n16ker1c	GF(4)	16	64	12	1	15	No
CHadq4n16ker2a	GF(4)	16	64	12	2	3	No
CHadq4n16ker2b	GF(4)	16	64	12	2	3	No
CHadq4n16ker3	GF(4)	16	64	12	3	0	Si
CHadq4n16ker3t	GF(4)	16	64	4	3	0	No
Cq4n4ker2	GF(4)	4	32	1	2	1	No
Cq4n16ker0	GF(4)	16	16	6	0	15	No
Cq8n20ker3	GF(8)	20	2560	12	3	4	No

3.2 Estudi de rendiment de *QaryCode(K, Rep)*

Un cop vistes les tres funcions anteriors, en aquest apartat l'objectiu és millorar l'eficiència de la funció *QaryCode(K, Rep)* proposant dos algoritmes diferents.

El primer algoritme presentat serà l'algoritme $\boldsymbol{\alpha}$. Aquest algoritme consistirà en la construcció d'un codi *L* mitjançant els paràmetres d'entrada *K* i *Rep* i un cop generat el codi, aquest serà passat a la funció *QaryCode(L, K)* per a que aquesta redueixi aquest codi *L*.

TAULA 2
PSEUDOCODI DE L'ALGORITME $\boldsymbol{\alpha}$

***Data**: A partial kernel* $\boldsymbol{K_P}$ *and partial coset representatives* $\boldsymbol{Rep_P}$ ***Result**: The kernel* $\boldsymbol{K}$ *and coset representatives* $\boldsymbol{Rep}$.
1. Generate the q-ary code L as a sorted sequence of codewords: $$\boldsymbol{L = \{k + c : k\ in\ K_P, c\ in\ Rep_P \cup \{0\}\}}$$ ***2. QaryCode(L,*** $\boldsymbol{K_P}$***)***

El segon algoritme a presentar és l'algoritme $\boldsymbol{\beta}$. En aquest algoritme la idea és, mitjançant la funció *QaryCode(K, Rep)*, directament donar per bona la sortida del codi ja reduït.

TAULA 3
PSEUDOCODI DE L'ALGORITME $\boldsymbol{\beta}$

***Data**: A partial kernel* $\boldsymbol{K_P}$ *and partial coset representatives* $\boldsymbol{Rep_P}$ ***Result**: The kernel* $\boldsymbol{K}$ *and coset representatives* $\boldsymbol{Rep}$.
1. QaryCode(K,Rep)

Els algoritmes $\boldsymbol{\alpha}$ i $\boldsymbol{\beta}$ donen codis equivalents (el *kernel* ha de ser el mateix però el conjunt de representants pot ser diferent). L'objectiu d'aquesta secció és determinar per a quines dimensions del *kernel* parcial és més eficient l'algorisme $\boldsymbol{\alpha}$ i per a quines dimensions ho és l'algorisme $\boldsymbol{\beta}$. Els resultats d'aquesta secció ens permeten determinar un valor de la dimensió del *kernel* parcial pel qual podrem triar l'algorisme $\boldsymbol{\alpha}$ o l'algorisme $\boldsymbol{\beta}$ per construir codis *q-aris*.

Per a realitzar les proves de rendiment s'han utilitzat 27 dels 29 codis presentats en la Taula 1 de l'apartat anterior. Tots aquests codis s'han tractat en base als dos algoritmes, executant 20 vegades cada codi amb cada algoritme i fent una mitjana dels temps per poder observar les diferències de rendiment en cada cas i poder-ne establir una aproximació d'aquell nombre de dimensió de *kernel* parcial amb la qual és preferible l'ús d'un algoritme o l'altre. La taula dels temps corresponent a aquest anàlisi de rendiment es podrà trobar en l'Apèndix A2.

3.3 Observacions de l'estudi i algoritme final

Un cop fetes totes les execucions i vists els resultats plasmats en la taula de l'Apèndix A2, les primeres observacions venen de la mà d'aquells temps marcats en *vermell*, ja que són els temps on es pot observar que hi ha un canvi de dinàmica; una transició en la que l'algorisme $\boldsymbol{\alpha}$ passa a ser millor que el $\boldsymbol{\beta}$.

En aquestes transicions podem observar que hi ha un patró força marcat on, deixant de banda que alguns codis triguen més en construir-se que d'altres degut a les seves característiques, es pot observar que la majoria de canvis de tendència significatius es produeixen quan el *kernel* té una dimensió igual a 1.

Entenem com a diferències de temps significatives aquelles en les que es superen els 2 segons de diferència entre el rendiment d'un algoritme i l'altre. Això clarament queda plasmat sobretot en els codis més grans, és a dir, els que tenen un major nombre de paraules-codi. Tenint present això prendríem com a número de referència de dimensió del *kernel* parcial el número 1, i serà per tant aquest el número de dimensió que determinarà canviar d'un algorisme a l'altre. En els casos on els codis són més petits i per tant la seva construcció en alguns casos és pràcticament immediata, les diferències en el rendiment es poden obviar. D'aquesta manera, es presenta el tercer i últim algoritme d'aquesta secció: l'algoritme $\boldsymbol{\Omega}$ descrit en la Taula 4.

TAULA 4
PSEUDOCODI DE L'ALGORITME $\boldsymbol{\Omega}$

***Data**: A partial kernel $\boldsymbol{K_P}$ and partial coset representatives $\boldsymbol{Rep_P}$* ***Result**: The kernel $\boldsymbol{K}$ and coset representatives $\boldsymbol{Rep}$.*

begin
  $k \leftarrow dim(K_P)$
  if $k = dim(K_P)$ ***is*** $\{0,1\}$ ***then***
  $K, Rep \leftarrow$ *Algorisme* α
  else
  $K, Rep \leftarrow$ *Algorisme* β
  end if
  return K, Rep
end

No obstant, aquest número s'ha extret en base a un mostreig de 27 codis i a més va estretament lligat amb la implementació interna de la funció *QaryCode(K,Rep)* cosa que implica que aquest patró no sempre es té perquè complir. Tenint en compte aquesta darrera observació es proposa que de cara a la implementació del paquet de codis *q-aris,* a l'hora de construir codis l'usuari pugui triar quina opció prefereix, on es proposa que l'opció per defecte sigui la de l'algoritme $\boldsymbol{\Omega}$, una segona opció on l'usuari triï l'algorisme concret i una tercera on l'usuari triï a partir de quin *kernel* es realitzarà la transició d'un algorisme a l'altre.

4 DESENVOLUPAMENT DE NOVES FUNCIONS EN EL PAQUET DE CODIS *Q-ARIS*

En aquesta part del treball s'han desenvolupat tests de validació per a les funcions, posteriorment implementades, corresponents a diversos apartats del manual *q-ari* [13].

Primer de tot a l'apartat de codis *q-ari* trivials (*Secció 1.2.2* [13]). Aquest apartat contenia les següents funcions: *ZeroQaryCode(F,n)*, *RepetitionQaryCode(F,n)*, *UniverseQaryCode(F,n)*, *RandomQaryCode(F,n,M)* i *RandomQaryCode(F,n,M,k).*

En aquestes funcions el que es pretén és la construcció de codis mitjançant diferents mètodes, on entra en joc l'ús de codis de repetició, codis aleatoris, codis univers, i codis zero.

Més endavant s'han desenvolupat aquelles funcions contingudes en l'apartat d'invariants per a codis *q-aris* (*Secció 1.3* [13]) les quals han sigut les següents: *Lenght(C)*, *#C*, *Parameters(C)* i *InformationRate(C).* Dins el mateix apartat, també s'han desenvolupat les funcions corresponents a l'espai vectorial on s'inclou el codi: *SpanCode(C)*, *DimensionOfSpan(C)*, *Rank(C)*, *KernelCode(C)* i *DimensionOfKernel(C).*

Totes aquestes funcions han estat validades, o sigui, s'ha tingut present comparar la sortida esperada de la funció amb la sortida concreta de cada funció de manera que per validar-se els dos resultats havien de coincidir. A més aquests tests s'han dut a terme mitjançant l'ús dels 29 codis presentats en la Taula 1 de l'Apartat 3.1 d'aquest document en forma de taula classificatòria. Aquesta mostra de codis, de propietats diferents, ha sigut molt útil per testar el comportament de cada funció i verificar-ne els resultats.

En el conjunt de funcions dels 3 apartats mencionats del manual *q-ari*: els codis trivials, els invariants i els de l'espai codi, el desenvolupament d'aquestes tenia com a finalitat l'expansió de funcionalitats de la llibreria de codis *q-ari,* en Magma, amb la que s'està treballant. Per tant amb la validació d'aquestes funcions mitjançant els tests, s'ha aconseguit ampliar funcionalitats per al paquet *q-ari*.

5 DESENVOLUPAMENT I ANÀLISI DE LES FUNCIONS DE CÀLCUL DE PES I DISTÀNCIA MÍNIMA

En aquest apartat, l'objectiu ha sigut poder implementar dos algoritmes eficients en el càlcul del pes mínim i la

distància mínima. La metodologia emprada ha sigut similar a la de l'Apartat 3, ja que l'objectiu amb les dues funcions del càlcul de distància i pes, és la de trobar la combinació més eficient de diferents mètodes d'obtenció d'aquests paràmetres. Per valorar com de millor pot ser un mètode respecte un altre, novament s'ha pres com a mesura de rendiment la dimensió del *kernel* d'entrada, comprovant el comportament de cada mètode amb la mostra dels 29 codis mencionats anteriorment.

5.1 Introducció a les funcions de pes i distància mínims

En la secció 1.7 del manual de codis *q-aris* [13], s'introdueixen dues funcions que determinen el càlcul del pes mínim i de la distància mínima, i que s'anomenen: *MinimumWeight(C)* i *MinimumDistance(C)* respectivament. En el sistema Magma ja existeixen dues funcions molt eficients per al càlcul d'aquests dos paràmetres, anomenades *MinimumWeight i MinimumDistance*. El problema d'aquestes dues funcions és que només funcionen per a codis lineals. Per aquest motiu, en aquest subobjectiu el que es pretén és implementar aquestes dues noves funcions per a codis no lineals.

Aquestes dues funcions poden tenir diferents mètodes del càlcul. Per tant, en els següents apartats s'estudien dos mètodes concrets: el *brute force* i l'*extend cosets,* de cara a comparar el rendiment d'aquests per a les noves funcions de pes i distància mínims.

5.2 Mètode *brute force*

Aquest primer mètode proposat serà el que seguiran les funcions *MinWeightBruteForce(K, Rep, AlgMethod)* i *MinDistanceBruteForce(K, Rep, AlgMethod).* En aquestes dues funcions tenim 3 paràmetres d'entrada: el *kernel* del codi inicial, els representants dels *cosets* del codi i el mètode algorítmic que s'utilitzarà per als subcodis lineals.

Tenint en compte la representació *kernel*/*cosets* donada per l'equació (1), per al càlcul del pes i distància mínima d'un codi *q-ari*, cal tenir en compte les paraules-codi contingudes en el kernel i en la resta de *cosets*.

Per al càlcul del pes o distància mínima del *kernel*, tindrem presents dues propietats dels codis lineals:

1. El *kernel* per definició és sempre un codi lineal.

2. Per a un codi lineal, el pes i la distància mínima sempre coincideixen.

Per tant, tenint en compte que les dues funcions ja implementades en el sistema Magma funcionen només per a codis lineals, la idea ha sigut trobar la distància mínima, d_k, del *kernel* a partir de la funció *MinimumDistance* del sistema Magma. Posteriorment es troba el pes mínim del *kernel*, w_k, degut a l'equivalència $d_k = w_k$ en codis lineals. D'aquesta manera, tant per la funció *MinWeightBruteForce* com per la *MinDistanceBruteForce* inicialment s'obtenen els dos paràmetres buscats, només per el *kernel*.

El paràmetre *AlgMethod* determina quin mètode algorítmic de càlcul utilitzarà el sistema Magma per als codis lineals, o sigui, en aquest cas, per al *kernel*. Hi ha tres opcions: l'algoritme *Distribution*, el *Zimmerman*, o la possibilitat de deixar el paràmetre com a *Auto* per a que sigui el propi sistema el que triï. L'opció *Distribution* calcula el pes mínim a partir de la distribució de pesos del codi, mentre que l'opció *Zimmerman* utilitza el conegut algoritme *Brouwer-Zimmerman* [8] per al càlcul de pes mínim. En aquest treball s'ha utilitzat l'opció *Auto.*

Per calcular el pes mínim del codi *q-ari*, després de calcular el pes mínim del seu *kernel*, es calcula el pes mínim de cada *coset*. Per calcular el pes mínim d'un *coset* concret, *K(C) + c*, es generen tots els elements del *coset* com a suma del representant *c* amb els diferents vectors del *kernel*. Un cop recorreguts tots els *cosets*, la funció es queda amb el pes mínim d'entre tots els valors obtinguts. Finalment, compara aquest valor amb el de w_k i el menor dels dos serà el pes mínim d'aquell codi.

Per calcular la distància mínima del codi *q-ari*, després de calcular la distància mínima (o pes mínim) del seu *kernel*, es calcula la distància mínima entre cada parella de dos *cosets* diferents. Concretament, per calcular la distància mínima entre els elements del *coset*, amb representant *c1* i el *coset* amb representants *c2*, es generen tots els elements del *coset* $K(C) + c1 - c2$ i es calcula el pes mínim. D'aquesta manera podem assegurar haver calculat cada parella d'elements de tot el conjunt de *cosets*. Un cop fet això, es compara el pes mínim obtingut per a cada parella de *cosets* amb d_k, i el menor dels dos valors serà la distància mínima d'aquell codi.

A continuació es mostren els pseudocodis de les dues funcions presentades.

TAULA 5
PSEUDOCODI DE LA FUNCIÓ MINWEIGHTBRUTEFORCE

***Data**: A kernel **K** and coset representatives **Rep** and a string with the algorithm method "**Auto**".*
***Result**: An integer with the minimum weight.*

```
begin
  minWeight ← MinimumWeight(K: Method = "Auto")
  for i in [1 .. nº Rep] do
    if minWeight = 1 then
      return 1
    end if
    for k in K do
      minWeight ← min(minWeight, weight(k +Rep))
    end for
  end for
  return minWeight
end
```

TAULA 6
PSEUDOCODI DE LA FUNCIÓ MINDISTANCEBRUTEFORCE

```
Data: A kernel K and coset representatives Rep and a string with the
algorithm method "Auto".
Result: An integer with the minimum distance.
begin
    minDistance ← MinimumWeight(K: Method = "Auto")
    for i in [1 .. nº Rep-1] do
      for j in [i+1 .. nº Rep] do
        if minDistance = 1 then
            return 1
        end if
        for k in K do
          minDistance ← min(minDistance, weight(k + Rep[i]-Rep[j]))
        end for
      end for
    end for
  return  minDistance
end
```

5.3 Mètode *extend cosets*

El segon mètode proposat serà el que seguiran les funcions *MinWeightExtendCosets(K, Rep, AlgMethod)* i *MinDistanceExtendCosets(K, Rep, AlgMethod)*. Novament disposem dels 3 paràmetres introduïts en l'apartat anterior. En aquest el paràmetre *AlgMethod* seguirà sent *Auto*.

Quan es vulgui calcular el pes mínim de tot el conjunt de *cosets* es tindran en compte dues propietats:

1. Les possibles combinacions entre el *kernel* i qualsevol dels representants dels *cosets* d'un codi no lineal tindran per resultat un codi lineal.

2. El pes mínim d'aquesta combinació tindrà el mateix valor que el pes mínim de tots els elements del *kernel* i el *coset*.

Tenint en compte aquestes dues propietats, el mètode de *extend cosets* calcularà el pes mínim del codi lineal resultant de la combinació del *kernel* amb cada representant, $< K, c1 >= K_{c1}$, amb la funció *MinimumDistance* ja implementada en el sistema Magma per a codis lineals i obtindrà com a resultat el valor de la distància mínima de cada *coset* (la qual serà equivalent al pes) així com la del *kernel*. Aquest procés es farà iterativament per a cada representant fins a obtenir el pes mínim de tot el codi *q-ari*.

En el cas de la distància, es tindrà en compte la combinació del *kernel* amb un representant, $< K(C), c1 >= K_{C1}$, i la combinació del *kernel* amb la diferència de dos representants, $< K(C) + c1 - c2 >$.

A continuació es mostres els pseudocodis de les dues funcions presentades.

TAULA 7
PSEUDOCODI DE LA FUNCIÓ MINWEIGHTEXTENDCOSETS

```
Data: A kernel K and coset representatives Rep and a string with the
algorithm method "Auto".
Result: An integer with the minimum weight.
begin
  minWeightK ← MinimumWeight(K : Method := "Auto")
  basisKernel ← Basis(K)
  n ← Length(K)
  minWeight ← MinimumWeight(LinearCode<BaseField(K),
                n | basisKernel cat [Rep[1]]>: Method = "Auto")
    for i in [2 .. nº Rep] do
      if minWeight = 1 then
          return 1
      end if
      minWeight ← Minimum(minWeight, Mini-
                  -mumWeight(LinearCode<BaseField(K),
                  n | basisKernel cat [Rep[i]]> : Method =
                  "Auto"))
    end for
    isMinWeight ← IsZero(Rep[1])or(minWeight < minWeightK)
    if not isMinWeight then
      C ← QaryCode(K, Rep)
      minWeight ← WeightDistribution(C)[2][1]
    end if
  return minWeight
end
```

TAULA 8
PSEUDOCODI DE LA FUNCIÓ MINDISTANCEEXTENDCOSETS

```
Data: A kernel K and coset representatives Rep and a string with the
algorithm method "Auto".
Result: An integer with the minimum distance.
begin
    minDistance ← MinimumWeight(K: Method = "Auto")
    basisKernel ← Basis(K)
    for i in [1 .. nº Rep - 1] do
      for j in [i+1 .. nº Rep] do
        if minDistance = 1 then
            return 1
        end if
        for k in K do
        minDistance ← min(minDistance, MinimumWeight(K,
                      Length(K) | basisKernel cat Rep[i]- Rep[j]),
                      Method = "Auto")
      end for
    end for
  end for
  return  minDistance
end
```

5.4 Observacions i estudi del rendiment

En els dos apartats anteriors s'han presentat els mètodes *brute force* i *extend cosets* per a les funcions de les Taules 5, 6, 7 i 8. Aquests dos mètodes han de donar resultats equivalents entre ells, però no necessàriament el mateix rendiment. Previ a aquest estudi, les 4 funcions han estat degudament validades a partir de diversos tests i posteriorment se'ls hi ha aplicat l'estudi de rendiment que a continuació es presenta.

Per determinar l'eficiència dels dos mètodes, igual que en l'Apartat 3, s'han realitzat unes proves de rendiment, on s'han utilitzat els 29 codis presentats en la Taula 1 d'aquest treball. Aquests codis s'han tractat en base a les funcions de pes i distància mínims, executant 20 vegades cada codi amb cada funció tractada amb els dos mètodes i fent una mitjana dels temps, per poder observar diferències en el rendiment segons la dimensió del *kernel* proposada. Aquests resultats es poden observar en les taules de temps dels Apèndixs A3 i A4.

Un cop analitzats els resultats en les taules de temps, el primer que es pot observar és que alguns dels codis testejats no eren prou grans com per aportar resultats significatius en els temps. Aquest inconvenient ja es va veure per a l'algoritme $\boldsymbol{\Omega}$. Tot i així, en tots aquells codis grans s'han pogut detectar canvis de temps a partir d'una dimensió concreta del *kernel* parcial.

Cal destacar com a segona observació que, comparant les dues taules de pes i distància podem observar les semblances en els canvis de dinàmica de temps, els quals s'han produït exactament per als mateixos valors de dimensió de *kernel* en cada codi.

Vist això la principal conclusió a extreure és que per a codis petits, categoria on entrarien la majoria dels testejats, el mètode *brute force* resulta ser molt eficient, sobretot a partir del moment en que el mètode *extend cosets* comença a trigar més. No obstant, en codis grans on el *kernel* parcial és més petit i per tant, hi ha més representants, el mètode *brute force* tant en pes com en distància patiria un augment en el temps d'execució, en canvi, el mètode *extend cosets* no ho notaria tant i treballaria amb més eficiència.

6 CONCLUSIONS I AMPLIACIONS

Aquest treball ha assolit diversos objectius. El principal ha sigut el de contribuir en el desenvolupament de la llibreria en Magma per treballar amb codis *q-aris* no lineals de forma eficient [13]. Més concretament la idea ha sigut assolir-ho tot millorant l'eficiència de funcions ja implementades dins el paquet, i implementant-ne de noves seguint l'estructura del manual del paquet proporcionat.

Durant aquest procés s'han anat complint parts d'aquest objectiu entre les que destaquen la necessitat d'un estudi a nivell teòric de la teoria de codis per a poder desenvolupar correctament les funcions i també els tests que les validessin, així com per al procés de formulació dels algorismes principals d'aquest treball.

La major part de les funcions implementades han resultat ser curtes ja que només existia una possible implementació que ja resultava ser l'eficient. En canvi, en els algorismes de construcció de codis i els de càlcul pes mínim i distància mínima, la implementació ha resultat ser més complexa ja que hi ha hagut un estudi previ per veure quin era el mètode més eficient a l'hora d'implementar-los segons la dimensió del *kernel*.

Tant la implementació de totes les funcions de l'Apartat 4 com la dels algoritmes dels Apartats 3 i 5, han conformat la part més experimental ja mencionada en l'apartat de la metodologia.

En l'Apartat 3 corresponent a la construcció de codis *q-ari*, tot analitzant l'estudi de rendiment dels algoritmes $\boldsymbol{\alpha}$ i $\boldsymbol{\beta}$, s'ha dissenyat l'algoritme $\boldsymbol{\Omega}$. Amb aquest anàlisi, s'ha pogut determinar que l'algorisme $\boldsymbol{\Omega}$ es bifurca en l'$\boldsymbol{\alpha}$ i el $\boldsymbol{\beta}$ segons la dimensió del *kernel* parcial. Per a *kernels* majors de 1 s'ha pogut comprovar que l'opció més eficient és la de la l'algoritme β, sempre en base al mostreig codis presentats en la Taula 1 d'aquest treball, i tenint present la implementació interna de la funció *QaryCode(K,Rep)*, cosa que implica que aquest patró no sempre es té perquè complir. Quan la dimensió del *kernel* parcial passa a ser 1 o 0, l'algoritme preferible és l'α.

No obstant, degut a que aquest patró no té perquè complir-se amb tots els codis existents es proposa, de cara a la implementació del paquet de codis *q-ari*, que, a l'hora de construir codis, l'usuari pugui triar quina opció prefereix, on es proposa que l'opció predeterminada sigui la de l'algoritme $\boldsymbol{\Omega}$, una segona opció on l'usuari trií l'algorisme concret i una tercera on l'usuari trií a partir de quin *kernel* es realitzarà la transició d'un algorisme a l'altre. Com a ampliació d'aquest estudi seria possible determinar amb més exactitud aquest patró en el canvi de rendiment dels algoritmes $\boldsymbol{\alpha}$ i $\boldsymbol{\beta}$ sempre que es realitzessin més tests de rendiment amb una mostra major de codis.

Finalment en l'Apartat 5 d'aquest treball s'han dut a terme dos estudis de rendiment. Un, on es comparaven els mètodes *brute force* i *extend cosets* per al càlcul del pes mínim, i el segon, on es comparaven per al càlcul de la distància mínima. Tal i com s'ha observat comparant els resultats dels dos mètodes tant en pes com en distància, els resultats han sigut similars. Els canvis de temps s'han produït en la mateixa dimensió de *kernel* parcial segons el codi testejat. Aquells codis prou grans com per aportar resultats significatius han demostrat que hi ha un patró comú en el càlcul de distància o pes mínims segons el mètode aplicat, i s'ha pogut apreciar que per a codis petits, categoria on estarien la major part dels codis testats, el mètode de *brute force* resulta ser el més efectiu, en canvi per als codis més grans s'ha pogut observar que, a partir de certa dimensió de *kernel* parcial, el mètode *brute force* deixa de ser eficient i en canvi, l'*extend* cosets treballa amb més eficiència amb aquests codis independentment de la dimensió del *kernel*.

Per ampliar aquest estudi, igual que per l'estudi de l'Apartat 3, es podrien testar proves de rendiment per una mostra major de codis per trobar més resultats diferents i intentar establir un patró més exacte.

AGRAÏMENTS

L'autor vol agrair l'oportunitat donada i el gran recolzament rebut pels seus tutors, Mercè Villanueva i Jaume Pujol, els quals l'han guiat al desenvolupament d'aquest projecte. A més se'ls hi agraeix, juntament amb el dEIC i la UAB, haver cedit un espai de treball idoni amb totes les facilitats possibles. Ha estat un plaer treballar amb vosaltres.

BIBLIOGRAFIA

[1] R. D. Barrolleta, J. Pernas, J. Pujol, and M.Villanueva, Codes over Z4. A Magma package, version 2.0, Universitat Autònoma de Barcelona, 2016. http://ccsg.uab.cat.

[2] J. Borges, C. Fernández, J. Pujol, J. Rifà, and M. Villanueva, Z2Z4-linear codes. A Magma package, version 3.5, Universitat Autònoma de Barcelona, 2012. http://ccsg/uab.cat.

[3] W. Bosma, J. J. Cannon, C. Fieker, and A. Steel (eds.), *Handbook of Magma functions*, Edition 2.22, 5669 pages, 2016. http://magma.maths.usyd.edu.au/magma/

[4] J. J. Cannon and C. Playoust, "First Steps in MAGMA," School of Mathematics and Statistics, Australia, August 1996.

[5] Computacional Algebra Group, "Overview of MAGMA V2.17 Features," University of Sidney, January 2016.

[6] C. Dieguez, "Ampliació d'una llibreria en MAGMA per a codis Z2Z4-lineals", projecte fi de grau, Dept. d'Enginyeria de la Informació i de les Comunicacions, Universitat Autònoma de Barcelona, Juny 2016.

[7] V. Gonzalez, "Codis binaris no lineals òptims: Propietats i construccions", projecte fi de grau, Dept. d'Enginyeria de la Informació i de les Comunicacions, Universitat Autònoma de Barcelona, Juny 2016.

[8] W.C Huffman and V. Pless, *Fundalmentals of error-correcting codes*, New York: Cambridge University Press, 2003.

[9] M. Kiermaier, A. Wassermann, J. Zwanzger, "New upper bounds on binary linear codes and a Z4-code with a better-than-linear Gray image", New York: Cornell University, 2016. https://arxiv.org/abs/1503.03394v2

[10] F. J. MacWilliams, *The theory of error-correcting codes*, North-Holland Mathematical Library, 1977.

[11] 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, 2001.

[12] Pujol and M. Villanueva, "Binary Codes. A Magma Package", version 2.0, Universitat Autònoma de Barcelona, 2014. http://ccsg.uab.cat

[13] J. Pujol and M. Villanueva, "Manual Q-ary Codes. Magma Packages", Universitat Autònoma de Barcelona, 2016. http://ccsg.uab.cat

[14] M. Rubio, "Desarrollo orientado a pruebas (TDD)," blog, January 1, 2009; http://altenwald.org/2009/01/08/desarrollo-orientado-a-pruebas-tdd/

[15] "Software Gap", 2016. http://www.gap-system.org/

[16] "Software Magma", 2017. https://magma.maths.usyd.edu.au/magma/overview/pdf/overv217.pdf

[17] "Software Maple", 2016. https://www.maplesoft.com/support/help

[18] "Software Sage", 2016. http://www.sage.es

[19] P. A. Vaca, C. Maldonado, C. Inchaurrondo, J. Peretti, M. S. Romero, M. Bueno, "Test-Driven Development: Una aproximación para entender su utilidad en el proceso de desarrollo de Soft-ware.," Universidad Tecnológica Nacional, Facultad Regional Córdoba.

[20] F. Zeng, "Nonlinear codes: Representation, constructions, minimum distance computation and decoding", doctor's degree thesis, Dept. d'Enginyeria de la Informació i de les Comunicacions, Universitat Autònoma de Barcelona, Juny 2014.

Erik Vicent Ferrer és estudiant de quart de grau en Enginyeria Informàtica a la Universitat Autònoma de Barcelona. El 2016 va col·laborar com a tècnic de suport a la recerca en el Departament d'Enginyeria de la Informació i de les Comunicacions.

APÈNDIXS

A1. APÈNDIX 1: PSEUDOCODI DE LES FUNCIONS *QARYCODES(L), QARYCODES(L,K) I QARYCODES(K, REP).*

Pseudocodi 1: QaryCode(L):

Data: *A q-ary code L as a sorted sequence of codewords.*
Result: *The kernel* $\boldsymbol{K}$ *and coset representatives* $\boldsymbol{Rep} = \{\boldsymbol{v_1}, \dots, \boldsymbol{v_t}\}$.

```
begin
 K ← {0}
 L* ← L\{0}
 Rep ← {}
 R ← {}

 while |L*| > 0 do
    c ← First(L*)
   if {L* + λc + : λ ∈ F_q\{0}} ⊆ L then
      L* ← L* \ {K + λc : λ ∈ F_q\{0}}
      K ← {K + λc : λ ∈ F_q}
   else
      for λ ∈ F_q \{0} such that λc ∈ L do
          R ← R ∪ { K + λc}
          L* ← L* \ { K + λc}
     end for
  end if
end while

while Rep ≠ Ø do
    v ← First(R)
    Rep ← Rep ∪ {v}
    R ← R \ {K + v}
end while
return K, Rep
end
```

Pseudocodi 2: QaryCode(L,K):

Data*: A q-ary code* $\boldsymbol{L}$ *as a sorted sequence of codewords. A partial kernel* $\boldsymbol{K_P}$.
Result: *The kernel* $\boldsymbol{K}$ *and coset representatives* $\boldsymbol{Rep} = \{\boldsymbol{v_1}, \dots, \boldsymbol{v_t}\}$.

```
begin
 K ← K_P
 L* ← L − K
 Rep ← {}
 R ← {}

 while |L*| > 0 do
    c ← First(L*)
   if {L* + λc + : λ ∈ F_q\{0}} ⊆ L then
      L* ← L* \ {K + λc : λ ∈ F_q\{0}}
      K ← {K + λc : λ ∈ F_q}
   else
      for λ ∈ F_q \{0} such that λc ∈ L do
          R ← R ∪ { K + λc}
          L* ← L* \ { K + λc}
     end for
  end if
end while

while Rep ≠ Ø do
    v ← First(R)
    Rep ← Rep ∪ {v}
    R ← R \ {K + v}
end while
return K, Rep
end
```

Pseudocodi 3: QaryCode(K,Rep):

Data: *A partial kernel* $\boldsymbol{K_P}$ *and partial coset representatives* $\boldsymbol{Rep_P} = \{\boldsymbol{v_1}, \dots, \boldsymbol{v_t}\}$.
Result: *The kernel* $\boldsymbol{K}$ *and coset representatives* $\boldsymbol{Rep} = \{\boldsymbol{v_1}, \dots, \boldsymbol{v_t}\}$.

```
begin
 K ← K_P
 Rep ← Rep_P

 while |Rep_P| > 0 do
    c ← First(Rep_P)
   if {Rep + λc + : λ ∈ F_q } ⊆ C then
      K ← {K + λc : λ ∈ F_q}
      Rep ← Rep \{c}
   end if
 end while
 return K, Rep
end
```

A2. APÈNDIX 2: ESTUDI DE RENDIMENT DELS ALGORITMES α i β.

En aquesta taula s'ha fet un estudi de rendiment on s'ha comprovat el temps que triga els algoritmes α i β en construir els codis de la columna "Codis" segons una dimensió de *kernel* parcial donada. En vermell es mostra la dimensió de *kernel* parcial amb la que l'algoritme β passa a trigar més en construir el codi *q-ari*.

Temps en segons		Dimensió dels *Kernels* parcials											
Codis	**Algoritmes**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
CHamq2n15ker11	α	0.005	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.004	0.005
	β	0.002	0.002	0.002	0.002	0.003	0.003	0.004	0.005	0.010	0.019	0.046	0.128
CHamq2n16ker4	α								0.022	0.029	0.043	0.047	0.051
	β								0.006	0.011	0.025	0.051	0.135
CHamq2n16ker4t	α								0.022	0.030	0.043	0.048	0.051
	β								0.006	0.012	0.025	0.051	0.136
CKerq2n256ker9	α			0.538	0.698	0.867	1.045	1.216	1.388	1.593	1.841	2.192	2.522
	β			0.067	0.077	0.098	0.139	0.267	0.734	2.458	8.850	33.290	156.507
Cq2n10ker0	α												0.008
	β												0.003
Cq2n20ker0	α												0.011
	β												0.001
Cq2n20ker5	α							0.001	0.001	0.002	0.001	0.002	0.003
	β							0.001	0.000	0.001	0.001	0.003	0.004
Cq3n13ker8a	α				0.671	0.891	1.106	1.354	1.571	1.821	2.039	2.295	2.653
	β				0.052	0.050	0.054	0.059	0.080	0.203	1.284	10.676	96.217
Cq3n13ker8b	α				0.594	0.787	0.993	1.227	1.427	1.656	1.858	2.108	2.457
	β				0.047	0.048	0.050	0.055	0.075	0.186	0.123	9.200	86.212
Cq3n13ker8c	α				0.645	0.861	1.077	1.323	1.532	1.784	1.994	2.261	2.626
	β				0.051	0.052	0.054	0.059	0.080	0.197	1.180	9.784	91.338
Cq3n13ker9a	α			0.380	0.585	0.807	1.041	1.256	1.475	1.684	1.910	2.148	2.372
	β			0.054	0.054	0.055	0.057	0.061	0.080	0.197	1.189	9.879	93.786
Cq3n13ker9b	α			0.385	0.589	0.808	1.042	1.261	1.482	1.694	1.924	2.163	2.380
	β			0.054	0.055	0.055	0.057	0.061	0.080	0.200	1.202	10.013	94.239
Cq3n13ker9c	α			0.387	0.591	0.814	1.004	1.264	1.486	1.699	1.931	2.171	2.392
	β			0.055	0.055	0.057	0.057	0.061	0.081	0.200	1.208	10.101	95.248
Cq3n13ker10	α		0.178	0.179	0.179	0.181	0.181	0.177	0.176	0.179	0.185	0.200	0.204
	β		0.054	0.054	0.057	0.055	0.057	0.064	0.080	0.197	1.211	10.108	95.845
CHadq4n4ker2	α										0.001	0.001	0.000
	β										0.000	0.000	0.001
CHadq4n8ker1	α											0.001	0.001
	β											0.001	0.001
CHadq4n12ker1	α											0.001	0.001
	β											0.001	0.004
CHadq4n16ker1a	α											0.001	0.002
	β											0.001	0.002
CHadq4n16ker1b	α											0.002	0.002
	β											0.001	0.002
CHadq4n16ker1c	α											0.002	0.004
	β											0.001	0.002
CHadq4n16ker2a	α										0.001	0.001	0.001
	β										0.000	0.001	0.002
CHadq4n16ker2b	α										0.001	0.001	0.001
	β										0.000	0.001	0.002
CHadq4n16ker3	α									0.000	0.001	0.000	0.000
	β									0.001	0.000	0.001	0.002
CHadq4n16ker3t	α									0.000	0.000	0.001	0.000
	β									0.001	0.000	0.001	0.002
Cq4n4ker2	α										0.001	0.001	0.001
	β										0.000	0.001	0.001
Cq4n16ker0	α												0.003
	β												0.001
Cq8n20ker3	α									0.016	0.039	0.015	0.079
	β									0.002	0.006	0.015	0.190

A3. APÈNDIX 3: ESTUDI DE RENDIMENT EN EL CÀLCUL DE PES MÍNIM.

En aquesta taula s'ha fet un estudi de rendiment on s'ha comprovat el temps que triga els algoritmes *Extend-Cosets i BruteForce* en el càlcul del pes mínim dels codis de la columna "Codis" segons una dimensió de *kernel* parcial donada. En vermell es mostra la dimensió de *kernel* parcial amb la que *ExtendCosets* passa a trigar més en calcular el pes mínim.

Taules de rendiment del PES		Dimensió dels *Kernels* Parcials											
Codis	**Algoritmes**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
CHamq2n15ker11	**ExtendCosets**	0.000	0.000	0.001	0.002	0.006	0.012	0.029	0.054	0.129	0.318	0.030	0.055
	BruteForce	0.000	0.002	0.002	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.004	0.003
CHamq2n16ker1	**ExtendCosets**											0.032	0.060
	BruteForce											0.003	0.004
CHamq2n16ker4	**ExtendCosets**								0.028	0.048	0.170	0.030	0.055
	BruteForce								0.003	0.003	0.004	0.004	0.004
CHamq2n16ker4t	**ExtendCosets**								0.001	0.001	0.001	0.001	0.002
	BruteForce								0.001	0.000	0.001	0.001	0.001
Chamq2n16ker9	**ExtendCosets**			0.001	0.001	0.004	0.007	0.018	0.044	0.0065	0.141	0.030	0.057
	BruteForce			0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.004	0.003	0.004
CKerq2n256ker9	**ExtendCosets**			0.216	0.421	1.000	1.966	3.878	0.264	0.501	0.958	1.906	3.326
	BruteForce			0.086	0.086	0.086	0.085	0.089	0.088	0.092	0.099	0.111	0.122
Cq2n10ker0	**ExtendCosets**												0.000
	BruteForce												0.000
Cq2n20ker0	**ExtendCosets**												0.004
	BruteForce												0.000
Cq2n20ker5	**ExtendCosets**							0.001	0.002	0.005	0.012	0.003	0.004
	BruteForce							0.000	0.001	0.001	0.001	0.001	0.000
Cq3n13ker8a	**ExtendCosets**				0.001	0.005	0.016	0.110	0.298	1.295	4.038	12.550	1.442
	BruteForce				0.064	0.069	0.072	0.072	0.072	0.074	0.077	0.087	0.102
Cq3n13ker8b	**ExtendCosets**				0.001	0.005	0.016	0.113	0.307	1.319	4.095	13.018	1.500
	BruteForce				0.066	0.073	0.072	0.073	0.075	0.075	0.078	0.090	0.108
Cq3n13ker8c	**ExtendCosets**				0.001	0.005	0.015	0.108	0.295	1.300	4.054	12.585	1.532
	BruteForce				0.065	0.072	0.073	0.073	0.078	0.075	0.078	0.090	0.108
Cq3n13ker9a	**ExtendCosets**			0.001	0.001	0.004	0.025	0.108	0.438	1.295	4.056	12.499	1.452
	BruteForce			0.049	0.067	0.071	0.074	0.075	0.074	0.076	0.078	0.089	0.107
Cq3n13ker9b	**ExtendCosets**			0.000	0.002	0.004	0.026	0.106	0.439	1.296	4.064	12.501	1.442
	BruteForce			0.048	0.066	0.071	0.073	0.073	0.074	0.075	0.078	0.089	0.106
Cq3n13ker9c	**ExtendCosets**			0.000	0.001	0.004	0.026	0.106	0.440	1.298	4.066	12.250	1.466
	BruteForce			0.049	0.066	0.071	0.073	0.074	0.075	0.075	0.078	0.090	0.106
Cq3n13ker10	**ExtendCosets**		0.000	0.000	0.002	0.008	0.038	0.136	0.444	1.312	4.085	12.614	1.545
	BruteForce		0.000	0.050	0.065	0.071	0.073	0.074	0.074	0.075	0.077	0.090	0.107
CHadq4n4ker2	**ExtendCosets**										0.000	0.000	0.002
	BruteForce										0.000	0.000	0.000
CHadq4n8ker1	**ExtendCosets**											0.002	0.001
	BruteForce											0.001	0.001
CHadq4n12ker1	**ExtendCosets**											0.003	0.002
	BruteForce											0.000	0.000
CHadq4n16ker1a	**ExtendCosets**											0.006	0.002
	BruteForce											0.001	0.001
CHadq4n16ker1b	**ExtendCosets**											0.005	0.002
	BruteForce											0.001	0.001
CHadq4n16ker1c	**ExtendCosets**											0.006	0.002
	BruteForce											0.001	0.000
CHadq4n16ker2a	**ExtendCosets**										0.002	0.005	0.002
	BruteForce										0.000	0.001	0.000
CHadq4n16ker2b	**ExtendCosets**										0.001	0.004	0.002
	BruteForce										0.000	0.000	0.001
CHadq4n16ker3	**ExtendCosets**									0.000	0.001	0.006	0.002
	BruteForce									0.001	0.000	0.000	0.001
CHadq4n16ker3t	**ExtendCosets**									0.001	0.003	0.006	0.002
	BruteForce									0.000	0.001	0.000	0.000
Cq4n4ker2	**ExtendCosets**										0.000	0.001	0.001
	BruteForce										0.000	0.000	0.000
Cq4n16ker0	**BruteForce**												0.001
	ExtendCosets												0.000
Cq8n20ker3	**ExtendCosets**									0.005	0.044	0.442	0.075
	BruteForce									0.003	0.003	0.005	0.004

A4. APÈNDIX 4: ESTUDI DE RENDIMENT EN EL CÀLCUL DE DISTÀNCIA MÍNIMA.

En aquesta taula s'ha fet un estudi de rendiment on s'ha comprovat el temps que triga els algoritmes *ExtendCosets i BruteForce* en el càlcul de la distància mínima dels codis de la columna "Codis" segons una dimensió de *kernel* parcial donada. En vermell es mostra la dimensió de *kernel* parcial amb la que *ExtendCosets* passa a trigar més en calcular la distància mínima.

Taules de rendiment del DISTÀNCIA		Dimensió dels *Kernels* Parcials											
Codis	**Algoritmes**	**11**	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
CHamq2n15ker11	**ExtendCosets**	0.000	0.001	0.001	0.006	0.041	0.224	1.059	3.875	18.300	90.988	16.832	62.836
	BruteForce	0.000	0.002	0.005	0.013	0.027	0.057	0.115	0.226	0.490	1.014	2.242	4.595
CHamq2n16ker1	**ExtendCosets**											17.213	63.864
	BruteForce											2.170	4.597
CHamq2n16ker4	**ExtendCosets**								1.975	6.688	47.555	16.782	63.524
	BruteForce								0.217	0.466	0.977	2.173	4.526
CHamq2n16ker4t	**ExtendCosets**								1.967	6.765	47.950	16.716	63.029
	BruteForce								0.217	0.483	0.985	2.167	4.627
Chamq2n16ker9	**ExtendCosets**			0.001	0.003	0.028	0.112	0.633	3.079	8.869	37.418	16.757	63.094
	BruteForce			0.005	0.012	0.026	0.055	0.108	0.221	0.455	0.968	2.188	4.521
CKerq2n256ker9	**ExtendCosets**			0.728	2.838	*	*	*	*	*	*	*	*
	BruteForce			0.345	0.681	*	*	*	*	*	*	*	*
Cq2n10ker0	**ExtendCosets**												0.001
	BruteForce												0.000
Cq2n20ker0	**ExtendCosets**												0.243
	BruteForce												0.018
Cq2n20ker5	**ExtendCosets**							0.001	0.006	0.044	0.200	0.066	0.243
	BruteForce							0.000	0.001	0.002	0.004	0.009	0.018
Cq3n13ker8a	**ExtendCosets**				0.001	0.003	0.030	0.690	5.656	*	*	*	*
	BruteForce				0.019	0.061	0.183	0.555	1.680	*	*	*	*
Cq3n13ker8b	**ExtendCosets**				0.000	0.004	0.032	0.710	5.743	*	*	*	*
	BruteForce				0.019	0.062	0.193	0.568	1.720	*	*	*	*
Cq3n13ker8c	**ExtendCosets**				0.000	0.003	0.033	0.725	5.972	*	*	*	*
	BruteForce				0.019	0.063	0.193	0.580	1.760	*	*	*	*
Cq3n13ker9a	**ExtendCosets**			0.000	0.001	0.003	0.054	0.682	8.833	*	*	*	*
	BruteForce			0.006	0.020	0.065	0.197	0.589	1.794	*	*	*	*
Cq3n13ker9b	**ExtendCosets**			0.000	0.001	0.003	0.057	0.728	8.918	*	*	*	*
	BruteForce			0.005	0.021	0.066	0.196	0.595	1.731	*	*	*	*
Cq3n13ker9c	**ExtendCosets**			0.000	0.001	0.003	0.054	0.674	8.593	*	*	*	*
	BruteForce			0.005	0.019	0.062	0.191	0.571	1.723	*	*	*	*
Cq3n13ker10	**ExtendCosets**		0.000	0.000	0.001	0.006	0.079	0.874	8.771	*	*	*	*
	BruteForce		0.000	0.005	0.020	0.064	0.194	0.591	1.736	*	*	*	*
CHadq4n4ker2	**ExtendCosets**										0.000	0.000	0.015
	BruteForce										0.000	0.000	0.000
CHadq4n8ker1	**ExtendCosets**											0.006	0.012
	BruteForce											0.001	0.001
CHadq4n12ker1	**ExtendCosets**											0.017	0.028
	BruteForce											0.001	0.003
CHadq4n16ker1a	**ExtendCosets**											0.038	0.052
	BruteForce											0.002	0.005
CHadq4n16ker1b	**ExtendCosets**											0.035	0.052
	BruteForce											0.001	0.005
CHadq4n16ker1c	**ExtendCosets**											0.043	0.052
	BruteForce											0.002	0.004
CHadq4n16ker2a	**ExtendCosets**										0.002	0.032	0.051
	BruteForce										0.001	0.002	0.004
CHadq4n16ker2b	**ExtendCosets**										0.002	0.033	0.052
	BruteForce										0.001	0.001	0.005
CHadq4n16ker3	**ExtendCosets**									0.000	0.001	0.042	0.052
	BruteForce									0.000	0.001	0.001	0.004
CHadq4n16ker3t	**ExtendCosets**									0.000	0.000	0.044	0.056
	BruteForce									0.000	0.001	0.001	0.004
Cq4n4ker2	**ExtendCosets**										0.000	0.001	0.001
	BruteForce										0.000	0.000	0.000
Cq4n16ker0	**BruteForce**												0.004
	ExtendCosets												0.004
Cq8n20ker3	**ExtendCosets**									0.014	0.955	77.700	98.674
	BruteForce									0.008	0.084	0.738	7.210