Implementació de funcions constructores de codis q-aris. Desenvolupament en MAGMA

Biplop Dey

Treball Final de Grau
Grau de Matemàtiques

Directora: Mercè Villanueva
Universitat Autònoma de Barcelona
Curs 2020-2021

Resum

L'objectiu principal d'aquest treball és contribuir en el desenvolupament de la llibreria Q-ARY CODES del sistema MAGMA per treballar amb codis q-aris no lineals de forma eficient. Concretament, hem desenvolupat les funcions constructores de codis q-aris `DirectSum(C, D)`, `DirectSum(Q)`, `PloktinSum(C, D)` i `PadCode(C, n)`. L'objectiu principal es divideix en quatre parts. La primera part ha consistit en l'estudi de la teoria de codis q-aris, sobre cossos finits amb q elements. Per als codis lineals s'han estudiat la codificació i la descodificació, i per a codis no lineals s'han demostrat resultats que permeten implementar les funcions citades de forma més eficient. La segona part ha consistit a aprendre el sistema MAGMA i la seva llibreria de codis lineals, a més de la llibreria Q-ARY CODES. En la tercera part s'han implementat les funcions i diferents algoritmes per a cada funció, segons la teoria de la primera part. Finalment, s'han validat les funcions i s'han fet estudis de rendiment per determinar quin algoritme de cada funció és millor, fent servir codis q-aris de la llibreria.

Abstract

The main objective of this project is to contribute to the development of the Q-ARY CODES library within MAGMA system to efficiently work with non-linear q-ary codes. Specifically, we have developed the functions `DirectSum(C, D)`, `DirectSum(Q)`, `PloktinSum(C, D)`, and `PadCode(C, n)` that construct q-aris codes. The main goal is divided into four parts. The first part has consisted of studying the theory of q-ary codes over finite fields with q elements. For linear codes, coding and decoding has been studied, and for nonlinear codes results that allow to implement the mentioned functions more efficiently have been proved. The second part has consisted of learning the MAGMA system and its library for linear codes, as well as the Q-ARY CODES library. In the third part the functions by using different algorithms have been implemented, according to the theory of the first part. Finally, the functions have been validated and performance studies have been performed to determine which algorithm for each function is the best, using the q-ary codes library.

"Potser l'eina més poderosa de les matemàtiques és la que ens permet considerar com a iguals coses que no ho són. Se'n diu el pas al quocient."

Jaume Aguadé

Agraïments a la meva tutora, Mercè Villanueva, per la direcció del treball i les seves correccions.

A la meva família, amics i companys per tot el suport.

Índex

1 Introducció

Tot el procés de transmissió d'informació requereix un canal per on circular, però sovint aquest canal té soroll i hi ha errors en la transmissió. La teoria de la codificació, que és la que estudiem en aquest treball, que s'ocupa de detectar i/o corregir aquests errors. Això ho fa mitjançant sistemes de codificació i descodificació d'informació. En la Figura 1 veiem com funciona tot el procés de transmissió d'informació digital. Primer es comprimeix el missatge per reduir la mida del missatge a enviar, segon es xifra per mantenir la privacitat i finalment es codifica fent servir codis correctors d'errors. Després d'enviar la informació per un canal, que pot ser a través fibres òptiques, d'ones electromagnètiques com connexió satelital, el wifi, el 5G, etc, es descodifica. Per això primer es mira si hi ha errors i s'intenta corregir, si no es pot garantir que la correcció estigui bé, es pot demanar retransmetre la informació. Després es desxifra i es descomprimeix. La teoria de codis té aplicacions en totes les àrees de telecomunicació, fins i tot en l'emmagatzematge de dades que pot ser al disc dur, a la memòria USB, al núvol, etc.

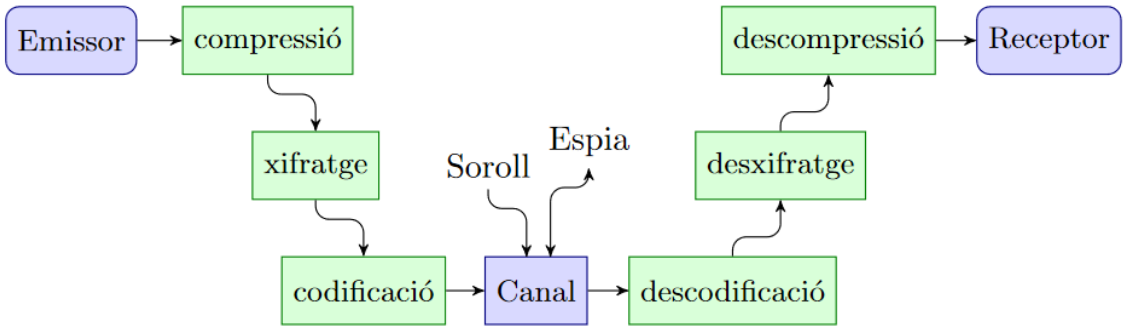

Figura 1: Sistema de comunicació digital, [14, Figura 1]

L'objectiu principal d'aquest treball és contribuir en el desenvolupament de la llibreria Q-ARY CODES del sistema MAGMA per treballar amb codis q-aris no lineals de forma eficient. El MAGMA té llibreries per a treballar amb conceptes de teoria de codis q-aris, però només per a codis lineals. Per aquest motiu es va iniciar la creació en 2006 de la llibreria Q-ARY CODES [12] per treballar amb codis no lineals de forma eficient fent servir l'estructura kernel i representants dels cosets, desenvolupada pel Combinatorics, Coding and Security Group (CCSG) de la Universitat Autònoma de Barcelona (UAB). Actualment la llibreria està en versió 1.0, però es preveu llançar la versió 2.0 on estaran incorporades les funcions que s'han desenvolupat en aquest treball. Hi ha altres sistemes com GAP i SAGE que també permeten treballar amb codis lineals, però no són tan eficients ni complets com és MAGMA per treballar amb codis lineals. Tampoc en cap d'aquests sistemes es pot treballar amb codis no lineals de forma eficient.

L'objectiu principal es divideix en quatre parts. La primera part ha consistit en l'estudi de la teoria de codis q-aris, sobre cossos finits amb q elements. Per als codis lineals s'ha estudiat el funcionament de la codificació i la descodificació, i per a codis no lineals s'han demostrat resultats que permeten implementar les funcions citades de forma més eficient. La segona part ha consistit a aprendre el sistema

MAGMA i la seva llibreria de codis lineals, a més de la llibreria Q-ARY CODES. En la tercera part s'han implementat les funcions i diferents algoritmes per a cada funció, segons la teoria de la primera part. Finalment, s'han validat les funcions i s'han fet estudis de rendiment per determinar quin algoritme de cada funció és millor, fent servir codis q-aris de la llibreria.

En la Secció 2, sobre **codis q-aris**, explicarem que són els codis q-aris, quines propietats tenen i com s'interpreta els codis permutacionalment equivalents. En la part de codis lineals, explicarem les seves propietats bàsiques, la codificació i descodificació, i perquè funciona la correcció dels errors mitjançant la descodificació per síndromes. Per últim, en la part dels codis no lineals donarem una demostració diferent d'un teorema que ens permet representar un codi no lineal de forma compacta a partir de cosets/traslladats d'un subcodi lineal que anomenarem kernel. Aquesta estructura l'anomenarem Kernel/Coset i serà la base del nostre treball.

En la Secció 3, sobre **construcció de codis nous**, explicarem la suma directa i la suma Plotkin, veurem i donarem demostracions de les propietats de les sumes. Aquestes propietats ens serveixen per implementar les funcions a MAGMA. Definirem un nou operador per treballar amb subgrups del grup d'automorfismes dels codis.

En la Secció 4, sobre **implementacioins de funcions en MAGMA**, explicarem l'objecte `CodeFld` i la funció `QaryCode` inclòs en llibreria. Seguidament explicarem les funcions `DirectSum(C, D)`, `DirectSum(Q)`, `PloktinSum(C, D)` i `PadCode(C, n)` que hem desenvolupat i quins algoritmes hem considerat per a cada funció. Finalment, passarem a les funcions, tests de caixa negra per comprovar que estan ben implementades i després farem una prova de rendiment per veure quins algoritmes són més eficients, utilitzant codis q-aris disponibles dins de la llibreria.

2 Codis q-aris

Definició 1. *Sigui $\mathbb{F}_q$ un cos finit amb q elements. Diem que C és un codi q-ari $(n, |C|)_q$, si $C \subseteq \mathbb{F}_q^n$ és un conjunt finit de seqüencies o vectors de longitud n sobre $\mathbb{F}_q$. Els vectors de C s'anomenen paraules codi o codewords.*

A partir d'ara $\mathbb{F}_q$ serà un cos finit amb q elements. Un codi q-ari, si $q = 2$ es diu també codi binari i si $q = 3$ codi ternari.

Suposem que la informació a transmetre està representada com una seqüència d'elements d'un cos $\mathbb{F}_q$. Primer dividim la seqüència en blocs de k símbols de $\mathbb{F}_q$. A continuació fem la codificació que consisteix en enviar cada bloc de $\mathbb{F}_q^k$ a $\mathbb{F}_q^n$, on $n \geq k$, mitjançant una aplicació injectiva:

$$f : \mathbb{F}_q^k \longrightarrow \mathbb{F}_q^n.$$

Aleshores, $C = Im(f)$ és un codi q-ari $(n, |C| = q^k)_q$. Un cop es codifica el missatge es procedeix a transmetre'l pel canal escollit. A partir d'ara direm que C és un codi q-ari o simplement un codi.

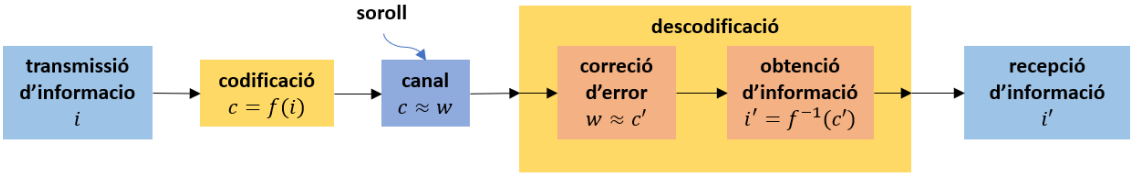

Figura 2: Procés de comunicació utilitzant teoria de codis

Sigui C un codi q-ari. Suposem que enviem la informació $i \in \mathbb{F}_q^k$, per tant enviem pel canal la paraula codi $c = f(i) \in C$ i rebem el vector $w \in \mathbb{F}_q^n$. La descodificació consisteix primer en trobar la paraula codi c' més "propera"a w, i a continuació obtenir la informació corresponent $i' = f^{-1}(c')$. En la Figura 2 podem veure un esquema d'aquest procés.

Exemple 1. *Suposem que volem transmetre la seqüencia* 1101 *binaria o de* $\mathbb{F}_2$*. El que podem fer és agafar* $k = 1$*,* $n = 3$ *i la funció:*

$$\begin{aligned} f : \mathbb{F}_2 &\longrightarrow \mathbb{F}_2^3 \\ 0 &\longmapsto (0,0,0) \\ 1 &\longmapsto (1,1,1). \end{aligned}$$

Observem que $C = Im(f) = \{(0,0,0),(1,1,1)\}$ *és un codi* 2*-ari o binari i la codificació de la seqüencia 1101 és la seqüencia de les paraules codi de* C*:* $(1,1,1),(1,1,1),(0,0,0),(1,1,1)$ *que és el que enviarem pel nostre canal. Una manera de descodificar seria seleccionar* 0 *o* 1 *segons el que més vegades aparegui en el vector rebut. Per exemple si rebem el vector* $(1,0,1)$*, el descodifiquem a* 1*.*

Com que $n \geq k$, això fa que la transmissió sigui més lenta perquè enlloc d'enviar una seqüencia de k elements enviem de n, però en canvi agafant una funció f adequada, guanyem que la transmissió sigui més fiable, ja que com més endavant veurem, la codificació i la descodificació serà ràpida i a més, ens permetrà corregir més errors.

A continuació definim la distància de Hamming que ens servirà entre altres coses per detectar i corregir errors.

Definició 2. *La distància de Hamming de* $u = (u_1 \ldots, u_n), v = (v_1, \ldots, v_n) \in \mathbb{F}_q^n$ *és*

$$d_H(v,v) = |\{i \in \{1,\ldots,n\} \mid u_i \neq v_i\}|.$$

La distància de Hamming compleix les propietats de distància topològica [6, Proposició 2].

Definició 3. *El pes de Hamming de* $v = (v_1, \ldots, v_n) \in \mathbb{F}_q^n$ *és*

$$wt_H(v) = |\{i \in \{1,\ldots,n\} \mid v_i \neq 0\}|.$$

Més endavant quan es parlin de distància i pes, serà de Hamming. Observem que $d_H(u,v) = wt_H(u-v)$ i $wt_H(v) = d_H(v,0)$.

Definició 4. *Sigui* C *un codi* q*-ari* $(n, |C|)_q$ *i sigui un vector* $e \in \mathbb{F}_q^n$*. Diem que* C *detecta l'error* e *si i només si* $c + e \notin C$ *per a qualsevol* $c \in C$*.*

Definició 5. *Sigui C un codi q-ari $(n, |C|)_q$ i sigui un vector $e \in \mathbb{F}_q^n$. Diem que C corregeix l'error e si i només si, per qualsevol paraula codi $c \in C$ tenim $d_H(c + e, c) \leq d_H(c + e, c')$, i.e $c + e$ està més proper a c que a qualsevol altra paraula codi $c' \in C \setminus \{c\}$.*

Definició 6. *Sigui C un codi q-ari. Diem que C és t-detector si C detecta tots els errors que tenen un pes menor o igual que t i no pot detectar com a mínim un de pes $t + 1$.*

Definició 7. *Sigui C un codi q-ari. Diem que C és t-corrector si C corregeix tots els errors que tenen un pes menor o igual que t i no pot corregir com a mínim un de pes $t + 1$.*

Definició 8. *Sigui C un codi q-ari $(n, |C|)_q$. Definim la distància mínima de C com*

$$d_H(C) = \min\{d_H(u, v) \mid u, v \in C, u \neq v\}.$$

Si tenim un codi q-ari C que tingui només una paraula codi, aleshores denotem la seva distància mínima $d_H(C) = 0$. Observem que si $|C| > 1$ aleshores $d_H(C) \geq 1$, o sigui, la distància mínima està fitada inferiorment per 1.

Teorema 1. *[6, Teoremes 1 i 2] Un codi q-ari C és $(d_H(C)-1)$-detector i $\lfloor (d_H(C)-1)/2 \rfloor$-corrector.*

Si d és la distància mínima d'un codi q-ari $(n, M)_q$, aleshores diem que és un codi $(n, M, d)_q$. La taxa de transmissió d'un codi $(n, M)_q$ és

$$R = \frac{\log_q(M)}{n},$$

i mesura la proporció d'informació que conté cada codi. Fixem-nos que R està fitat $0 < R \leq 1$. Aquesta taxa serveix per comparar diferents codis i mentre més proper a 1 millor. Ens interessaria obtenir un codi amb n menor i M i d majors, perquè si M és major ens permet enviar més codis i si d és major podem corregir més errors i també tindríem una taxa de transmissió més gran. Però hi ha una relació entre els paràmetres M, q, n i d, la *fita de Singleton*:

$$M \leq q^{n-d+1}$$

i la *fita de Hamming*:

$$M \sum_{i=0}^{t} \binom{n}{i} (q-1)^i \leq q^n,$$

on $t = \lfloor (d-1)/2 \rfloor$. Diem que un codi $(n, M, d)_q$ és òptim si no es pot incrementar M per valors de n, d i q fixats ni tampoc incrementar d per n, M, q fixats.

Exemple 2. *[14, Exemple 11] El codi $C_1 = \{(0,0,0), (1,1,1)\}$ és un codi binari $(3, 2, 3)_2$, per tant pot detectar fins a dos errors o corregir fins a un error, com ja ho intuíem de l'Exemple 1. La seva taxa de transmissió és $R = 1/3 \simeq 0.33$. El codi $C_2 = \{(0,0,0,0), (1,2,1,2), (2,1,2,1), (1,1,2,2), (2,2,1,1)\}$ és un codi 3-ari $(4, 5, 2)_3$ que permet detectar un error però no pot corregir cap i $R \simeq$*

0.36. *El codi* $C_3 = \{(0,0,0,0,0),(2,1,0,2,0),(1,2,0,1,0),(2,1,1,1,2),(1,2,2,2,1),(1,2,1,0,2),(2,1,2,0,1),(0,0,1,2,2),(0,0,2,1,1)\}$ *és un codi 3-ari* $(5,9,3)_3$ *amb la mateixa capacitat detectora i correctora que* C_1*, però amb millor taxa de transmissió* $R = 0.4$*.*

A partir d'ara denotarem S_n com el grup simètric de permutacions sobre el conjunt $\{1,2,\ldots,n\}$ i id el seu element neutre. Sigui $c = (c_1,\ldots,c_n) \in C$ una paraula codi d'un codi $(n,|C|)_q$, per $\pi \in S_n$ farem servir que

$$\pi(c) := (c_{\pi(1)},\ldots,c_{\pi(n)}) \text{ i } \pi(C) := \{\pi(c) \mid c \in C\}.$$

Definició 9. *Diem que dos codis* $(n,M)_q$*,* C *i* C' *són permutacionalment equivalents si existeix una permutació* $\pi \in S_n$*, tal que* $C' = \pi(C)$*.*

Considerem l'acció de S_n sobre $X = \mathcal{P}(\mathbb{F}_q^n)$, on X és el conjunt de les parts de $\mathbb{F}_q^n$, com

$$\begin{aligned} S_n \times X &\longrightarrow X \\ (\pi, C) &\longmapsto \pi \cdot C := \pi(C). \end{aligned}$$

Veièm que és una acció, per $C \in X$ i $\pi_1,\pi_2 \in S_n$, tenim que $id \cdot C = id(C) = C$ i $(\pi_1 \circ \pi_2)\cdot C = (\pi_1 \circ \pi_2)(C) = \pi_1(\pi_2(C)) = \pi \cdot (\pi \cdot C)$. Observem que $C \in X$ és un codi $(n,|C|)_q$ i les orbites de C són

$$\mathcal{O}_C = S_n(C) := \{\pi(C) \mid \pi \in S_n\}.$$

Per $C, C' \in X$ la relació $C \sim_o C'$ si i només si existeix $\pi \in S_n$ tal que $C = \pi(C')$, és una relació d'equivalència. Observem que $C \sim_o C'$ si i només si C i C' són permutacionalment equivalents, és a dir els codis permutacionalements equivalents pertanyen a la mateixa orbita. El conjunt quocient és $X/\sim_o = X/S_n$ i els seus elements són $\bar{C} = S_n(C)$ les orbites de $C \in X$, per tant X/S_n és el conjunt d'òrbites. A partir d'ara si es parla de classes de codis q-aris es farà referencia a classes d'equivalència de $\sim_o$. Com que S_n és finit tenim

$$|\mathcal{O}_C| = [S_n : (S_n)_C] = |S_n|/|(S_n)_C| = n!/|(S_n)_C|,$$

on $(S_n)_C := \{\pi \in S_n | \pi(C) = C\}$ és l'estabilitzador de C. L'estabilitzador també s'anomena **grup d'automorfismes** de C, i és un subgrup de S_n. A partir d'ara utilitzarem la notació $PAut(C)$ per denotar $(S_n)_C$.

Exemple 3. *Diem que un codi* C *és cíclic si existeix la permutació* $\sigma = (1,2,\ldots,n) \in (S_n)_C$*, i com l'ordre de la permutació és* $o(\sigma) = n$ *tenim que* $n \mid |(S_n)_C|$*. Per tant, si trobem un codi* C *tal que* $n \nmid |(S_n)_C|$*,* C *no serà cíclic.*

Proposició 1. *Siguin* C *i* C' *codis* q*-aris* $(n,M)_q$*, permutacionalment equivalents, aleshores tenen la mateixa distància mínima, o sigui* $d_H(C) = d_H(C')$*. La distància mínima és una propietat de classe.*

Demostració. Per definició de distància de Hamming tenim que $d_H(u,v) = d_H(\pi(u), \pi(v))$, per qualsevol $\pi \in S_n$. Suposem que $C = \pi(C')$ i siguin $u, v \in C$ tal que

$d_H(u,v) = d := d_H(C)$. Com $C = \pi(C')$, existeixen $u', v' \in C'$ tal que $u = \pi(u')$ i $v = \pi(v')$ aleshores $d = d_H(u,v) = d_H(\pi(u'), \pi(v')) = d_H(u', v') \leq d' := d_H(C')$. Anàlogament, agafant $C' = \pi^{-1}(C)$, es veu que $d' \leq d$ i per tant $d = d'$. □

Exemple 4. *El reciproc de la proposició anterior no és cert, com es pot veure en el següent contraexemple. Considerem els següents codis* $(3, 3^2)_3$:

$$C_2 = \langle (1,0,0), (0,1,1) \rangle \text{ i } C_3 = \langle (1,0,2), (0,1,0) \rangle$$

com a $\mathbb{F}_3$ *espais vectorials a* $\mathbb{F}_3^3$. *Com que la distància mínima està fitada inferiorment per 1 i* $(1,0,0), (0,0,0) \in C_2$ *i* $(0,1,0), (0,0,0) \in C_3$, *tenim que* $d_H((1,0,0), (0,0,0)) = d_H((0,1,0), (0,0,0)) = 1$ *i els dos codis tenen distància mínima 1. Veiem que no són permutacionalment equivalents. Tenim que* $v = (1,1,1) \in C_2$, *però* $v \notin C_3$, *ja que en el cas contrari existirien* $\alpha, \beta \in \mathbb{F}_3$ *tals que* $\alpha(1,0,2) + \beta(0,1,0) = (1,1,1) \Leftrightarrow \alpha = 1$ *i* $2\alpha = 1$ *que no pot ser.*

Hem vist que els codis $(n, M)_q$ permutacionalment equivalents són de la mateixa classe d'equivalència $\sim_o$ i que la distància mínima és una propietat de classe, però no és suficient per classificar tots els codis $(n, M)_q$, ja que com hem vist amb el contraexemple anterior de l'Exemple 4, hi ha codis $(n, M)_q$ que tenen la mateixa distància mínima, però no pertanyen a la mateixa classe.

Tots els codis permutacionalment equivalents tenen la mateixa taxa de transmissió i la mateixa distància mínima, per tant tenen les mateixes capacitats correctores i detectores. Aquests resultats són útils alhora de descodificar, ja que donat un codi q-ari podem trobar un codi equivalent permutacionalment amb el qual la descodificació serà més eficient computacionalment.

2.1 Codis lineals

Els codis q-aris lineals són els que més es fan servir en les aplicacions pràctiques, ja que com veurem més endavant, són més eficaços en codificació i descodificació.

Definició 10. *Un codi* q*-ari* $C \subseteq \mathbb{F}_q^n$ *és lineal si és un subespai vectorial de* $\mathbb{F}_q^n$.

Exemple 5. *El codi binari* $C_1 = \{(0,0,0), (1,1,1)\}$ *de l'Exemple 1 és un codi lineal. Els codis ternaris* $C_2 = \langle (1,0,0), (0,1,1) \rangle$ *i* $C_3 = \langle (1,0,2), (0,1,0) \rangle$ *de l'Exemple 4, són codis lineals perquè són generats pels vectors* $\{(1,0,0), (0,1,1)\}$ *i* $\{(1,0,2), (0,1,0)\}$ *de* $\mathbb{F}_3^3$, *respectivament.*

Donat un codi lineal, observem que si $k = \dim(C)$ aleshores $|C| = q^k$ i la taxa de transmissió és $R = k/n$. A partir d'ara si coneixem $d = d_H(C)$ direm que és un codi lineal $[n, k, d]_q$ i sinó un codi $[n, k]_q$.

Definició 11. *Sigui* C *un codi lineal. Definim el seu pes mínim com*

$$wt_H(C) = \min\{wt_H(c) \mid c \in C \text{ i } c \neq 0\}.$$

Proposició 2. *[6, Proposició 7] Si* C *és un codi lineal, aleshores* $d_H(C) = wt_H(C)$.

Farem servir la notació $M_{n\times m}(K)$, per denotar el conjunt de totes les matrius de n files i m columnes, sobre un cos K. I també farem servir la notació G^- per denotar la pseudoinversa de la matriu G. Finalment, direm que M^t és la matriu transposada de la matriu M.

Definició 12. *Sigui C un codi lineal $[n,k]_q$. Diem que $G \in M_{k\times n}(\mathbb{F}_q)$ és una matriu generadora de C si*

$$G = M(\mathcal{B},\mathcal{C})^t$$

és la transposada de la matriu de canvi de base de $\mathcal{C}$ a $\mathcal{B}$, on $\mathcal{B}$ és una base de C i $\mathcal{C}$ és la base canònica de $\mathbb{F}_q^n$. Les files de la matriu G estan formades pels vectors de la base $\mathcal{B}$.

Exemple 6. *El codi binari $C_1 = \{(0,0,0),(1,1,1)\}$ de l'Exemple 1 té la matriu generadora $G_1 = (1 \quad 1 \quad 1)$, i els codis ternaris de l'Exemple 4, $C_2 = \langle(1,0,0),(0,1,1)\rangle$ i $C_3 = \langle(1,0,2),(0,1,0)\rangle$, tenen matriu generadora*

$$G_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \end{pmatrix} \text{ i } G_3 = \begin{pmatrix} 1 & 0 & 2 \\ 0 & 1 & 0 \end{pmatrix},$$

respectivament.

Ara veurem com es fa la codificació per a codis lineals. Donat C un codi $[n,k]_q$, una codificació és l'aplicació lineal:

$$\begin{aligned} f : \mathbb{F}_q^k &\longrightarrow \mathbb{F}_q^n \\ i &\longmapsto iG \end{aligned}$$

on $G = M(\mathcal{B},\mathcal{C})^t$ és una matriu generadora de C, on $\mathcal{B}$ és una base de C i $\mathcal{C}$ és la base canònica de $\mathbb{F}_q^n$. Observem que $i = C(f(i),\mathcal{B})$ és la coordenada de $f(i)$ respecte la base $\mathcal{B}$. Una manera simple de descodificar $w \in \mathbb{F}_q^n$ seria fent servir la pseudoinversa de G, que seria $wG^- = wG^t(GG^t)^{-1}$, ja que G té rang màxim.

Definició 13. *Una matriu generadora G d'un codi $[n,k]_q$ C conté la matriu identitat I_k de mida k com a submatriu, aleshores diem que G és sistemàtic o que la codificació és sistemàtica en les coordenades corresponents a les posicions de les columnes de I_k en G.*

Definició 14. *Diem que una matriu generadora G d'un codi $[n,k]_q$ està en forma estàndard si*

$$G = (I_k|A),$$

on $I_k \in M_{k\times k}(\mathbb{F}_q)$ és la matriu identitat i $A \in M_{k\times(n-k)}(\mathbb{F}_q)$.

Proposició 3. *[6, Proposició 9] Sigui C un codi lineal $[n,k]_q$. Aleshores existeix un codi C' lineal $[n,k]_q$ permutacionalment equivalent a C tal que la seva matriu generadora G' està en forma estàndard.*

Donada una matriu G, podem utilitzar el mètode de reducció de Gauss per calcular una matriu sistemàtica. A continuació permutant columnes podem obtenir G' en forma estàndard.

Exemple 7. *Considerem el codi binari de l'Exemple 1,* $C_1 = \{(0,0,0),(1,1,1)\}$ *que té matriu generadora* $G_1 = (1 \quad 1 \quad 1)$*. Com havíem vist a l'exemple si volem codificar la seqüencia* 1011 *utilitzàvem la funció* f*, però observem que* f *és lineal, i la codificació és* $f(i) = iG_1 = i(1 \quad 1 \quad 1)$*.*

Exemple 8. *Considerem ara un codi ternari* C *amb una matriu generadora*

$$G = \begin{pmatrix} 1 & 2 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

i volem codificar la seqüencia 122210*. Primer dividim la seqüencia en blocs de* $k = 2$ *elements. A continuació multipliquem cada bloc i per la matriu generadora,* iG*, com podem veure en la segona columna de la Taula 1. Per descodificar suposem que no hi hagut cap error, calculem la pseudoinversa de* G*,*

$$G^- = G^t(GG^t)^{-1} = \begin{pmatrix} 2 & 0 \\ 1 & 0 \\ 2 & 1 \end{pmatrix},$$

i multipliquem el vector que hem rebut per la pseudoinversa, com veiem en la tercera columna de la Taula 1.

$i \in \mathbb{F}_3^2$	$c = iG \in \mathbb{F}_3^3$	$i = c \cdot G^- \in \mathbb{F}_3^2$
$(1,2)$	$(0,0,2)$	$(1,2)$
$(2,2)$	$(1,2,2)$	$(2,2)$
$(1,0)$	$(1,2,0)$	$(1,0)$

Taula 1: Codificació i descodificació

Aquesta descodificació és lenta perquè hem de multiplicar per la pseudoinversa. Ara farem servir un codi C' *permutacionalment equivalent del codi* C *i veurem que la descodificació és sistemàtica, Si per exemple intercanviem les columnes* 2 *i* 3 *de* G *tenim* $\begin{pmatrix} 1 & 0 & 2 \\ 1 & 1 & 2 \end{pmatrix}$ *que és una matriu generadora del codi* C' *permutacionalment equivalen a* C*, si ho diagonalitzem tenim que* $G' = \begin{pmatrix} 1 & 0 & 2 \\ 0 & 1 & 0 \end{pmatrix}$ *està en forma estàndard, si volem codificar* $(1,2),(2,2),(1,0)$ *es fa com abans. Multipliquem els elements de la seqüencia per la matriu* G' *i tenim* $(1,2,2),(2,2,2),(1,0,2)$ *i la descodificació és sistemàtica en les dues primeres coordenades, ens estalviem multiplicar per la pseudoinversa de* G' *i a més* C' *manté totes les propietats de* C*.*

Donat un codi lineal C podem trobar un codi C' permutacionalment equivalent de manera que la codificació en les primeres coordenades és sistemàtica, i això ens evita fer servir la pseudoinversa de G que té un cost computacionalment més gran.

Definició 15. *El producte escalar estàndard de* K^n*, on* K *és un cos, és l'aplicació*

$$\langle -|- \rangle : K^n \times K^n \longrightarrow K^n$$

definida per $\langle (u_1,\ldots,n_n)|(v_1,\ldots,v_n)\rangle = u_1v_1 + \cdots + u_nv_n$*.*

Definició 16. *Sigui C un codi $[n,k]_q$. L'ortogonal o dual de C és $C^\perp = \{v \mid \langle c \mid v\rangle = 0, \forall c \in C\}$.*

Proposició 4. *[6, Proposició 10] Sigui C un codi $[n,k]_q$. L'ortogonal de C és un codi $[n, n-k]_q$.*

Definició 17. *Sigui C un codi $[n,k]_q$. Diem que H es una matriu de control de C, si H és una matriu generadora de l'ortogonal de C.*

Observem que la matriu generadora d'un codi lineal i la seva matriu de control són ortogonals per definició.

Proposició 5. *[6, Proposició 11] Sigui C un codi $[n,k]_q$ amb una matriu generadora en la forma estàndard $G = (I_k|A)$. Aleshores la matriu de control de de C és*

$$H = (-A^t|I_{n-k}).$$

Proposició 6. *[6, Proposicició 14] La distància mínima d'un codi lineal C coincideix amb el mínim nombre de columnes linealment independents d'una matriu de control de C.*

En l'Exemple 8 hem vist com es codifica i descodifica la informació suposant que no es produeix errors en la transmissió. Però això no passa en la realitat, pot haver errors en la transmissió, ja que els canals tenen soroll. Veurem com podem aplicar teoria de codis per detectar i corregir aquests errors de manera eficaç.

Sigui C un codi $[n,k]_q$. Considerem l'espai vectorial quocient $\mathbb{F}_q^n/C$ que ve de la relació d'equivalència $u \sim v \in \mathbb{F}_q^n$ si i només si $u - v \in C$, ja que C és un subespai vectorial. Tenim que $|\mathbb{F}_q^n/C| = q^{n-k}$ i denotarem per C_i, per $i = 0, 1, \ldots, q^{n-k}-1$, les q^{n-k} classes d'equivalència.

Definició 18. *Sigui C un codi $[n,k]_q$. Les classes d'equivalència s'anomenen també cosets. Un element $e_i \in C_i$ tal que tingui pes mínim direm que és el líder del coset C_i o més breument coset líder.*

Proposició 7. *[6, Proposició 16] Sigui C un codi q-ari lineal t-corrector i e el líder d'un coset C_i per $i = 0, 1 \ldots, q^{n-k}-1$, aleshores:*

i) Si $wt_H(e) \leq t$, e és únic.

ii) Si $wt_H(e) > t$, e pot no ser únic.

Definició 19. *El síndrome de $w \in \mathbb{F}_q^n$, respecte a una matriu $H \in M_{(n-k)\times n}(\mathbb{F}_q)$, és $wH^t \in \mathbb{F}_q^{n-k}$.*

Sigui C un codi $[n,k]_q$ i H una matriu de control de C. Denotem fent abús de notació l'aplicació lineal:

$$\begin{aligned} H : \mathbb{F}_q^n &\longrightarrow \mathbb{F}_q^{n-k} \\ w &\longmapsto H(w) = wH^t. \end{aligned}$$

Observem que $Ker(H) = C$, ja que H és ortogonal a la matriu generadora de C. Pel teorema d'isomorfisme, $\bar{H} : \mathbb{F}_q^n/C \longrightarrow Im(H) \subseteq \mathbb{F}_q^{n-k}$ és una bijecció i com

que $q^{n-k} = |\mathbb{F}_q^n/C| = |Im(H)|$, tenim que $Im(H) = \mathbb{F}_q^{n-k}$. Així, hem demostrat la següent proposició.

Proposició 8. *Sigui C un codi lineal $[n,k]_q$ i H una matriu de control de C, aleshores:*

i) $w \in C \Leftrightarrow H(w) = 0$.

ii) *Per* $v, w \in \mathbb{F}_q^n, H(v) = H(w) \Leftrightarrow v - w \in C$.

iii) *Cada vector de* $\mathbb{F}_q^{n-k}$ *és un síndrome.*

Ara ens centrarem en la descodificació via síndrome. Sigui $L = \{0, e_1, \dots e_r, e_{r+1}, \dots, e_{q^{n-k}-1}\}$ el conjunt de tots els líders dels cosets de C, fixats de tal manera que $wt_H(e_i) \leq t$ per $i = 0, 1, \dots, r$ i $wt_H(e_i) > t$ per $i = r+1, \dots, e_{q^{n-k}-1}$. Amb les notacions anteriors demostrem el següent lema:

Lema 1. *L'aplicació $H|_L : L \longrightarrow \mathbb{F}_q^{n-k}$ és bijectiva.*

Demostració. Com que $|L| = |\mathbb{F}_q^{n-k}|$, només hem de veure que $H|_L$ és exhaustiva. Sigui $h \in \mathbb{F}_q^{n-k}$. Com que H és exhaustiu, existeix $w \in \mathbb{F}_q^n$ tal que $H(w) = h$, aleshores $\bar{w} = w + C$ té un coset líder e que pertany a L i per tant $H(e) = H(w) = h$. □

Definició 20. *La taula de síndromes és el conjunt*

$$TS(H|_L) = \{(e, H|_L(e)) \in L \times \mathbb{F}_q^{n-k} \mid e \in L\}.$$

Definició 21. *Sigui C un codi $(n, |C|)_q$ i $w \in \mathbb{F}_q^n$ un vector. Aleshores la distància mínima de w al codi C és $d_H(w, C) = \min\{d_H(w, c) \mid c \in C\}$.*

Hi ha dues maneres de descodificació, la descodificació **completa** i la **incompleta**. En la descodificació incompleta només es descodifica qualsevol vector rebut w si aquest té distància mínima a C, $d_H(w, C)$, menor o igual que la capacitat correctora t de C. Això es fa per assegurar que no hi hagi error si es fa la descodificació, ja que si rebem w serà de la forma $w = c + e$, per $c \in C$ i l'error és $e \in \mathbb{F}_q^n \setminus C$ i $d_H(w, C) = wt_H(e) > t$ i com C és t-corrector, la descodificació pot no corregir l'error e. Mentre que en la descodificació completa, es busca la paraula codi de C més propera a w. Si n'hi ha més d'una, s'assigna una a l'atzar.

En la descodificació per síndrome tenim descodificació completa i incompleta. En la descodificació completa, suposem que hem enviat $c \in C$ i rebem $w = c + e$. Llavors la descodificació és $c' = w - e' \in C$ on $e' = (H|_L)^{-1}(H(w))$. Si no hi ha error, $w \in C$ i $H(w) = 0$ llavors $e' = 0$ i $c' = w$. En la descodificació incompleta només es descodifica w si $wt_H(e') \leq t$, on C és t-corrector, sinó es demana retransmetre el vector enviat.

Per què la descodificació per síndrome funciona? Suposem que enviem c, rebem w, i que hi ha hagut algun error, o sigui que $w \notin C$, $w = c + e$. La idea de la descodificació és trobar $c' \in C$ tal que $d_H(c', w) = \min\{d_H(a, w) \mid a \in C\} \Leftrightarrow wt_H(e') = \min\{wt_H(s) \mid s \in C + w\}$, on $e' = w - c'$. Només podem assegurar que

la descodificació funciona si C és un codi t-corrector, si $w = c + e$ amb $wt_H(e) \leq t$. Suposem $wt_H(e) \leq t$, com que $H(w) = H(e)$ tenim que $\bar{w} = \bar{e} = C + e$. Sigui $e_i = (H|_L)^{-1}(H(w))$, aleshores e_i és el coset líder de la classe $\bar{e_i} = \bar{e}$ i per la Proposició 7 com que $wt_H(e) \leq t$, la classe $\bar{e}$ té un únic coset líder i per tant $e = e_i$. En canvi, si $wt_H(e) \geq t + 1$, la classe $C + e$ pot no tenir un líder únic i per tant, la descodificació pot no funcionar.

líder e on $wt_H(e) \leq 1$	síndrome	líder e on $wt_H(e) > 1$	síndrome
00000	000	12000	112
10000	120	21000	221
20000	210	10002 *	122
01000	022	01100	111
00200	011	02200	222
00200	200	10001	121
00010	010	20002	212
00020	020	22000*	202
00001	001	20002	212
00002	002	22000*	202
		11000*	101
		00102	102
		00201	201
		20200*	110
		10100*	220
		02020*	012
		01010*	021

Taula 2: Taula de síndromes per l'Exemple 9, on "*"vol dir que els líders no són únics

Exemple 9. *[6, Exemple 50] Sigui C un codi $[5, 2, 3]_3$ amb una matriu generadora*

$$G = \begin{pmatrix} 1 & 0 & 2 & 1 & 0 \\ 0 & 1 & 0 & 2 & 2 \end{pmatrix},$$

que està en forma estàndard. Per la Proposició 5, tenim que la matriu de control és

$$H = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 2 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{pmatrix}.$$

La capacitat correctora del codi és $t = 1$. Calculem la taula de síndromes, donada en la Taula 2. Suposem que volem transmetre la seqüencia d'informació $(1, 0), (2, 1), (2, 2), (1, 1)$. La codificació fent servir la matriu generadora G és $(1, 0, 2, 1, 0), (2, 1, 1, 2), (2, 2, 1, 0, 1), (1, 1, 2, 0, 2)$ i suposem que després d'enviar-lo pel canal rebem $(2, 0, 2, 1, 0), (2, 0, 0, 1, 2), (2, 2, 2, 1, 1), (0, 1, 0, 0, 2)$. Apliquem la descodificació completa per síndrome. En la Taula 3 es mostra detalladament el procediment per corregir els errors, podem veure que no es corregeixen tots.

c	e	$w = c + e$	síndrome	líder e'	$c' = w - e'$
10210	1000	20210	120	10000	10210
21112	02200	20012	222	02200	21112
22101	00110	22211	110	20200	02011
11202	20100	01002	010	00010	01022

Taula 3: Descodificació

En la última columna de la Taula 3 tenim la seqüencia 10210 21112 02011 01022 *corregida. Per recuperar la informació, com que la matriu G està en forma estàndard, la descodificació és sistemàtica en les dues primeres coordenades, per tant obtenim* 10 21 02 01.

La descodificació per síndrome és eficaç sempre i quan $k > n/2$, ja que requereix una taula de q^{n-k} entrades comparat amb una taula de q^n vectors mostrant quina paraula codi és la més propera a cada vector.

Per a més informació sobre codis lineals es poden consultar per exemple els següents llibres [8, 9].

2.2 Codis no lineals

Definició 22. *Siguin C i C' dos codis q-aris $(n, M)_q$. Direm que són equivalents si existeix una permutació $\pi \in S_n$ tal que $C = a + \pi(C') := \{a + c' \mid c' \in \pi(C')\}$ per algun $a \in \mathbb{F}_q^n$. Observem que és el mateix que $C - a \sim_o C'$.*

Dos codis equivalents C i C' tenen la mateixa distància de Hamming, ja que $C-a \sim_o C' \Rightarrow d_H(C-a) = d_H(C')$ i per definició de distància $d_H(C-a) = d_H(C)$, per tant $d_H(C) = d_H(C')$. Els codis equivalents tenen els mateixos paràmetres $(n, M, d)_q$ i R, per tant, si tenim un codi C tal que $0 \notin C$, considerant el nou codi $C' = -c+C$, per $c \in C$, C i C' són equivalents i $0 \in C'$. Doncs a partir d'ara considerem que el zero pertany als codis q-aris no lineals.

Definició 23. *[2] El kernel d'un codi C q-ari és*

$$K_C = \{x \in C \mid \lambda x + C = C, \forall \lambda \in \mathbb{F}_q\}.$$

Observem que és el mateix que $K_C = \{x \in C \mid \langle x \rangle + C \subseteq C\}$. Es veu fàcilment que és lineal. Siguin $v, u \in K_C$ i $\lambda \in \mathbb{F}_q$, com $0 \in C$, $\lambda u \in K_C \Leftrightarrow \langle \lambda u \rangle + C = \langle u \rangle + C \subseteq C$ i $u + v \in K_C \Leftrightarrow \langle u + v \rangle + C \subseteq C \Leftrightarrow \alpha u + \alpha v + c = \alpha u + c' \in C$, on $c' = av + c \in C$. A més K_C és un espai vectorial dins de C amb dimensió maximal [15], però aquest resultat no el demostrem.

Definició 24. *Siguin C un codi q-ari i K_C el seu kernel. Diem que $K' \subseteq K_C$ és un kernel parcial de C si K' és un subespai vectorial de K_C.*

Teorema 2. *[2, 11] Sigui K_C el kernel del codi q-ari $C \subseteq \mathbb{F}_q^n$, aleshores*

$$C = \bigsqcup_{i=0}^{t} (K_C + v_i),$$

on $v_0 = 0, t+1 = |C|/q^\kappa$, $\kappa = \dim(K_C)$, *i* $v_1, \ldots, v_t \in C$.

Demostració. Considerem l'acció de K_C sobre C donada per:

$$\begin{array}{rl} K_C \times C & \longrightarrow C \\ (k, c) & \longmapsto k + c. \end{array}$$

Aquesta acció està ben definida per la definició de kernel i és fàcil veure que és una acció, ja que per $c \in C$ i $k_1, k_2 \in K_C$ tenim $0 + c = c$ i $(k_1 + k_2) + c = k_1 + (k_2 + c)$. Les orbites de $c \in C$ són

$$\mathcal{O}_c = K_C + c.$$

Per $c, c' \in C$, $c \sim_o c' \Leftrightarrow \exists k \in K_C$ tal que $c = c' + k$, és una relació d'equivalència i la classe de c és la seva orbita $K_C + c$. L'estabilitzador de c és

$$(K_C)_c = \{k \in K_C | k + c = c\} = \{0\}.$$

Com que C és finit llavors el conjunt quocient de la relació d'equivalència està formada per diferents orbites $C/\sim_o = C/K_C = \{\mathcal{O}_{v_0}, \ldots, \mathcal{O}_{v_t}\}$ on $t \geq 0$ i $v_i \in C$ són de diferents classes per $i = 0, 1, \ldots, t$ amb $v_0 = 0$. A més, $C = \mathcal{O}_{v_0} \sqcup \cdots \sqcup \mathcal{O}_{v_t}$ i el t ve de que com $|\mathcal{O}_u| = |K_C|/|(K_C)_u| = |K_C|$. Per tant,

$$|C| = (t+1)|K_C|$$

i tenim $t + 1 = |C|/q^\kappa$. □

Observem que el teorema anterior també és vàlid per un kernel parcial en lloc d'un kernel. A partir d'ara anomenarem els vectors $v_0, v_1, \ldots, v_t$ com a **representants dels cosets**.

El teorema anterior ens permet representar un codi q-ari C de manera compacta a partir d'una matriu generadora G del kernel de C, K_C, i $L = \{v_1, \ldots, v_t\}$ que és el conjunt format per un representant de cadascun dels cosets del kernel K_C, menys el vector zero, que representa el coset K_C.

El subespai vectorial de C que permet representar C de forma més compacta possible, o sigui, amb el mínim nombre d'elements a L, és precisament el kernel de C, K_C. Si coneixem un kernel parcial d'un codi C, és possible construir K_C de manera més eficient que només amb el codi C [15].

Exemple 10. *[10, Exemple 2.8] Sigui el codi ternari* $(5, 9)_3$ $C = \{(0,0,0,0,0), (2,2,2,2,0), (0,0,2,1,0), (1,0,2,2,1), (2,2,1,0,0), (0,0,1,2,0), (1,0,1,0,1), (1,0,0,1,1)\}$. *El seu kernel* K_C *té dimensió* 1 *i la seva matriu generadora és* $G = (0 \quad 0 \quad 1 \quad 2 \quad 0)$. *I els representats dels cosets són* $L = \{(0,0,0,0,0), (2,2,2,2,0), (1,0,2,2,1)\}$. *Les particions del codi són:* $K_C + (0,0,0,0,0) = \{(0,0,0,0,0), (0,0,2,1,0), (0,0,1,2,0)\}$, $K_C + (2,2,2,2,0) = \{(2,2,2,2,0), (2,2,0,1,0), (2,2,1,0,0)\}$ *i* $K_C + (1,0,2,2,1) = \{(1,0,2,2,1), (1,0,1,0,1), (1,0,0,1,1)\}$.

3 Construcció de codis nous

En aquesta secció veurem com podem construir codis nous a partir de codis existents i veurem algunes de les seves propietats. Aquestes construccions són les que s'implementaran per a la llibreria Q-ARY CODES.

3.1 Suma directa

Definició 25. *Siguin C_1 i C_2 dos codis q-aris $(n_1, |C_1|)_q$ i $(n_2, |C_2|)_q$. La suma directa de C_1 amb C_2 és*

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

No s'ha de confondre la suma directa de codis q-aris amb la suma directa d'espais vectorials encara que els codis siguin lineals, per això denotarem per $\oplus_v$ la suma directa d'espais vectorials.

Definició 26. *Diem que el mínim positiu d'un conjunt $l \subseteq \mathbb{R}$ és $\min_+(l)$, l'element positiu més petit del conjunt l i si l no en té cap, aleshores $\min_+(l) = 0$.*

Per exemple $\min_+(\{0,1,2\}) = 1$ i $\min_+(\{0,0,0\}) = 0$.

Lema 2. *[15] Siguin C_1 i C_2 dos codis q-aris $(n_1, |C_1|, d_1)_q$ i $(n_2, |C_2|, d_2)_q$. La suma directa $C_1 \oplus C_2$ és un codi q-ari $(n_1 + n_2, |C_1||C_2|, \min_+\{d_1, d_2\})_q$. Si a més C_1 i C_2 són lineals amb dimensió k_1 i k_2, aleshores $C_1 \oplus C_2$ és lineal i té dimensió $k_1 + k_2$.*

Demostració. Observem que $|C_1 \oplus C_2| = |C_1 \times C_2| = |C_1||C_2|$. Calculem la distància mínima $d = d_H(C_1 \oplus C_2)$. Siguin $c = (c_1|c_2), c' = (c_1'|c_2') \in C_1 \oplus C_2$. Llavors $d_H(c, c') = d_H(c_1, c_1') + d_H(c_2, c_2') \geq d_1 + d_2$. Suposem que $|C_i| = 1$ per algun $i \in \{1, 2\}$. Sense pèrdua de generalitat, podem suposar que $i = 1$. Aleshores tenim que

- si $|C_2| > 1$, per $c \neq c'$ tenim $d_H(c, c') = d_H(c_2, c_2') \geq d_2 \Rightarrow d = d_2$,
- si $|C_2| = 1$, per definició $d = 0$.

En els dos casos tenim que $d = \min_+\{d_1, d_2\}$. Suposem el cas contrari, és a dir, $|C_1| > 1$ i $|C_2| > 1$. Sigui $c \neq c'$, si $c_1 = c_1'$, aleshores $d_H(c, c') = d_H(c_2, c_2') \geq d_2$ i si $c_2 = c_2'$ tenim que $d_H(c, c') \geq d_2$, per tant $d = \min\{d_1, d_2\} = \min_+\{d_1, d_2\}$.

Ara suposem que C_1 i C_2 són lineals. Per tant, $C_1 \times 0_2$, $0_1 \times C_2 \subseteq \mathbb{F}_q^{n_1+n_2}$ són subespais vectorials, on 0_i és el vector zero de longitud n_i, $i \in \{1, 2\}$. Com que $(C_1 \times 0_2) \cap (0_1 \times C_2) = \{0\}$ llavors $(C_1 \times 0_2) \oplus_v (C_1 \times 0_2)$ estan en suma directa, notem que és el mateix que $C_1 \oplus C_2$ i per tant és lineal. Per la fórmula de Grassmann, tenim

$$\dim((C_1 \times 0_2) \oplus_v (0_1 \times C_2)) = \dim(C_1 \times 0_2) + \dim(0_1 \times C_2) = k_1 + k_2.$$

□

Observem que $C_1 \oplus C_2$ és lineal si i només si C_1 i C_2 són lineals.

Exemple 11. *Siguin* $C_1 = \{(0,0),(1,1),(2,2)\}$ *i* $C_2 = \{(0,0),(0,1)\}$ *dos codis ternaris,* C_1 *és lineal però* C_2 *no ho és perquè el vector* $2(0,1)$ *no hi pertany. La suma directa és* $C_1 \oplus C_2 = \{(0,0|0,0),(0,0|0,1),(1,1|0,0),(1,1|0,1),(2,2|0,0),(2,2|0,1)\}$*, té cardinal* $|C_1 \oplus C_2| = |C_1||C_2| = 6$*,* $n = 4$ *i la distància mínima és* $d = \min_+\{2,1\} = 1$*, per tant és un codi* $(4,6,1)_3$*, no és lineal perquè el vector* $2(0,0|0,1)$ *no hi pertany.*

En la suma directa es fa servir el $\min_+$ en lloc de min per calcular distàncies mínimes perquè tal com hem vist en la demostració del lema anterior, el $\min_+$ és vàlid per tots els codis, però min no ho és si en la suma directa un codi té el cardinal igual a u i l'altre no.

Exemple 12. *Sigui* $C_1 = \{(0,0,0),(1,1,1)\}$ *un codi binari que té distància mínima* 3 *i* $C_2 = \{(0,0,0)\}$ *el codi binari zero que té distància mínima* 0 *per definició. Aleshores el codi* $C_1 \oplus C_2 = \{(0,0,0|0,0,0),(1,1,1|0,0,0)\}$ *té distància mínima 3 que coincideix amb* $\min_+(3,0) = 3$*, mentre que si es fa servir el mínim* $\min(3,0) = 0$ *no és cert. En canvi, si considerem el codi* $C_2 \oplus C_2 = \{(0,0,0|0,0,0)\}$*, veiem té distància mínima* 0 *i* $\min_+(0,0) = \min(0,0) = 0$ *coincideixen.*

Proposició 9. *Siguin* C_1 *i* C_2 *codis q-aris lineals* $[n_1,k_1]_q$ *i* $[n_2,k_1]_q$*, amb matrius generadores* G_1 *i* G_2 *i matrius de control* H_1 *i* H_2*, respectivament. Aleshores la matriu generadora i la matriu de control de la suma directa* $C_1 \oplus C_2$ *són*

$$G_1 \oplus G_2 = \begin{pmatrix} G_1 & 0 \\ 0 & G_2 \end{pmatrix} \text{ i } H_1 \oplus H_2 = \begin{pmatrix} H_1 & 0 \\ 0 & H_2 \end{pmatrix},$$

respectivament.

Demostració. Com que $C_1 \oplus C_2 = (C_1 \times 0_2) \oplus_v (0_1 \times C_2)$ on 0_i és el vector zero de longitud n_i per $i \in \{1,2\}$, és una suma directa d'espais vectorials, la matriu generadora de l'espai $C_1 \times 0_2$ és $(G_1 \quad 0)$ i de $0_1 \times C_2$ és $(0 \quad G_2)$ i com estan en suma directa vectorial, la matriu generadora de $C_1 \oplus C_2$ serà $G_1 \oplus G_2$. També tenim que $H_1 \oplus H_2$ és una matriu de control de $C_1 \oplus C_2$, ja que $(G_1 \oplus G_2)(H_1 \oplus H_2)^t = 0$. □

Teorema 3. *[15] Siguin* C_1 *i* C_2 *dos codis q-aris* $(n_1,|C_1|)_q$ *i* $(n_2,|C_2|)_q$ *amb els kernels* K_{C_1} *i* K_{C_2}*, i els seus representants dels cosets* $L_1 = \{v_0,\ldots,v_{t_1}\}$ *i* $L_2 = \{w_0,\ldots,w_{t_2}\}$*, respectivament. Aleshores el kernel de la suma directa* $C_1 \oplus C_2$ *és* $K_{C_1 \oplus C_2} = K_{C_1} \oplus K_{C_2}$ *i els seus representants dels cosets són* $L_\oplus = L_1 \oplus L_2$*.*

Demostració. Provem que $K_{C_1 \oplus C_2} = K_{C_1} \oplus K_{C_2}$. Siguin $c = (c_1|c_2), k = (k_1|k_2) \in C_1 \oplus C_2$, aleshores $k \in K_{C_1 \oplus C_2} \Leftrightarrow \langle k \rangle + c \subseteq C_1 \oplus C_2$ per tot $c \in C_1 \oplus C_2 \Leftrightarrow \langle k_i \rangle + c_i \subseteq C_i$ per tot $c_i \in C_i$ i per $i = 1,2 \Leftrightarrow k_i \in K_{C_i}$ per tot $c_i \in C_i$ i per $i = 1,2 \Leftrightarrow k \in K_{C_1} \oplus K_{C_2}$. Per tant $K_{C_1 \oplus C_2} = K_{C_1} \oplus K_{C_2}$.

Veiem que els elements de $L_\oplus$ pertanyen a diferents classes del conjunt quocient $L = (C_1 \oplus C_2)/(K_{C_1 \oplus C_2})$. Siguin $l = (l_1|l_2), l' = (l_1'|l_2') \in L_\oplus$ on $l_1, l_1' \in L_1$ i $l_2, l_2' \in L_2$, tals que $l \neq l'$. Hem de veure que l i l' són de diferents classes. Suposem que són de la mateixa classe, $\bar{l} = \bar{l'}$, aleshores existeix un $k = (k_1|k_2) \in K_{C_1 \oplus C_2} = K_{C_1} \oplus K_{C_2}$ tal que $k + l = l'$. Per tant $(k_1|k_2) + (l_1|l_2) = (l_1'|l_2') \Rightarrow k_i + l_i = l_i' \Rightarrow \bar{l_i} = \bar{l_i'}$ a

C_i/K_{C_i}, per $i = 1, 2$. Però, com que $l_1, l_1' \in L_1$ i $l_2, l_2' \in L_2$, necessàriament $l_1 = l_1'$ i $l_2 = l_2'$, aleshores arribem a una contradicció, ja que hem suposat $l \neq l'$.

A continuació calculem $|L|$:

$$|L| = \frac{|C_1 \oplus C_2|}{|K_{C_1 \oplus C_2}|} = \frac{|C_1||C_2|}{|K_{C_1}||K_{C_2}|} = \frac{(t_1+1)|K_{C_1}|(t_2+1)|K_{C_2}|}{|K_{C_1}||K_{C_2}|} = (t_1+1)(t_2+1).$$

Com que $L_\oplus \subseteq L$ i $|L| = |L_\oplus| = (t_1+1)(t_2+1)$ obtenim que $L = L_\oplus$. □

Observem que en la demostració del teorema anterior podíem fer servir un kernel parcial en lloc del kernel. Això és el que diu el següent corol·lari i la demostració és idèntica.

Corol·lari 1. *Siguin C_1 i C_2 dos codis q-aris $(n_1, |C_1|)_q$ i $(n_2, |C_2|)_q$ amb els kernels parcials K'_{C_1} i K'_{C_2}, amb els representants dels cosets $L'_1 = \{v_0, \ldots, v_{t_1'}\}$ i $L'_2 = \{w_0, \ldots, w_{t_2'}\}$, respectivament. Aleshores un kernel parcial de la suma directa $C_1 \oplus C_2$ és $K' = K'_{C_1} \oplus K'_{C_2}$ i els seus representants dels cosets són $L'_\oplus = L'_1 \oplus L'_2$.*

El següent corol·lari es pot demostrar per inducció fent servir el Teorema 3.

Corol·lari 2. *Siguin C_i codis q-aris $(n_i, |C_i|, d_i)_q$ per $i = 1, \ldots, r$ amb kernels K_i (poden ser kernels parcials) amb els representants dels cosets L_i. Aleshores la suma directa $C_1 \oplus \cdots \oplus C_r$ és un codi $(n_1 + \cdots + n_2, |C_1| \cdots |C_r|, \min_+\{d_1, \ldots, d_r\})$, el seu kernel és $K = K_1 \oplus \cdots \oplus K_r$ i els seus representants dels cosets són $L = L_1 \oplus \cdots \oplus L_r$ i si C_i són lineals per tot $i = 1, \ldots, r$ aleshores $C_1 \oplus \ldots \oplus C_r$ és lineal i $\dim(C_1 \oplus \cdots \oplus C_r) = \dim(C_1) + \cdots + \dim(C_r)$.*

Definició 27. *Sigui S_m el grup simètric del conjunt $\{1, 2, \ldots, m\}$. Definim l'operador $*^n$, on $n \in \mathbb{N} \cup \{0\}$, com*

$$\begin{array}{rl} *^n : S_m & \longrightarrow S(\{n+1, \ldots, n+m\}) \leq S_{n+m} \\ \pi & \longmapsto *^n(\pi) \end{array}$$

on $S(\{n+1, \ldots, n+m\})$ és el grup simètric del conjunt $\{n+1, n+2, \ldots, n+m\}$ i

$$\begin{array}{rl} *^n(\pi) : \{n+1, \ldots, n+m\} & \longrightarrow \{n+1, \ldots, n+m\} \\ j & \longmapsto \pi(j-n)+n. \end{array}$$

Observem que $*^n(\pi)(i+n) = \pi(i)+n$ per $i = 1, \ldots, m$ i $*^n(S_m) = S(\{n+1, \ldots, n+m\})$. Denotarem $*^n\pi$ en lloc de fer servir el parèntesis $*^n(\pi)$.

Exemple 13. *Sigui una permutació $\pi = (1, 2, 3) \in S_3$, aleshores $*^2\pi = (3, 4, 5) \in S(\{3, 4, 5\})$ és una permutació al grup simètric $S(\{3, 4, 5\})$. Però també podem pensar com $*^2\pi = (3, 4, 5) \in S_5$.*

Siguin S_n i S_m grups simètrics, observem que S_n i $*^n S_m$ són subgrups de S_{n+m} i són disjunts $S_n \cap *^n S_m = \{id\}$. Aleshores el producte dels subgrups és un altre subgrup $S_n *^n S_m \leq S_{n+m}$, ja que els subgrups commuten $S_n *^n S_m = *^n(S_m)S_n$.

Proposició 10. *Siguin C_1 i C_2 dos codis $(n, |C_1|)_q$ i $(m, |C_2|)_q$, aleshores*

$$PAut(C_1) *^n PAut(C_2) \leq PAut(C_1 \oplus C_2).$$

Demostració. Observem que com $PAut(C_1) \leq S_n$ i $*^n PAut(C_2) \leq *^n S_m$, aleshores $PAut(C_1) *^n PAut(C_2) \leq S_{n+m}$ és un subgrup de S_{n+m}. Veiem que $PAut(C_1) *^n PAut(C_2) \subseteq PAut(C_1 \oplus C_2)$. Siguin $\pi_1 \in PAut(C_1)$ i $\pi_2 \in PAut(C_2)$. Aleshores $(\pi_1 * \pi_2)(C_1 \oplus C_2) = \pi_1(C_1) \oplus \pi_2(C_2) = C_1 \oplus C_2$, i així $\pi_1 * \pi_2 \in PAut(C_1 \oplus C_2)$, per tant $PAut(C_1) *^n PAut(C_2) \leq PAut(C_1 \oplus C_2)$. □

A partir d'ara per simplificar farem servir $*$ en lloc de $*^n$, sempre i quan no hi hagi confusió. És a dir, per exemple sigui el subgrup $G \leq S_n$ i $F \leq S_m$, en lloc d'escriure $G *^n F \leq S_{n+m}$ escriurem $G * F \leq S_{n+m}$.

Exemple 14. *En aquest exemple veiem que en la proposició anterior la igualtat $PAut(C_1) * PAut(C_2) = PAut(C_1 \oplus C_2)$ no sempre es compleix. Considerem la suma directa dels codis zeros q-aris 0_n $(n,1)_q$ i 0_m $(m,1)_q$, $0_n \oplus 0_m$. Com que $PAut(0_n) = S_n$, $|PAut(0_n) * PAut(0_m)| = n!m!$. En canvi $PAut(0_n \oplus 0_m) = S_{n+m}$ i per tant $|PAut(0_n \oplus 0_m)| = (n+m)!$. Per $n, m > 0$, tenim que $n!m! = |PAut(0_n) * PAut(0_m)| < |PAut(0_n \oplus 0_m)| = (n+m)!$.*

Exemple 15. *A continuació veiem que també pot ser el cas que $PAut(C_1) * PAut(C_2) = PAut(C_1 \oplus C_2)$. Considerem $C = \{(0,0,0),(1,1,1)\}$ i $D = \{(0,0,0),(2,2,2)\}$ dos codis ternaris. Suposem que existeix una permutació $\pi \in PAut(C \oplus D) \setminus (PAut(C) * PAut(D))$. Considerem $(1,1,1|2,2,2) \in C \oplus D$. Com que $\pi \in PAut(C \oplus D)$, aleshores $\pi((1,1,1|2,2,2)) = (a,a,a|b,b,b) \in C \oplus D$, però com π ha de moure necessàriament elements del $\{1,2,3\}$ al $\{4,5,6\}$ perquè $\pi \notin PAut(C) * PAut(D)$, per tant $b = 1$. Arribem a una contradicció perquè $(1,1,1) \notin D$ i per tant $PAut(C_1) * PAut(C_2) = PAut(C_1 \oplus C_2)$.*

Es pot demostrar per inducció que en general si C_i són codis $(n_i, |C_i|)_q$ per $i = 1, \ldots, r$, aleshores $PAut(C_1) * \cdots * PAut(C_r) \leq PAut(C_1 \oplus \cdots \oplus C_r)$.

Definició 28. *Sigui C un codi q-ari $(m, |C|)_q$, anomenem el codi Pad de longitud n del codi C com $C \oplus 0_n$, on 0_n és el codi zero $(n,1)_q$.*

Es veu fàcilment a partir de la definició anterior que el codi Pad és un codi $(m+n, |C|)_q$ i té distància mínima $d_H(C)$. A més, un subgrup del grup d'automorfismes és $PAut(C) * S_n$, ja que $PAut(0_n) = S_n$.

Proposició 11. *Sigui C un codi q-ari $(m, |C|)_q$ i 0_n el codi zero $(n,1)_q$. Descomponem C com a suma directa de codis q-aris $C = C_1 \oplus \cdots \oplus C_r$, on $r \geq 1$ i C_i són codis q-aris $(n_i, |C_i|)_q$ per $i = 1, \ldots, r$ i amb $n_1 + \cdots + n_r = m$. Si existeix algun $i \in \{1, \ldots, r\}$ tal que C_i és un codi zero $(n_i, 1)_q$, aleshores*

$$PAut(C) * S_n \neq PAut(C \oplus 0_n).$$

En cas contrari, és a dir que no pertany cap codi zero en la descomposició de C, tenim

$$PAut(C) * S_n = PAut(C \oplus 0_n).$$

Demostració. Demostrem la igualtat. Suposem que no pertany cap codi zero en la descomposició de C. Considerem $\pi \in PAut(C \oplus 0_n) \setminus (PAut(C) * S_n)$. Aleshores π ha de intercanviar almenys una coordenada de C i 0_n, és a dir $\pi(i) = j$, on

$1 \leq i \leq m$ i $m+1 \leq j \leq n+m$. Per tant, sigui $c \in C$ tal que $c_i \neq 0$, existeix aquesta paraula codi c perquè el codi C no té cap coordenada que sigui tot zero. Aleshores, per $(c|0) \in C \oplus 0_n$ tenim que $\pi(c|0) \notin C \oplus 0_n$, ja que $\pi(c|0)_j = c_i \neq 0$. Per tant $PAut(C \oplus 0_n) \setminus (PAut(C) * S_n) = \{\emptyset\}$ i tenim $PAut(C) * S_n = PAut(C \oplus 0_n)$.

Demostrem la no igualtat. Suposem que pertany algun codi zero en la descomposició de C. Sigui $i \in \{1, \ldots, m\}$ la coordenada on per a totes les paraules codi de C hi ha un zero. Considerem la transposició $\tau = (i, j) \in S_{n+m}$, que intercanvia la posició i a una posició j on $m+1 \leq j \leq n+m$. Aleshores tenim $\tau \in PAut(C \oplus 0_n) \setminus (PAut(C) * S_n)$, ja que $\tau \in PAut(C \oplus 0_n)$ està intercanviant la coordenada i de C que es tot zero amb una coordenada j de 0_n que també es tot zero. En canvi $\tau \notin PAut(C) * S_n$ perquè $PAut(C)$ i $*S_n$ són disjunts, ja que només mouen elements del $\{1, \ldots, m\}$ al $\{1, \ldots, m\}$ i del $\{m+1, \ldots, m+n\}$ al $\{m+1, \ldots, m+n\}$ respectivament. Llavors $PAut(C) * S_n \neq PAut(C \oplus 0_n)$. □

Exemple 16. *Considerem el codi binari $C = \{(0,0,0), (1,1,1)\}$ que té com a grup d'automorfismes S_3. Fem el seu codi Pad de longitud 2, $C \oplus \{(0,0)\} = \{(0,0,0|0,0), (1,1,1|0,0)\}$ i el seu grup d'automorfismes és $S_3 * S_2$ i no S_5. Perquè si existís una permutació π de S_5 que no pertany a $S_3 * S_2$, aleshores tindríem que $\pi(1,1,1|0,0) = (*,*,*|0,1)$ o $(*,*,*|1,0)$ i cap de les dues possibilitats pertany al codi Pad $C \oplus \{(0,0)\}$.*

3.2 Suma Plotkin

Definició 29. *Siguin C_1 i C_2 dos codis $(n, |C_1|)_q$ i $(n, |C_2|)_q$. La suma Plotkin de C_1 amb C_2 és*

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

Lema 3. *[15] Siguin C_1 i C_2 dos codis q-aris $(n, |C_1|, d_1)_q$ i $(n, |C_2|, d_2)_q$. La suma Plotkin $C_1 \oplus (C_1 + C_2)$ és un codi q-ari $(2n, |C_1||C_2|, \min_+\{2d_1, d_2\})_q$. Si a més, C_1 i C_2 són lineals amb dimensió k_1 i k_2, aleshores $C_1 \oplus (C_1 + C_2)$ és lineal i té dimensió $k_1 + k_2$.*

Demostració. Calculem primer la distància mínima $d = d_H(C_1 \oplus (C_1 + C_2))$. Siguin $c = (c_1|c_1+c_2), c' = (c_1'|c_1'+c_2') \in C_1 \oplus (C_1+C_2)$. Tenim que $d_H(c, c') = wt_H(c-c') = wt_H((c_1 - c_1'|c_1 - c_1' + c_2 - c_2')) = wt_H(c_1 - c_1') + wt_H(c_1 - c_1' + c_2 - c_2')$. Distingim els següents casos:

- Si $|C_1| = 1$ i $|C_2| > 1$, per $c \neq c' \Rightarrow d_H(c, c') = d_H(c_2, c_2') \geq d_2 \Rightarrow d = d_2$.
- Si $|C_1| > 1, |C_2| = 1$, per $c \neq c' \Rightarrow d_H(c, c') = 2d_H(c_1, c_1') \geq 2d_1 \Rightarrow d = 2d_1$.
- Si $|C_1| = |C_2| = 1 \Rightarrow d = 0$.
- Si $|C_1|, |C_2| > 1$, sigui $c \neq c'$, si $c_2 - c_2' = 0 \Rightarrow d_H(c, c') = 2d_H(c_1, c_1') \geq 2d_1$, en cas contrari $d_H(c, c') = wt_H(c_1 - c_1') + wt_H(c_1 - c_1' + c_2 - c_2') \geq wt_H(c_1 - c_1') + wt_H(c_2 - c_2') - wt_H(c_1 - c_1') \geq d_2$, per tant $d = \min\{2d_1, d_2\}$.

En tots el casos tenim que $d = \min_+\{2d_1, d_2\}$. Suposem que C_1 i C_2 són lineals. Observem que $C_1 \oplus (C_1 + C_2) = C_1^2 \oplus_v (0_1 \times C_2)$, on $C_1^2 = \{(u,u) \mid u \in C_1\}$ és un espai vectorial de la mateixa dimensió que C_1 i 0_1 és el vector zero de longitud n. Notem que és una suma directa vectorial ja que $C_1^2 \cap (0_1 \times C_2) = \{0\}$. Per tant, $C_1 \oplus (C_1 + C_2)$ és lineal i per la fórmula de Grassmann tenim que

$$\dim(C_1^2 \oplus_v (0_1 \times C_2)) = \dim(C_1^2) + \dim(0_1 \times C_2) = \dim(C_1) + \dim(C_2).$$

Finalment, l'aplicació

$$\begin{array}{rl} f : C_1^2 \times C_2 & \longrightarrow C_1 \oplus (C_1 + C_2) \\ ((u,u),v) & \longmapsto (u|u+v) \end{array}$$

és una bijecció i per tant $|C_1 \oplus (C_1 + C_2)| = |C_1^2 \times C_2| = |C_1||C_2|$. □

Igual que en la suma directa, en la suma Plotkin es fa servir el $\min_+$ en lloc de min per calcular distàncies mínimes perquè hem vist en la demostració del lema anterior que el $\min_+$ és vàlid per tots els codis, però min no és vàlid si en la suma Plotkin $C \oplus (C + D)$ si un codi, C o D, té cardinal u i l'altre no.

Exemple 17. *Sigui* $C_1 = \{(0,0,0),(1,1,1)\}$ *un codi binari que té distància mínima* 3 *i* $C_2 = \{(0,0,0)\}$ *un codi binari que té distància mínima* 0 *per definició. Aleshores, el codi* $C_1 \oplus (C_1 + C_2) = \{(0,0,0|0,0,0),(1,1,1|1,1,1)\}$ *té distància mínima 6 que coincideix amb* $\min_+(2\cdot 3, 0) = 6$ *mentre que si es fa servir el mínim* $\min(2\cdot 3, 0) = 0$ *no és cert. El mateix passa amb* $C_2 \oplus (C_2 + C_1) = \{(0,0,0|0,0,0),(0,0,0|1,1,1)\}$*, té distància mínima* 3 *que coincideix amb* $\min_+(2\cdot 0, 3) = 3$*, però no amb* $\min(2\cdot 0, 3) = 0$*. En canvi, si considerem el codi* $C_2 \oplus (C_2 + C_2) = \{(0,0,0|0,0,0)\}$*, veiem que té distància mínima* 0 *i* $\min_+(0,0) = \min(0,0) = 0$ *coincideixen.*

Proposició 12. *Siguin* C_1 *i* C_2 *dos codis lineals* $[n,k_1]_q$ *i* $[n,k_2]_q$*, amb matrius generadores* G_1 *i* G_2 *i matrius de control* H_1 *i* H_2*, respectivament. Aleshores la matriu generadora i matriu de control de la suma Plotkin* $C_1 \oplus (C_1 + C_2)$ *són*

$$G = \begin{pmatrix} G_1 & G_1 \\ 0 & G_2 \end{pmatrix} \quad i \quad H = \begin{pmatrix} H_1 & 0 \\ -H_2 & H_2 \end{pmatrix},$$

respectivament.

Demostració. Com que $C_1 \oplus (C_1 + C_2) = C_1^2 \oplus_v (0_1 \times C_2)$ on 0_1 és el vector zero de longitud n, és una suma directa d'espais vectorials i la matriu generadora de C_1^2 és $(G_1 \quad G_1)$ i de $(0_1 \times C_2)$ és $(0 \quad G_1)$, la matriu generadora de la suma Plotkin és G. També tenim que H és la matriu de control ja que $GH^t = 0$. □

Teorema 4. *[15] Siguin* C_1 *i* C_2 *dos codis q-aris* $(n, |C_1|)_q$ *i* $(n, |C_2|)_q$ *amb els kernels* K_{C_1} *i* K_{C_2}*, i els representants dels cosets* $L_1 = \{v_0, \ldots, v_{t_1}\}$ *i* $L_2 = \{w_0, \ldots, w_{t_2}\}$*, respectivament. Aleshores el kernel de la suma Plotkin* $C_1 \oplus (C_1 + C_2)$ *és* $K_{C_1 \oplus (C_1 + C_2)} = K_{C_1} \oplus (K_{C_1} + K_{C_2})$ *i els seus representants dels cosets són* $L_\oplus = L_1 \oplus (L_1 + L_2)$*.*

Demostració. Sigui $C = C_1 \oplus (C_1 + C_2)$ i K_C el kernel de C. Observem primer que si $(a|a+b) \in C_1 \oplus (C_1 + C_2)$ aleshores $a \in C_1$ i $b \in C_2$, ja que existeixen $c_1 \in C_1$ i $c_2 \in C_2$ tals que $(a|a+b) = (c_1|c_1+c_2)$ i per tant $a = c_1 \in C_1$ i $b = c_2 \in C_2$.

Demostrem primer la igualtat $K_C = K_{C_1} \oplus (K_{C_1} + K_{C_2})$. Sigui $c = (c_1|c_1+c_2) \in C$. Veiem que $K_C \subseteq K_{C_1} \oplus (K_{C_1} + K_{C_2})$. Sigui $k = (k_1|k_1+k_2) \in K_C$ on $k_1 \in C_1$ i $k_2 \in C_2$. Aleshores per qualsevol $\lambda \in \mathbb{F}_q$, tenim que $\lambda k + c \in C$, però com que $\lambda k + c = (\lambda k_1 + c_1|(\lambda k_1 + c_1) + (\lambda k_2 + c_2)) \in C$, per l'observació anterior s'arriba a $\lambda k_1 + c_1 \in C_1$ i $\lambda k_2 + c_2 \in C_1$. Per tant, $k_1 \in K_{C_1}$, $k_2 \in K_{C_2}$, i $k = (k_1|k_1+k_2) \in K_{C_1} \oplus (K_{C_1} + K_{C_2})$. Ara veiem que $K_C \supseteq K_{C_1} \oplus (K_{C_1} + K_{C_2})$. Siguin $k = (k_1|k_1+k_2) \in K_{C_1} \oplus (K_{C_1} + K_{C_2}) \Rightarrow \langle k_1 \rangle + c_1 \subseteq C_1$ i $\langle k_1 + k_2 \rangle + c_1 + c_2 \subseteq C_1 + C_2 \Rightarrow \langle k \rangle + c \subseteq C_1 \oplus (C_1 + C_2)$ aleshores $k \in K_C$. Per tant $K_{C_1} \oplus (K_{C_1} + K_{C_2}) = K_C$.

Sigui L el conjunt dels representants dels cosets de C/K_C. Veiem que $L_1 \oplus (L_1 + L_2) \subseteq L$, per això hem de veure que els elements de $L_1 \oplus (L_1 + L_2)$ pertanyen a diferents classes a C/K_C. Siguin $l = (v|v+w), l' = (v'|v'+w') \in L_1 \oplus (L_1 + L_2)$ on $v, v' \in L_1$ i $w, w' \in L_2$, tal que $l' \neq l$. Hem de veure que l i l' són de diferents classes. Suposem que són de la mateixa classe, $\bar{l} = \bar{l'}$, aleshores existeix $k = (k_1|k_1+k_2) \in K_C = K_{C_1} \oplus (K_{C_1} + K_{C_2})$ on $k_1 \in K_{C_1}$ i $k_2 \in K_{C_2}$, tal que $k + l = l' \Rightarrow (k_1|k_1+k_2) + (v|v+w) = (v'|v'+w') \Rightarrow k_1 + v = v'$ i reescrivint tenim que $(v'|v'+k_2+w) = (v'|v'+w') \Rightarrow v' + k_2 + w = v' + w' \Rightarrow k_2 + w = w'$. Aleshores, $\bar{v} = \bar{v'}$ a C_1/K_{C_1} i $\bar{w} = \bar{w'}$ a C_2/K_{C_2}, però com que $v, v' \in L_1$ i $w, w' \in L_2$, necessàriament $v = v'$ i $w = w'$, aleshores arribem a una contradicció, ja que hem suposat $l \neq l'$.

Com que $|C| = |C_1||C_2| = (t_1+1)|K_{C_1}|(t_2+1)|K_{C_2}|$ i hem vist que $|K_C| = |K_{C_1}||K_{C_2}|$. Aleshores

$$|L| = |C/K_C| = \frac{|C|}{|K_C|} = (t_1+1)(t_2+1) = |L_1 \oplus (L_1 + L_2)|.$$

Com que $L_1 \oplus (L_1 + L_2) \subseteq L$, tenim $L = L_1 \oplus (L_1 + L_2)$. □

Observem que en la demostració del teorema anterior podríem fer servir un kernel parcial en lloc del kernel, que és el que diu el següent corol·lari i la demostració és idèntica.

Corol·lari 3. *Siguin C_1 i C_2 dos codis q-aris $(n, |C_1|)_q$ i $(n, |C_2|)_q$ amb els kernels parcials K'_{C_1} i K'_{C_2}, i els representants dels cosets $L'_1 = \{v'_0, \ldots, v_{t'_1}\}$ i $L_2 = \{w'_0, \ldots, w_{t'_2}\}$, respectivament. Aleshores un kernel parcial de la suma Plotkin $C_1 \oplus (C_1 + C_2)$ és $K' = K'_{C_1} \oplus (K_{C_1} + K'_{C_2})$ i els seus representants dels cosets són $L'_\oplus = L'_1 \oplus (L'_1 + L'_2)$.*

Corol·lari 4. *Siguin C i D dos codis $(n, |C|)_q$ i $(n, |D|)_q$. Aleshores, $C \oplus (C + D)$ és lineal si i només si C i D són lineals.*

Demostració. Pel Lema 3, hem vist que si C i D són lineals aleshores $C \oplus (C + D)$ és lineal. Ara suposem que $C \oplus (C + D)$ és lineal. En aquest cas només té un coset

representatiu que és el zero $(0|0)$. Pel Teorema 3, tenim que $L_1 \oplus (L_1 + L_2) = (0|0)$. Per tant, $L_1 = \{0\}$ i $L_2 = \{0\}$, i obtenim que C i D són lineals. □

Exemple 18. *Siguin* $C_1 = \{(0,0),(1,1),(2,2)\}$ *i* $C_2 = (0,0),(0,1))$ *dos codis ternaris. Clarament* C_1 *és lineal i* C_2 *no ho és perquè* $2(0,1)$ *no hi pertany. La suma Plotkin* $C_1 \oplus (C_1 + C_2) = \{(0,0|1,1),(0,0|0,1),(1,1|0,0),(1,1|0,1),(2,2|0,0),(2,2|0,1)\}$ *és un codi* $(4,6)_3$*, que no és lineal perquè* $2(0,0|0,1)$ *no hi pertany.*

Proposició 13. *Siguin* C_1 *i* C_2 *dos codis* $(n,|C_1|)_q$ *i* $(n,|C_2|)_q$*. Un subgrup del grup d'automorfismes de la suma Plotkin* $C_1 \oplus (C_1 + C_2)$ *és*

$$PAut(C_1) * (PAut(C_1) \cap PAut(C_2)) \leq PAut(C_1 \oplus (C_1 + C_2)).$$

Demostració. Observem que $PAut(C_1) \leq S_n$ i $*(PAut(C_1) \cap PAut(C_2)) \leq *S_n$, aleshores $PAut(C_1) * (PAut(C_1) \cap PAut(C_2)) \leq S_{2n}$ és un subgrup de S_{2n}. Veiem que $PAut(C_1)*(PAut(C_1)\cap PAut(C_2)) \subseteq PAut(C_1 \oplus (C_1+C_2))$. Siguin $\pi_1 \in PAut(C_1)$ i $\pi_2 \in PAut(C_1) \cap PAut(C_2)$. Aleshores $(\pi_1 * \pi_2)(C_1 \oplus (C_1+C_2)) = \pi_1(C_1) \oplus (\pi_2(C_1 + C_2)) = C_1 \oplus (C_1+C_2)$. L'última igualtat és certa si $\pi_2(C_1+C_2) = C_1+C_2$. En efecte, siguin $c \in C_1$ i $d \in C_2$. Veiem que $\pi_2(C_1 + C_2) \subseteq C_1 + C_2$, tenim que $\pi_2(c+d) = (c_{\pi_2(1)} + d_{\pi_2(1)}, \ldots, c_{\pi_2(n)} + d_{\pi_2(n)}) = (c_{\pi_2(1)}, \ldots, c_{\pi_2(n)}) + (d_{\pi_2(1)}, \ldots, d_{\pi_2(n)}) = \pi_2(c) + \pi_2(d) \in C_1 + C_2$, ja que $\pi_2(c) \in C_1$ i $\pi_2(d) \in C_2$. Veiem que $\pi_2(C_1 + C_2) \supseteq C_1 + C_2$. Donats $c \in C_1$ i $d \in C_2$ existiran $c' \in C_1$ i $d' \in C_2$ tals que $c = \pi_2(c')$ i $d = \pi_2(d')$, per tant $c + d = \pi_2(c') + \pi_2(d') = \pi_2(c' + d') \in \pi_2(C + D)$. □

Exemple 19. *Veiem un exemple en que no es dona la igualtat de la proposició anterior,* $PAut(C_1)*(PAut(C_1)\cap PAut(C_2)) \neq PAut(C_1 \oplus (C_1+C_2))$*. En l'Exemple 14 hem vist que* 0_n *i* 0_m *codis zero* $(n,1)_q$ *i* $(m,1)_q$*, tenien* $PAut(0_n \oplus 0_m) \neq PAut(0_n) * PAut(0_m)$*. Aleshores, observem que la suma Plotkin de codis zeros* 0_n*,* $0_n \oplus (0_n + 0_m) = 0_n \oplus 0_n$ *és el mateix que la suma directa. Per tant,* $PAut(0_n \oplus 0_n) \neq PAut(0_n) * PAut(0_n) = PAut(0_n) * (PAut(0_n) \cap PAut(0_n))$*.*

Exemple 20. *Un exemple en que es dona la igualtat en la proposició anterior,* $PAut(C_1)*(PAut(C_1)\cap PAut(C_2)) = PAut(C_1 \oplus (C_1+C_2))$*. Considerem el codi* $C_1 \oplus (C_1 + C_2)$*, on* $C_1 = \{(0,0),(1,1)\}$ *és un codi binari que té* $PAut(C_1) = \{(1,2), id\}$*. Aleshores* $C_1 \oplus (C_1 + C_1) = \{(0,0|0,0),(0,0|1,1),(1,1|0,0),(1,1|1,1)\}$*, és clar que no podem intercanviar les posicions* 1 *i* 2 *amb les posicions* 3 *i* 4*. Per tant* $PAut(C_1 \oplus (C_1 + C_1)) = PAut(C_1) * PAut(C_1)$*.*

4 Implementacions de funcions en MAGMA

En aquesta secció explicarem les funcions `DirectSum(C, D)`, `PlotkinSum(C, D)`, `PadCode(C, n)` i `DirectSum(Q)`, que hem implementat per a la llibreria Q-ARY CODES [12] de MAGMA desenvolupada pel Combinatorics, Coding and Security Group (CCSG) de la Universitat Autònoma de Barcelona (UAB). Aquestes funcions s'han desenvolupat seguint l'estàndard de la llibreria [7].

4.1 MAGMA

MAGMA és un sistema algebraic computacional dissenyat per computacions en àlgebra, teoria de nombres, geometria algebraica i combinatòria algebraica. Està desenvolupat per Computational Algebra Group de la University of Sydney [3]. És un software de pagament. El MAGMA té llibreries per a treballar amb conceptes de teoria de codis q-aris, però només per a codis lineals. Per aquest motiu es va iniciar la creació de la llibreria Q-ARY CODES [12] que té com a objectiu treballar amb codis no lineals de forma eficient.

L'objecte més important de la llibreria és **CodeFld**, que representa un codi q-ari $(n, M)_q$ i té els següents atributs:

- **BaseField**: El cos base $\mathbb{F}_q$ del codi q-ari.
- **Kernel**: El kernel del codi.
- **CosetRepresentatives**: Seqüencia de representants dels cosets del codi.
- **Length**: La longitud n del codi.
- **MinimumDistance**: La distància mínima del codi.
- **MinimumDistanceLowerBound**: Una fita inferior de la distància mínima.
- **MinimumDistanceUpperBound**: Una fita superior de la distància mínima.
- **PAutSubgroup**: Un subgrup del grup d'automorfismes del codi.
- **IsLinear**: És `true` si només si el codi és lineal sobre $\mathbb{F}_q$.
- **IsPAutGroup**: És `true` si se sap que el subgrup **PAutSubgroup** del grup d'automorfismes del codi és el grup d'automorfismes del codi.

Si volem accedir a un atribut d'un codi C, per exemple el **Kernel**, el que fem és `C‘Kernel`.

La funció més important de la llibreria és **QaryCode**, que retorna un objecte `CodeFld`. És una funció constructora de codis en general. A continuació comentem alguna de les seves propietats:

- Si tenim el conjunt `L` amb totes les paraules codi, `QaryCode(L)` construeix el codi calculant el kernel i els representants dels cosets. Si coneixem un kernel parcial (`partialKernel`) i els representants dels cosets (`coset`). Aleshores podem fer servir `QaryCode(partialKernel, coset)`. Finalment, si coneixem el kernel (`kernel`) i els seus representants dels cosets (`coset`), podem especificar-ho en l'entrada de la següent manera `QaryCode(kernel, coset: IsFinalKernel := true)`.
- La funció intenta calcular el kernel a partir de l'argument que li donem. Si l'argument és del tipus `QaryCode(L)`, excepte per `L` petit, trigarà més a calcular el kernel que si és del tipus `QaryCode(partialKernel, coset)`, ja que hi ha algorismes més sofisticats per calcular el kernel a partir del kernel parcial

[15]. Finalment si és del tipus `QaryCode(kernel, coset: IsFinalKernel := true)` aleshores no calcula el kernel, encara que `kernel` no sigui el kernel.

- Després de calcular el kernel, si veu que només té un representant dels cosets que és el vector zero aleshores posa `isLinear := true`, en cas contrari posa `isLinear := false`. Hem vist a la teoria que un codi és lineal si i només si, només té un representant dels cosets que és el vector zero.
- La funció no calcula la distància mínima, perquè en general aquest càlcul és costós computacionalment. Per defecte assigna `MinimumDistanceLowerBound := 1` i per `MinimumDistanceUpperBound` fa servir la fita de Sigleton: $n - \log_q(M) + 1$. Si el codi només té una paraula codi, aleshores la funció assigna `MinimumDistance := 0`.
- Si la distància mínima és coneguda, aleshores els valors assignats als dos camps `MinimumDistanceLowerBound` i `MinimumDistanceUpperBound` són iguals a `MinimumDistance`.
- Per defecte assigna el grup trivial al camp `PAutSubgroup` i `IsPAutGroup := false`.

4.2 Funcions implementades

En aquesta secció explicarem com hem implementat les funcions `DirectSum(C, D)`, `DirectSum(Q)`, `PlokinSum(C, D)` i `PadCode(C, n)`, que es poden trobar a l'Annex. Per a la funció `DirectSum(Q)` hem desenvolupat dos algoritmes, iteratiu i directe. Per a la resta de funcions també hem desenvolupat dos algoritmes: un fent servir l'estructura donada pel kernel i els representants dels cosets, que anomenarem Kernel/Coset (KC), i l'altre fent servir força bruta (BF). Els algoritmes BF i KC només difereixen en la part de la construcció del codi, mentre que els càlculs dels altres atributs del codi com el subgrup d'automorfismes, la distància mínima i les seves cotes, són iguals. Després hem fet proves de rendiment per veure quin algoritme és més eficient. Ja es pot intuir que l'algoritme KC és millor que l'algoritme BF, però pot passar que el BF sigui millor en alguns casos de codis, per exemple en codis amb cardinal petit.

A partir d'ara direm que $d_l(C)$ i $d_u(C)$ són la cota inferior i superior de la distància mínima d'un codi C, respectivament. En `CodelFld`, si es coneix la distància mínima, i.e està assignada, aleshores $d_l(C) = d_u(C) = d_H(C)$. També si s'assigna $d_l(C)$ o $d_u(C)$ fent un `UpdateMinimumDistanceLowerBound()` i `UpdateMinimumDistanceUpperBound()`, aleshores es comprova si $d_l(C) = d_u(C)$ i en aquest cas MAGMA assigna automàticament la distància mínima $d_H(C) = d_l(C)$. A partir d'ara si $d_H(C)$ és conegut tindrem que $d_l(C) = d_u(C) = d_H(C)$ i si les fites coincideixen $d_l(C) = d_u(C)$ tindrem que $d_H(C) = d_l(C)$. Aquesta part serà important per simplificar la implementació dels atributs de distàncies. En les funcions hem calculat tots els atributs dels codis construïts amb les noves funcions menys l'atribut `isLinear`, perquè ja és calcula automàticament quan construïm els codis amb la funció `QaryCode`.

4.2.1 PlotkinSum(C, D)

Donat dos codis C i D q-aris $(n, |C|)_q$ i $(n, |D|)_q$, la funció `PlotkinSum(C, D)` construeix la suma Plotkin $C \oplus (C + D)$. L'algoritme BF és simple, es calculen totes les paraules codi de la suma Plotkin (`plotkinSumCodewords`) a partir de les paraules codi dels codis C i D. Finalment per construir el nou codi amb la suma Plotkin, s'executa la funció `QaryCode(plotkinSumCodewords)`.

En l'algoritme KC fem servir els resultats del Teorema 4. Donats K_C i K_D kernels respectius de C i D, calculem en MAGMA el kernel (`plotkinSumKernel`) de la suma Plotkin $C \oplus (C+D)$, $K_{C\oplus(C+D)} = K_C \oplus (K_C + K_D)$ utilitzant la funció ja implementada en MAGMA per a codis lineals `plotkinSumKernel := PlotkinSum(C‘Kernel, D‘Kernel)`. Els representants dels cosets (`plotkinSumRep`) es calculen a partir dels representants dels cosets dels codis C i D. Finalment per construir el nou codi amb la suma Plotkin s'executa la funció `QaryCode(plotkinSumKernel, plotkinSumRep: IsFinalKernel := true)`.

A continuació, mirem com són les parts comunes dels dos algoritmes. Pel Lema 3 sabem que $d_H(C \oplus (C + D)) = \min\{2d_H(C), d_H(D)\}$, aleshores

$$d_l(C \oplus (C + D)) = \min_+\{2d_l(C), d_l(D)\}$$

$$d_u(C \oplus (C + D)) = \min_+\{2d_u(C), d_u(D)\},$$

ja que per definició de les fites,

$$\min_+\{2d_l(C), d_l(D)\} \leq \min_+\{2d_H(C), d_H(D)\} \leq \min_+\{2d_u(C), d_u(D)\}.$$

Només calculem les cotes $d_u(C \oplus (C + D))$ i $d_l(C \oplus (C + D))$ en la implementació dels algoritmes BF i KC. Això ens evita distingir casos en què les distàncies mínimes dels codis C i D estan assignades o no. Comprovem que és cert en tots els cassos:

- Si les distàncies mínimes de C i D són conegudes, aleshores $d_l(C) = d_u(C) = d_H(C)$ i $d_l(D) = d_u(D) = d_H(D)$ i per tant $d_l(C \oplus (C + D)) = d_u(C \oplus (C + D)) \Rightarrow d_H(C \oplus (C + D) = d_l(C \oplus (C + D))$.
- Si només està assignada la distància mínima d'un codi, per exemple, del codi C, aleshores $d_u(C) = d_l(C) = d_H(C)$, i $d_l(C \oplus (C+D)) = \min_+\{2d_H(C), d_l(D)\}$ i $d_u(C \oplus (C + D)) = \min_+\{2d_H(C), d_u(D)\}$ segueixen sent les cotes.
- Si no està assignada cap distància mínima, aleshores $d_l(C \oplus (C + D)) = \min_+\{2d_l(C), d_l(D)\}$ i $d_u(C \oplus (C + D)) = \min_+\{2d_u(C), d_u(D)\}$.

Per la Proposició 13, un subgrup d'automorfismes de $C \oplus (C + D)$ és $PAut(C) * (PAut(C) \cap PAut(D))$. Per calcular-lo fem servir la funció `DirectSProduct` que està implementada en MAGMA, `DirectProduct(C‘PAutSubgroup, C‘PAutSubgroup meet D‘PAutSubgroup)`. Hem vist a l'Exemple 19 que en general no sabem si $PAut(C) * (PAut(C) \cap PAut(D))$ és el grup d'automorfisme del codi, per tant assignem `IsPAutGroup := false`.

4.2.2 DirectSum(C, D)

Donat dos codis C i D q-aris $(n, |C|)_q$ i $(m, |D|)_q$, la funció `DirectSum(C, D)` construeix la suma directa $C \oplus D$. L'algoritme BF és simple, es calculen totes les paraules codi de la suma directa (`directSumCodewords`) concatenant les paraules codi de C amb les del D. Finalment per construir el nou codi amb la suma directa, s'executa la funció `QaryCode(directSumCodewords)`.

En l'algoritme KC fem servir els resultats del Teorema 3. Donats K_C i K_D kernels respectius de C i D, calculem en MAGMA el kernel (`directSumKernel`) de la suma directa $C \oplus D$, ultilitzant la funció ja implementada en MAGMA per a codis lineals `directSumKernel := DirectSum(C‘Kernel, D‘Kernel)`. Per calcular els representants dels cosets (`directSumRep`) es concatenen els representants dels cosets del C amb els del D. Finalment per construir el nou codi amb la suma la directa s'executa la funció `QaryCode(directSumKernel, directSumRep: IsFinalKernel := true)`.

A continuació, mirem com són les parts comunes dels dos algoritmes. Pel Lema 2 sabem que $d_H(C \oplus D) = \min\{d_H(C), d_H(D)\}$, aleshores

$$d_l(C \oplus D) = \min_+\{d_l(C), d_l(D)\}$$

$$d_u(C \oplus D) = \min_+\{d_u(C), d_u(D)\},$$

ja que per definició de les fites,

$$\min_+\{d_l(C), d_l(D)\} \leq \min_+\{d_H(C), d_H(D)\} \leq \min_+\{d_u(C), d_u(D)\}.$$

Només calculem les cotes $d_u(C \oplus D)$ i $d_l(C \oplus D)$ en la implementació dels algoritmes BF i KC. Això ens evita distingir casos en què les distàncies mínimes dels codis C i D estan assignades o no. Comprovem que és cert en tots els cassos:

- Si les distàncies mínimes de C i D són conegudes, aleshores $d_l(C) = d_u(C) = d_H(C)$ i $d_l(D) = d_u(D) = d_H(D)$ i per tant $d_l(C \oplus D) = d_u(C \oplus D) \Rightarrow d_H(C \oplus D) = d_l(C \oplus D)$.
- Si només està assignada la distància mínima d'un codi. Per exemple, del codi C, aleshores $d_u(C) = d_l(C) = d_H(C)$, i $d_l(C \oplus D) = \min_+\{d_H(C), d_l(D)\}$ i $d_u(C \oplus D) = \min_+\{d_H(C), d_u(D)\}$ segueixen sent les cotes.
- Si no està assignada cap distància mínima, aleshores $d_l(C \oplus D) = \min_+\{d_l(C), d_l(D)\}$ i $d_u(C \oplus D) = \min_+\{d_u(C), d_u(D)\}$.

Per la Proposició 10, tenim que un subgrup d'automorfismes de $C \oplus D$ és $PAut(C) * PAut(D)$. Per calcular-lo farem servir la funció `DirectProduct(C‘PAutSubgroup, D‘PAutSubgroup)`. Hem vist a l'Exemple 14 que en general no sabem si $PAut(C) * PAut(D)$ és el grup d'automorfismes del codi, per tant assignem `IsPAutGroup := false`.

4.2.3 PadCode(C, n)

Donat un codi C q-ari $(m, |C|)_q$ i un nombre natural $n \geq 0$, la funció `PadCode(C, n)` construeix el codi Pad de longitud n de C, $C \oplus 0_n$, on 0_n és el codi zero $(n, 1)_q$. Els algoritmes BF i KC es fan de la mateixa manera de com hem explicat en l'Apartat 4.2.2 per `DirectSum(C, D)`. L'única diferència és que en l'algoritme KC, per calcular el kernel del codi $C \oplus 0_n$ fem servir la funció `PadCode(C‘Kernel, n)` de MAGMA per a codis lineals. Finalment construïm el codi Pad que anomenarem `padCode` fent servir la funció `QaryCode` amb diferents entrades segons l'algoritme KC o BF.

A continuació mirem com són les parts comunes dels dos algoritmes. Pel Lema 2 sabem que la distància mínima és $d_H(C \oplus 0_n) = d_H(C)$, i per tant les cotes són $d_l(C \oplus 0_n) = d_l(C)$ i $d_u(C \oplus 0_n) = d_u(C)$. Per la Proposició 10, un subgrup d'automorfismes de $C \oplus 0_n$ és $PAut(C) * S_n$. Per calcular-lo fem servir la funció `DirectProduct(C‘PAutSubgroup, Sym(n))`, on la funció `Sym(n)` calcula S_n.

Gràcies a la Proposició 11, sabem si el subgrup $PAut(C) * S_n$ és el grup d'automorfismes o no. Per això només hem de comprovar que no pertany cap codi zero en la descomposició de C com a suma directa de codis q-aris. Això és equivalent a veure que no hi ha coordenades de C on totes les paraules codi tenen un zero en aquelles coordenades.

En l'algoritme BF es van mirant les paraules codi de C fins que no s'ha trobat cap coordenada que sigui tot zero. En canvi, en l'algoritme KC mirem si hi ha cap coordenada on és tot zero només en els vectors fila de la matriu generadora G del kernel de C, K_C, i els vectors dels representants dels cosets, $L = \{v_0, v_1, \ldots, v_t\}$, menys el vector zero v_0. No fa falta mirar totes les paraules codi de C, ja que pel Teorema 2 qualsevol paraula codi $c \in C$ es pot escriure com

$$c = \lambda_1 u_1 + \cdots + \lambda_\kappa u_\kappa + v,$$

on $\kappa = \dim(K_C)$, els constants $\lambda_1, \ldots, \lambda_\kappa \in \mathbb{F}_q$ i $u_1, \ldots, u_\kappa$ són els vectors d'una base de K_C o sigui, els vectors fila de la matriu generadora G i $v \in L$. Si no se sap que `C‘PAutSubgruoup` és $PAut(C)$, és a dir `C‘IsPAutGroup` és `false`, aleshores posem `padCode‘IsPAutGroup := false`. En canvi, si `C‘IsPAutGroup` és `true` i no hi ha cap coordenada de C que sigui tot zero, aleshores `padCode‘IsPAutGroup := true`, en cas contrari posem `padCode‘IsPAutGroup := false`.

En l'algoritme BF, en el pitjor dels casos, si el codi C té algunes coordenades amb tots zeros, aleshores hem de comprovar totes les paraules codi de C. És a dir, hem de comprovar en $|C|$ paraules codi. En canvi en l'algoritme KC, en el pitjor dels casos, només es comproven tots els vectors de la base del kernel i tots els representants dels cosets menys el vector zero. És a dir, hem de comprovar $\kappa + t$ paraules codi, on $t = |C|/q^\kappa - 1$. Les comprovacions que hem de fer en BF són sempre majors o iguals que en KC, ja que $|C| \geq \kappa + t$. És uns dels avantatges de la descomposició Kernel/Coset.

4.2.4 DirectSum(Q)

Sigui $Q = [C_1, \ldots, C_r]$, on C_i són codi q-aris $(n_i, |C_i|)_q$, on $r \geq 1$. La funció `DirectSum(Q)` construeix la suma directa $C_1 \oplus \cdots \oplus C_r$. Tindrem dos algoritmes, un és l'algoritme directe que és similar a l'algoritme KC de l'Apartat 4.2.2 però amb petites variacions, i l'altre és l'algoritme iteratiu, que fa servir iterativament l'algoritme KC de l'Apartat 4.2.2. És a dir, si hem de calcular $C_1 \oplus C_2 \oplus C_3$, aleshores es calcula `DirectSum(DirectSum(C1, C2), C3)`. Aquest algoritme és més fàcil de programar i menys complex, però creiem que serà més lent que l'algoritme directe, ja que l'algoritme iteratiu ha de cridar la funció `DirectSum` tota l'estona.

Similar a l'Apartat 4.2.2, en l'algoritme directe calculem la cota superior i inferior de la distància mínima:

$$d_u(C_1 \oplus \cdots \oplus C_r) = \min_+\{d_u(C_1), \ldots, d_u(C_r)\}$$

$$d_l(C_1 \oplus \cdots \oplus C_r) = \min_+\{d_l(C_1), \ldots, d_l(C_r)\}.$$

Un subgrup del grup d'automorfismes és $PAut(C_1) * \cdots * PAut(C_r)$, on $PAut(C_i)$ s'obté de l'atribut `PAutSubgroup` de C_i per tot $i \in \{1, \ldots, r\}$. No sabem si el subgrup és el grup d'automorfismes, per tant `IsPAutGroup := false`.

4.3 Tests de caixa negra

Els tests de caixa negra serveixen per comprovar si les funcions que hem implementat són correctes sense veure el funcionament intern de les funcions. És a dir, donem els inputs a la funció que volem testejar i mirem si els outputs són correctes, per això hem de saber els outputs. Aquests tests ens serveixen per comprovar que les funcions que desenvolupem funcionen correctament. Hem fet 10 tests per `DirectSum(C, D)` i `PlotkinSum(C, D)`, 9 tests per `PadCode(C, n)` i 15 tests per `DirectSum(Q)`.

Els codis que s'han utilitzat són: primer s'ha fet servir parelles de codis trivials (codi zero i codi univers), després s'ha agafat parelles de codis amb kernels de diferents dimensions (de dimensió 0 i més gran), codis lineals i no lineals, codis de diferents longituds amb diferents mides i sobre diferents cossos finits $\mathbb{F}_q$. La idea del test és intentar donar els inputs que són els codis, el més divers possible. Els outputs de totes les funcions que hem implementat són també codis, per tant comprovem que aquests codis tenen totes les seves paraules codi i tots els seus atributs correctes.

A continuació mostrem un test de caixa negra, en particular el test 6 per la funció `PlotkinSum(C, D)`, dels algoritmes KC: `PlotkinSumKC(C, D)` i BF: `PlotkinSumBF(C, D)`. La resta dels tests per a les altres funcions tenen la mateixa estructura.

```
print "test 6: q=4, codes with kernel of dimension 2 and 3";
load "./data/CHadq4n16ker2a.m";
load "./data/CHadq4n16ker2a_seqgen.m";
L := CHadq4n16ker2a;
kernel := LinearCode(Matrix(L`Kernel));
C := QaryCode(kernel, L`CosetRepresentatives : IsFinalKernel :=
    true);
C`MinimumDistanceLowerBound := 2;
```

```
8
9  load "./data/CHadq4n16ker3t.m";
10 load "./data/CHadq4n16ker3t_seqgen.m";
11 L := CHadq4n16ker3t;
12 kernel := LinearCode(Matrix(L`Kernel));
13 D := QaryCode(kernel, L`CosetRepresentatives : IsFinalKernel :=
       true);
14 d2 := MinimumDistance(D);
15
16 V := VectorSpace(C`BaseField, C`Length+D`Length);
17 expectedOutput1 := QaryCode([V!(Eltseq(c) cat Eltseq(c+d)): c in
       Set(C), d in Set(D)]);
18 OutputKC := PlotkinSumKC(C, D);
19 OutputBF := PlotkinSumBF(C, D);
20
21 assert expectedOutput1 eq OutputKC;
22 assert expectedOutput1 eq OutputBF;
23 assert OutputKC`MinimumDistanceLowerBound eq 4;
24 assert OutputBF`MinimumDistanceLowerBound eq 4;
25 assert OutputKC`MinimumDistanceUpperBound eq 12;
26 assert OutputBF`MinimumDistanceUpperBound eq 12;
27 assert OutputKC`PAutSubgroup eq DirectProduct(C`PAutSubgroup,
28 C`PAutSubgroup meet D`PAutSubgroup);
29 assert OutputBF`PAutSubgroup eq DirectProduct(C`PAutSubgroup,
30 C`PAutSubgroup meet D`PAutSubgroup);
31 assert not OutputKC`IsLinear;
32 assert not OutputBF`IsLinear;
```

Algoritme 1: Exemple test de `PlotkinSum(C, D)`

En les línies 2 i 3 carreguem les dades, en les línies 4-6 creem el codi `C`, fem el mateix en les línies 9-13 per crear el codi `D`. En la línia 7 canviem la fita inferior de distància mínima de `C` i en la línia 14 calculem la distància mínima de `D`. En la línia 17 construïm la suma Plotkin, en les línies 18 i 19 calculem la suma Plotkin amb l'algorisme KC i BF que hem implementat, respectivament. Finalment, a partir de la línia 21 fem les comprovacions dels codis `OutputKC` i `OutputBF`. Es fa servir `assert A eq B`, que vol dir que si no es compleix $A = B$ aleshores s'atura l'execució del programa.

4.4 Estudi del rendiment

Un cop hem passat els tests de caixa negra, hem de trobar quines versions de les funcions són més eficients computacionalment. Per això hem fet proves de rendiment i hem comparat el temps que triguen les diferents versions.

Per fer l'estudi de rendiment hem triat codis que estan ja implementats a la llibreria. A través dels noms dels fitxers on estan emmagatzemats, podem trobar diferents informacions dels codis. Per exemple, el codi `Cq3n13ker9c`, és un codi sobre el $\mathbb{F}_q$ amb $q = 3, n = 13$ i la dimensió del kernel és 9. Hem agafat aquests codis de la llibreria perquè hi ha molta varietat de codis i ens estalviem el temps en crear diferents codis fer les proves de rendiment. Un cop tenim fetes les proves, comparem el temps que triga cada algoritme, també hem de veure en quins tipus de codis és

millor un algoritme que l'altre.

4.4.1 PlotkinSum(C, D) i DirectSum(C, D)

Identificador del codi	paràmetres	$\|C\|$	$\|L\|$	Plotkin		Direct	
				BF	KC	BF	KC
Cq3n13ker9c	$(13, 59049)_3$	3486784401	9	-	0.001	-	0.000
CHamq2n15ker11	$[15, 11, 3]_2$	4194304	1	28.918	0.000	27.504	0.001
COsterq2n24ker0	$(24, 136)_2$	18496	18496	-	0.100	-	0.091
Cq2n10ker0	$(10, 100)_2$	10000	10000	32.774	0.034	33.005	0.032
Cq4n16ker0	$(16, 16)_4$	256	256	0.039	0.001	0.039	0.001
CHadq4n16ker2a, CHadq4n16ker3t	$(16, 64)_4$ $(16, 64)_4$	4096	4	1.233	0.000	1.224	0.000
COsterq2n18ker3	$(18, 5632)_2$	31719424	495616	-	2.262	-	2.087
COsterq2n25ker7	$(25, 17920)_2$	321126400	19600	-	0.108	-	0.098
CHamq2n16ker4t	$(16, 2048)_2$	4194304	16384	-	0.073	-	0.064
Cq4n4ker2	$(4, 32)_2$	1024	4	0.001	0.000	0.018	0.000
CHadq4n16ker1b Cq4n16ker0	$(16, 64)_4$ $(16, 16)_4$	1024	256	0.097	0.001	0.091	0.001

Taula 4: Proves de rendiment dels algoritmes BF i KC de `DirectSum(C, D)` i `PlotkinSum(C, D)`

Hem utilitzat els mateixos codis per fer les proves de rendiment per a les funcions `PlotkinSum(C, D)` i `DirecteSum(C, D)`. Els codis que hem fet servir i els resultats de les proves estan a la Taula 8 de l'Annex i hem seleccionat alguns resultats significatius en la Taula 4.

En la primera columna de la Taula 4 estan els identificadors dels codis tal com apareixen dins de la base de dades de codis q-aris que conté la llibreria. Si una fila té només un nom d'un codi, `C`, el que hem fet és calcular el `PlotkinSum(C, C)` i `DirecteSum(C, C)`. Si la fila té dos noms, `C` i `D`, calculem `DirectSum(C, D)` i `PlotkinSum(C, D)`. En la segona columna apareixen els paràmetres $(n, M)_q$ si és no lineal o $[n, k, d]_q$ si és lineal. En la tercera i la quarta, el cardinal i el nombre dels representants dels cosets del codi de la suma Plotkin i de la suma directa respectivament. En les dues últimes columnes estan els temps en segons de la suma Plotkin i suma directa amb els seus algoritmes BF i KC.

No hem calculat el temps en BF en alguns casos perquè es triga molt més que en KC. Per exemple, la versió BF pot trigar minuts mentre que la KC triga mil·lèsimes de segon. Observem que la funció `PlotkinSum(C, D)` triga una mica més que la `DirectSum(C, D)` en tots els algoritmes. Això és perquè quan construïm els codis en el `PlotkinSum(C, D)`, per cada paraula codi del codi `C` fem la suma amb totes les paraules codi del codi `D`, mentre que en `DirectSum(C, D)` no cal sumar, i aquesta suma té un cost.

L'algoritme BF sempre és molt més lent que el KC, ja que el BF depèn del cardinal del codi construït $|C|$ mentre que el KC només depèn del nombre de representants

dels cosets del codi construit $|L|$ i no del $|C|$. Per exemple, el cas més extrem de KC és el codi `Cq3n13ker9c`. En aquest cas el cardinal del codi construït és $|C| = 3486784401$ i el nombre dels cosets és $|L| = 9$, per tant el càlcul és en un instant. En canvi, amb el codi `COsterq2n24ker0`, el cardinal $|C| = 18496$ és molt més petit i el nombre de cosets és $|L| = 18496$ molt més gran, però el càlcul triga molt més.

Això és perquè quan construïm els codis amb l'algoritme KC, l'únic cost que té l'algorisme KC és el de construir els representants dels cosets, el kernel de forma directa a partir dels kernels dels codis `C` i `D`, i la construcció del nou codi amb l'estructura Kernel/Coset tenint ja el kernel del nou codi, per tant sense que la funció `QaryCode` l'intenti calcular. En canvi, amb l'algorisme BF, s'han de guardar totes les paraules codi, fent que es necessiti molta memòria, i a més a més, la funció `QaryCode` ha de calcular el kernel i els representants dels cosets, fent que es necessiti més temps de còmput. Per exemple, en `Cq2n10ker0` que és un codi que té un kernel de dimensió 0 i té 100 representants dels cosets, els algoritmes KC i BF triguen aproximadament 0.03 i 33 segons respectivament. En teoria haurien de trigar el mateix perquè el codi `Cq2n10ker0` no té kernel i per tant teòricament la construcció és la mateixa i ocupa la mateixa memòria en KC i en BF. Però com que la funció `QaryCode` sempre calcula el kernel si no li donem, en BF triga més.

Hem agafat algunes parelles de codis diferents entre ells, i hem vist que el BF triga més amb codis combinats que amb codis repetits, tot i tenir el mateix cardinal. Per exemple, com es pot veure en la fila de `Cq4n4ker2` i en la fila de `CHadq4n16ker1b, Cq4n16ker0`.

4.4.2 PadCode(C, n)

Identificador del codi	$\|C\|$	$\|L\|$	BF	KC
Cq4n16ker0	16	16	0.001	0.000
Cq4n4ker2	32	2	0.001	0.000
CHadq4n16ker3t	64	1	0.001	0.000
Cq2n10ker0	100	100	0.005	0.001
CHamq2n15ker11	2048	1	0.011	0.000
Cq2n20ker5	128	4	0.002	0.000
COsterq2n18ker3	5632	704	1.304	0.004
COsterq2n25ker7	17920	140	19.053	0.001
Cq3n13ker9c	59049	3	1.371	0.000
Cq3n13ker10	59049	1	0.337	0.000
Cq3n13ker8b	59049	9	1.628	0.000
CKerq2n256ker9	65536	128	3.507	0.005
COsterq2n24ker12	327680	80	10.250	0.001

Taula 5: Proves de rendiment dels algoritmes BF i KC de **PadCode(C, n)**

Els codis que hem fet servir i els resultats de les proves de rendiment estan en la Taula 5. Per a cada prova hem agafat n igual a la longitud del codi C. Com era

d'esperar el KC és més ràpid que el BF, en tots els casos considerats, inclòs quan el kernel té dimensió 0. Com passa amb `DirectSum(C, D)`, l'algoritme KC depèn del nombre de representants dels cosets del codi C i el BF del nombre de paraules codi de C, com podem veure a la Taula 5. Tot i així, hi ha unes excepcions, com per exemple en BF el codi `COsterq2n25ker7` que té cardinal 17920 triga 19.053 segons, molt més que els codis que tenen cardinal més grans. El mateix codi en KC triga 0.001 segons, molt menys que codis amb menys nombre de representants dels cosets. Aquestes exepcions poden ser causades per l'algoritme que fan servir per calcular el camp `IsPAutGroup`.

4.4.3 DirectSum(Q)

Per la funció `DirectSum(Q)` hem fet comparacions de l'algoritme iteratiu amb el directe. Calculem `DirectSim(Q)`, on `Q` és un vector amb r-vegades el codi C. Fem la prova amb r de 2 fins a un nombre tal que el cardinal del codi obtingut, $|C|^r$, no superi la capacitat màxima de la memòria de MAGMA, i mirem com evoluciona el temps de computació en funció de r. Els codis que considerem tenen el cardinal $|C|$ petits de 2, 4, 16, 32, 48 i 64 paraules codi. Per exemple, amb codis de cardinal $|C| = 2$ hem fet proves fins a $r = 30$, que seria fins a $|C|^{30} = 1073741824$ paraules codi i pels que tenen cardinal $|C| = 48$, hem fet fins a $r = 5$, que seria fins a $|C|^5 = 254803968$ paraules codi.

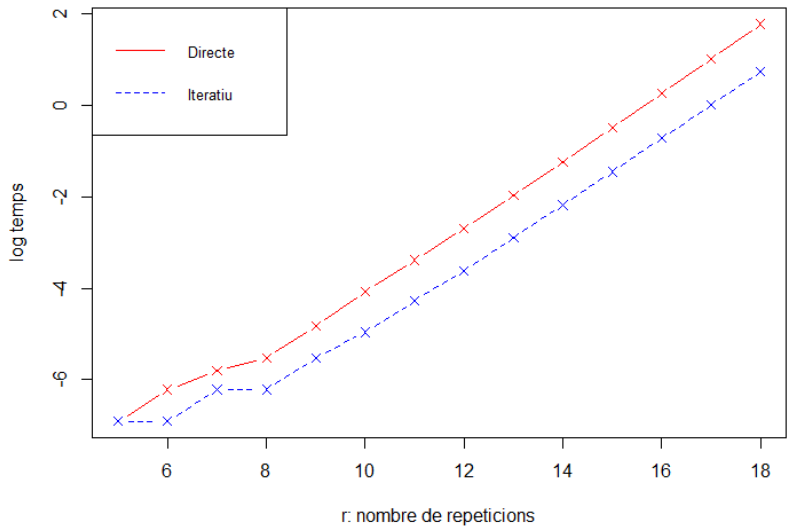

Figura 3: Codi binari $(2, 2)_2$ amb dos representants dels cosets

Podem mirar els resultats de les proves de rendiment en les Taules 6 i 7 de l'Annex. Hem fet alguns gràfics amb els resultats més significatius. Observem en les Figures 3 i 5 que el creixement és lineal en escala logarítmica, això vol dir que el creixement és exponencial a mesura que augmenta r, ja que el cardinal del codi en la repetició r-ésima és $|C|^r$. El nombre de paraules codi augmenta exponencialment amb r i per això el temps de còmput ha d'augmentar exponencialment. Veiem també en les Figures 3 i 5 que la recta de *Iteratiu* està a baix de la recta *Direct*. L'algoritme iteratiu sempre és més ràpid que el directe, i a mesura que el nombre dels repre-

sentants dels cosets augmenta exponencialment la diferència entre els algoritmes també augmenta exponencialment.

En general com veieu en la Taula 6 l'algoritme iteratiu és més ràpid que el directe quan el nombre dels representants dels cosets augmenta. En canvi si el codi Q és lineal, no hi ha diferència entre directe i iteratiu per r petita, però per r gran el temps va augmentant per l'iteratiu mentre que pel directe es manté igual, com veiem en la Figura 4.

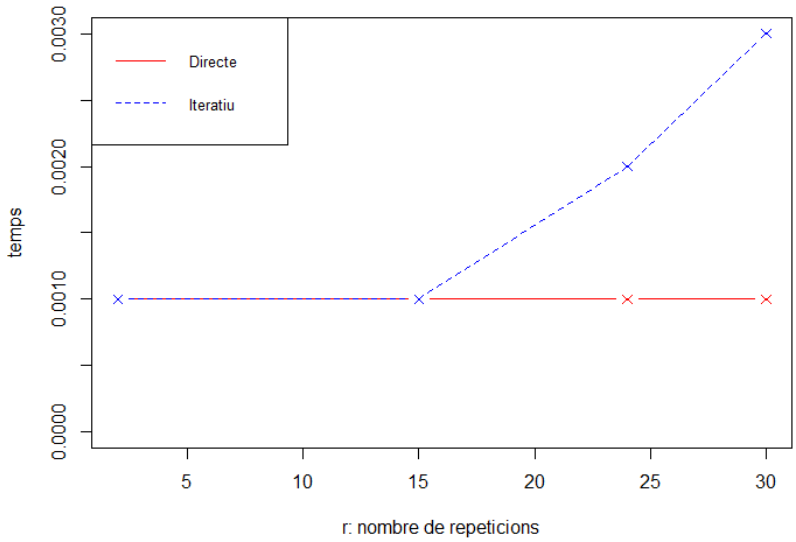

Figura 4: Codi binari $[2, 1, 2]_2$

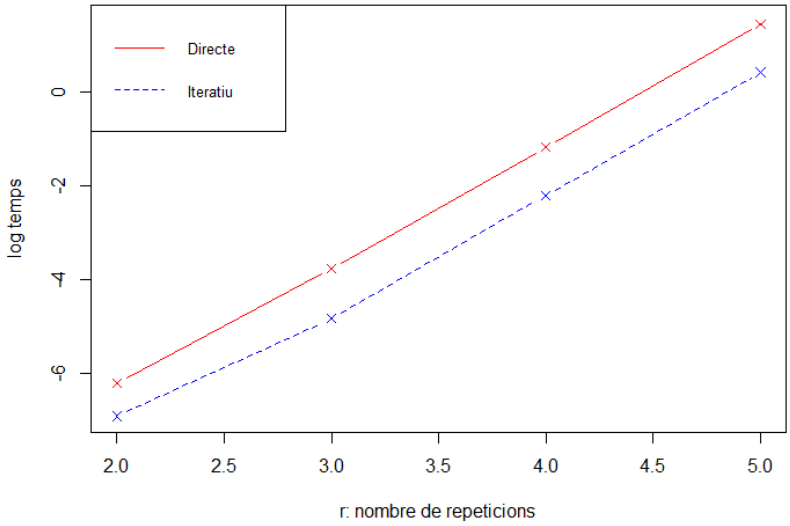

Figura 5: Codi `CHadq4n12ker1`, $(12, 48)_4$ amb 12 representants dels cosets

Creiem que l'algoritme directe en teoria hauria de ser més ràpid, però a la pràctica hem vist que no ho és. Això és a causa de la manera que hem fet l'algoritme directe, en MAGMA no és eficaç perquè per calcular el producte cartesià dels representants dels cosets hem utilitzat variables i càlculs intermedis que han ocupat memòria i

temps de còmput. No hem trobat una manera més simple i directa de fer-ho, però hem vist que per a codis lineals grans l'algoritme directe és més ràpid. Ja que quan els codis són lineals, només tenen un representant dels cosets que és el vector zero i calcular els productes cartesians de vectors zero és ràpid.

5 Conclusions

La primera part ha consistit en l'estudi de la teoria de codis q-aris, sobre cossos finits amb q elements. Per als codis lineals s'ha estudiat el funcionament de la codificació i la descodificació, i per a codis no lineals s'han demostrat resultats que permeten implementar les funcions citades de forma més eficient. La segona part ha consistit a aprendre el sistema MAGMA i la seva llibreria de codis lineals, a més de la llibreria Q-ARY CODES. En la tercera part s'han implementat les funcions i diferents algoritmes per a cada funció, segons la teoria de la primera part. Finalment, s'han validat les funcions i s'han fet estudis de rendiment per determinar quin algoritme de cada funció és millor, fent servir codis q-aris de la llibreria.

En la part de teoria, hem vist des del punt de vista teòric com es codifica i descodifica fent servir codis lineals, perquè funciona la correcció dels errors mitjançant la descodificació per síndromes. Hem donat diferents interpretacions, per exemple, els codis permutacionalments equivalents pertanyen a la mateixa òrbita en una acció de grups i el grup d'automorfismes del codi C són els estabilitzadors de C. Més endavant hem definit un nou operador per a permutacions que ens servirà per simplificar les demostracions dels càlculs dels subgrups del grup d'automorfismes de suma directa, codi Pad i suma Plotkin. També hem trobat i demostrat un resultat que ens permet saber si el subgrup d'un grup d'automorfismes que calculem d'un codi Pad és el grup d'automorfismes del codi o no, i ho hem fet servir per implementar un algoritme eficient en `PadCode(C, n)` gràcies a la representació Kernel/Coset del codi Pad.

Hem donat algunes demostracions diferents, entre elles la que en dona la partició d'un codi en forma Kernel/Coset, fent servir accions de grups. També en els teoremes de la suma directa i suma Plotkin, i aquestes demostracions serveixen també per a kernels parcials. També hem fet servir el mínim positiu $\min_+$ que ens permetia generalitzar el mínim i considerar casos en què els codis tenien només una paraula codi. Junt amb l'estudi de com funciona l'assignació interna de distàncies en la llibreria i la utilització del mínim positiu, ens ha facilitat el càlcul de les distàncies en les funcions implementades (la distància mínima, si es podia calcular, i les seves cotes), ja que hem evitat considerar tots els casos en què la distància mínima dels codis C i D, estava assignada o no.

En `DirectSum(C, D)`, `PloktinSum(C, D)`, `PadCode(C, n)` hem vist que l'algoritme BF és sempre més lent que el KC. Això és dona perquè fem servir la funció `QaryCode` per construir els codis, i `QaryCode` sempre intenta calcular el kernel si no el donem calculat. Com hem vist, l'únic cost que té l'algorisme KC és el de construir els representants dels cosets, el kernel de forma directa a partir dels kernels

dels codis `C` i `D`, i la construcció del nou codi amb l'estructura Kernel/Coset tenint ja el kernel del nou codi, per tant sense que la funció `QaryCode` l'intenti calcular. En canvi, amb l'algorisme BF, s'han de guardar totes les paraules codi, fent que es necessiti molta memòria, i a més a més, la funció `QaryCode` ha de calcular el kernel i els representants dels cosets, fent que es necessiti més temps de còmput. Fins i tot per codis que tenen kernel de dimensió 0, el KC triga menys que el BF. Això és degut al temps per calcular el kernel en el BF. En teoria haurien de trigar el mateix perquè la construcció és la mateixa i ocupen la mateixa memòria, però és per la funció `QaryCode` que es fan servir.

En `DirectSum(Q)` hem vist que l'algoritme iteratiu és més ràpid que l'algoritme directe. En teoria l'algoritme directe hauria de ser més ràpid, però no ha estat el cas perquè no hem trobat una manera eficient de fer l'algoritme directe sense utilitzar càlculs intermedis que han ocupat memòria i temps de còmput. En cavi per a codis lineals l'algoritme directe és més ràpid, perquè no li costa fer aquells càlculs intermedis.

Propostes de millora i investigacions futures:

- En algunes proves de rendiment per codis petits els temps d'execució han quedat bastant propers o iguals entre l'algoritme BF i KC del `DirectSum(C, D)`, `PadCode(C, n)`, `PloktinSum(C, D)` i entre l'algoritme iteratiu i directe de `DirectSum(Q)`. Hauríem de fer més proves de rendiment amb codis petits i agafar més xifres decimals dels segons. Així sabríem definitivament si amb codis petits BF és millor o no.
- En `DirectSum(Q)` fer proves de rendiment amb diferents codis que pertanyin a la seqüencia `Q`. Només s'han fet proves de rendiment fent servir un mateix codi.
- En `PadCode(C, n)`, calcular l'efecte que tenen els algoritmes que es fan servir per calcular `IsPAutGroup` en KC i BF.
- Investigar i trobar per quins tipus de codis es poden trobar subgrups del grup d'automorfismes més grans dels que tenim en suma directa i suma Plotkin.
- Investigar i trobar condicions necessàries i suficients per dir que un subgrup d'automorfismes que calculem en suma directa i suma Plotkin és el grup d'automorfismes o no, com s'ha fet amb el codi Pad.

Referències

[1] R. Antoine, R. Camps, J. Moncasi. *Introducció a l'àlgebra abstracta*. Manuals de la UAB, Servei de Publicacions, UAB, Bellaterra, 2007.

[2] H. Bauer, B. Ganter, F. Hergert. *Algebraic techniques for non-linear codes*. Combinatorica, 3.1, pp. 21–33, 1983.

[3] J. Cannon, W. Bosma, C. Fieker, A. Steel. (2016). *"Handbook of Magma Functions"*. `http://magma.maths.usyd.edu.au/magma/`, 2021.

[4] F. Cedo, A. Reventós. *Geometria plana i Àlgebra lineal.* Manuals de la UAB, Servei de Publicacions, UAB, Bellaterra, 2004.

[5] J. Dorronsoro. *Números, grupos y anillos*. Addison-Wesley Iberoamericana España, S.A. Universidad Autónoma de Madrid, Madrid, 1996.

[6] C. Fernández, M. Villanueva. *Error correction*. Chapter 17, `https://sites.google.com/view/discretemathematicsresources/home`, 2021.

[7] C. C. S. Group. *Magma developers Guide*. Versió 1.1, `http://ccsg.uab.cat`, 2017.

[8] W. C. Huffman, V. Pless. *Fundamentals of error-correcting codes*. Cambridge University Press, 2003.

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

[10] R. Montes. *Amplicación de una librería en MAGMA para la construcción de códigos q-arios*. TFG en Grau de Matemàtiques, Mercè Villanueva (dir.), Jaume Pujol (dir.), Universitat Autònoma de Barcelona, 2019.

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

[12] J. Pujol, M. Villanueva. *Q-ary codes. A MAGMA package*. Versió 1.0, Universitat Autònoma de Barcelona, 2017. `http://ccsg.uab.cat`.

[13] M. Villanueva, F. Feng, J. Pujol. *Eficient representation of binary nonlinear codes: Constructions and minimum distance computation*. Designs, Codes and Cryptography, vol. 76, no. 1, pp. 3-21, 2015.

[14] M. Villanueva, C. Fernández. *Codis detectors i correctors d'errors i algunes de les seves aplicacions a la societat de la informació*. Butlletí de la Societat Catalana de Matemàtiques, vol. 34, no. 1, 53-89 (2019).

[15] F. Zeng. *Nonlinear codes: representation, constructions, minimum distance computation and decoding*. Tesis doctoral, Mercè Villanueva (dir.), Jaume Pujol (dir.), Universitat Autònoma de Barcelona, 2014.

6 Annex

```
////////////////////////////////////////////////////////////////////////////
////////    Copyright 2021 Jaume Pujol and Mercè Villanueva           ///////
////////                                                               ///////
////////    This program is distributed under the terms of GNU         ///////
////////              General Public License                           ///////
////////////////////////////////////////////////////////////////////////////

/*************************************************************************
    This program is free software; you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation; either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.
*************************************************************************/

/*********************************************************/
/*                                                       */
/* Project name: Qary nonlinear codes in MAGMA           */
/* File name: QaryCodes_ConstructionsBiplop.m            */
/*                                                       */
/* Comment: Package developed within the CCSG group      */
/*                                                       */
/* Authors: Biplop Dey and Mercè Villanueva              */
/*                                                       */
/* Revision version and last date: version 1.0 18/03/2021*/
/*                                       1.1 02/06/2021  */
/*                                                       */
/*********************************************************/

/* PACKAGE VERSION */
intrinsic QaryCodes_Extension_version() -> SeqEnum
{Return the current version of this package.}
    version := [1,1];
    return version;
end intrinsic;

import "QaryCodes_Core.m": UpdateMinimumDistance;
import "QaryCodes_Core.m": UpdateMinimumDistanceLowerBound;
import "QaryCodes_Core.m": UpdateMinimumDistanceUpperBound;

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
///////      CONSTRUCTION OF QARY CODES                                ///////
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
```

```
/*****************************************************************/
/*                                                               */
/* Function name: zeroPositions                                  */
/* Parameters: seqVectors, seqCoordPositions                     */
/* Function description: Given a sequence of vectors, seqVectors, */
/*   and a sequence of coordinate positions, seqCoordPositions,  */
/*   return a sequence with the coordinate positions where all   */
/*   the vectors in seqVectors have a zero.                      */
/* Input parameters description:                                 */
/*   - seqVectors: A sequence of vectors of the same length      */
/*   - seqCoordPositions: A sequence of coordinate positions     */
/* Output parameters description:                                */
/*   - A sequence of coordinate positions,                       */
/*     if there is not any, returns an empty sequence           */
/*                                                               */
/*****************************************************************/
zeroPositions := function(seqVectors, seqCoordPositions)
    for c in seqVectors do
        for i in seqCoordPositions do
            if c[i] ne 0 then
                seqCoordPositions := Exclude(seqCoordPositions, i);
            end if;
        end for;

        if #seqCoordPositions eq 0 then
            break;
        end if;
    end for;
    return seqCoordPositions;
end function;

zeroPositionsBF := function(C)
    listLength := [1..(C`Length)];
    for c in Set(C) do
        for i in listLength do
            if c[i] ne 0 then
                listLength := Exclude(listLength, i);
            end if;
        end for;

        if #listLength eq 0 then
            break;
        end if;
    end for;
    return listLength;
end function;
```

```
/*****************************************************************/
/*                                                               */
/* Function name: PadCode                                        */
/* Parameters: C, n                                              */
/* Function description: Add n zeros to the end of each          */
/*   codeword of C.                                              */
/* Input parameters description:                                 */
/*   - C: A qary code                                            */
/*   - n: Integer with the number of zeros to be added           */
/* Output parameters description:                                */
/*   - The q-ary code                                            */
/*                                                               */
/* Signature: (<CodeFld> C, <RngIntElt> n) -> CodeFld            */
/*                                                               */
/*****************************************************************/
//ALGORITME "BF" PER A LA FUNCIÓ PadCode(C,n)
intrinsic PadCodeBF(C::CodeFld, n::RngIntElt) -> CodeFld
{Add n zeros to the end of each codeword of C.}
    require  n ge 0: "n must be non negative";

    if n eq 0 then
        return C;
    end if;

    V := VectorSpace(C`BaseField, C`Length + n);
    padCode := QaryCode([V!(Eltseq(c) cat [0^^n]) : c in Set(C)]);

    padCode`PAutSubgroup := DirectProduct(C`PAutSubgroup, Sym(n));
    padCode`IsPAutGroup := C`IsPAutGroup and (#zeroPositionsBF(C) eq 0);

    UpdateMinimumDistanceLowerBound(~padCode, C`MinimumDistanceLowerBound);
    UpdateMinimumDistanceUpperBound(~padCode, C`MinimumDistanceUpperBound);

    return padCode;

end intrinsic;

//ALGORITME "KC" PER A LA FUNCIÓ PadCode(C,n)
intrinsic PadCodeKC(C::CodeFld, n::RngIntElt) -> CodeFld
{Add n zeros to the end of each codeword of C.}
    require  n ge 0: "n must be non negative";

    if n eq 0 then
        return C;
    end if;

    V := VectorSpace(C`BaseField, C`Length + n);
    kernelPadCode := PadCode(C`Kernel, n);
    cosetRepPadCode := [ V!(Eltseq(c) cat [0^^n]) : c in C`CosetRepresentatives];
    padCode := QaryCode(kernelPadCode, cosetRepPadCode : IsFinalKernel := true);

    padCode`PAutSubgroup := DirectProduct(C`PAutSubgroup, Sym(n));
    if C`IsPAutGroup eq false then
        padCode`IsPAutGroup := false;
    elif C`IsLinear then                //C = C`Kernel
```

```
        padCode`IsPAutGroup := (#zeroPositions(Rows(GeneratorMatrix(C`Kernel)),
                                [1..(C`Length)]) eq 0);
    elif Dimension(C`Kernel) eq 0 then  //C = C`CosetRepresentatives
        padCode`IsPAutGroup := (#zeroPositions(C`CosetRepresentatives,
                                [1..(C`Length)]) eq 0);
    else                                // General case
        zeroCoordinatesKernel := zeroPositions(Rows(GeneratorMatrix(C`Kernel)),
                                [1..(C`Length)]);
        padCode`IsPAutGroup := (#zeroCoordinatesKernel eq 0) and
            (#zeroPositions(C`CosetRepresentatives, zeroCoordinatesKernel) eq 0);
    end if;

    UpdateMinimumDistanceLowerBound(~padCode, C`MinimumDistanceLowerBound);
    UpdateMinimumDistanceUpperBound(~padCode, C`MinimumDistanceUpperBound);

    return padCode;

end intrinsic;
```

```
/*************************************************************************/
/*                                                                       */
/* Function name: minPositive                                            */
/* Parameters: L                                                         */
/* Function description: Given a sequence of numbers, return the  */
/*   minimum positive number in the sequence.                            */
/* Input parameters description:                                         */
/*   - L: A sequence of numbers                                          */
/* Output parameters description:                                        */
/*   - the minimum positive number                                       */
/*                                                                       */
/*************************************************************************/
minPositive := function(L)
    m := 0;
    positiveL := [n : n in L | n gt 0];
    if #positiveL gt 0 then
        m := Min(positiveL);
    end if;
    return m;
end function;

/*************************************************************************/
/*                                                                       */
/* Function name: PlotkinSum                                             */
/* Parameters: C, D                                                      */
/* Function description: Given q-ary codes C and D, both of the    */
/*   same length and over the same base field, construct the       */
/*   Plotkin sum of C and D. The Plotkin sum is a q-ary code that  */
/*   consists of all vectors of the form (u,u+v), where u in C and */
/*   v in D.                                                             */
/* Input parameters description:                                         */
/*   - C: A q-ary code                                                   */
/*   - D: A q-ary code                                                   */
/* Output parameters description:                                        */
/*   - The Plotkin sum code                                              */
/*                                                                       */
/* Remark: This function is based on Proposition 3.2.15            */
/*         in page 51 of Fanxuan Zeng's dissertation,              */
/*         "Nonlinear codes: representation, constructions,        */
/*         minimum distance computation and decoding"              */
/*                                                                       */
/* Signature: (<CodeFld> C, <CodeFld> D) -> CodeFld                */
/*                                                                       */
/*************************************************************************/
//ALGORITME "BF" PER A LA FUNCIÓ PlotkinSum(C,D)
intrinsic PlotkinSumBF(C::CodeFld, D::CodeFld) -> CodeFld
{Given q-ary codes C and D, both of the same length and over the same base field,
construct the Plotkin sum of C and D. The Plotkin sum is a q-ary code that
consists of all vectors of the form (u,u+v), where u in C and v in D.}
    require not IsNull(C): "Argument 1 can not be an empty code";
    require not IsNull(D): "Argument 2 can not be an empty code";
    require (C`BaseField cmpeq D`BaseField): "Arguments must have same base field";
    require (C`Length eq D`Length): "Arguments must have the same length";

    V := VectorSpace(C`BaseField, 2*(C`Length));
```

```
    plotkinSumCodewords := [V!(Eltseq(c) cat Eltseq(c+d)) : c in Set(C), d in Set(D)];
    plotkinSumCode := QaryCode(plotkinSumCodewords);

    plotkinSumCode`PAutSubgroup := DirectProduct(C`PAutSubgroup,
                                                 C`PAutSubgroup meet D`PAutSubgroup);
    plotkinSumCode`IsPAutGroup := false;

    UpdateMinimumDistanceLowerBound(~plotkinSumCode,
          minPositive([2*C`MinimumDistanceLowerBound, D`MinimumDistanceLowerBound]));
    UpdateMinimumDistanceUpperBound(~plotkinSumCode,
          minPositive([2*C`MinimumDistanceUpperBound, D`MinimumDistanceUpperBound]));

    return plotkinSumCode;

end intrinsic;

//ALGORITME "KC" PER A LA FUNCIÓ PlotkinSum(C,D)
intrinsic PlotkinSumKC(C::CodeFld, D::CodeFld) -> CodeFld
{Given q-ary codes C and D, both of the same length and over the same base field,
construct the Plotkin sum of C and D. The Plotkin sum is a q-ary code that
consists of all vectors of the form (u,u+v), where u in C and v in D.}
    require not IsNull(C): "Argument 1 can not be an empty code";
    require not IsNull(D): "Argument 2 can not be an empty code";
    require (C`BaseField cmpeq D`BaseField): "Arguments must have same base field";
    require (C`Length eq D`Length): "Arguments must have the same length";

    V := VectorSpace(C`BaseField, 2*(C`Length));
    plotkinSumRep:= [V!(Eltseq(c) cat Eltseq(c+d)): c in CosetRepresentatives(C),
                                                    d in CosetRepresentatives(D)];
    plotkinSumKernel := PlotkinSum(C`Kernel, D`Kernel);
    plotkinSumCode := QaryCode(plotkinSumKernel, plotkinSumRep : IsFinalKernel := true);

    plotkinSumCode`PAutSubgroup := DirectProduct(C`PAutSubgroup,
                                                 C`PAutSubgroup meet D`PAutSubgroup);
    plotkinSumCode`IsPAutGroup := false;

    UpdateMinimumDistanceLowerBound(~plotkinSumCode,
          minPositive([2*C`MinimumDistanceLowerBound, D`MinimumDistanceLowerBound]));
    UpdateMinimumDistanceUpperBound(~plotkinSumCode,
          minPositive([2*C`MinimumDistanceUpperBound, D`MinimumDistanceUpperBound]));

    return plotkinSumCode;

end intrinsic;
```

```
/*****************************************************************/
/*                                                               */
/* Function name: DirectSum                                      */
/* Parameters: C, D                                              */
/* Function description: Given q-ary codes C and D, both over    */
/*   the same base field, construct the direct sum of C and D.   */
/*   The direct sum is a q-ary code that consists of all vectors*/
/*   of the form (u,v), where u in C and v in D.                 */
/* Input parameters description:                                 */
/*   - C: A q-ary code                                           */
/*   - D: A q-ary code                                           */
/* Output parameters description:                                */
/*   - The direct sum code                                       */
/*                                                               */
/* Signature: (<FldFin> C, <FldFin> D) -> CodeFld                */
/*                                                               */
/*****************************************************************/
//ALGORITME "BF" PER A LA FUNCIÓ DirectSum(C,D)
intrinsic DirectSumBF(C::CodeFld, D::CodeFld) -> CodeFld
{Given q-ary codes C and D, both over the same base field, construct the direct
sum of C and D. The direct sum is a q-ary code that consists of all vectors of
the form (u,v), where u in C and v in D.}
    require not IsNull(C): "Argument 1 can not be an empty code";
    require not IsNull(D): "Argument 2 can not be an empty code";
    require (C`BaseField cmpeq D`BaseField): "Arguments must have same base field";

    V := VectorSpace(C`BaseField, C`Length + D`Length);
    directSumCodewords := [V!(Eltseq(c) cat Eltseq(d)) : c in Set(C), d in Set(D)];
    directSumCode := QaryCode(directSumCodewords);

    directSumCode`PAutSubgroup := DirectProduct(C`PAutSubgroup, D`PAutSubgroup);
    directSumCode`IsPAutGroup := false;

    UpdateMinimumDistanceLowerBound(~directSumCode,
          minPositive([C`MinimumDistanceLowerBound, D`MinimumDistanceLowerBound]));
    UpdateMinimumDistanceUpperBound(~directSumCode,
          minPositive([C`MinimumDistanceUpperBound, D`MinimumDistanceUpperBound]));

    return directSumCode;

end intrinsic;

//ALGORITME "KC" PER A LA FUNCIÓ DirectSum(C,D)
intrinsic DirectSumKC(C::CodeFld, D::CodeFld) -> CodeFld
{Given q-ary codes C and D, both over the same base field, construct the direct
sum of C and D. The direct sum is a q-ary code that consists of all vectors of
the form (u,v), where u in C and v in D.}
    require not IsNull(C): "Argument 1 can not be an empty code";
    require not IsNull(D): "Argument 2 can not be an empty code";
    require (C`BaseField cmpeq D`BaseField): "Arguments must have same base field";

    V := VectorSpace(C`BaseField, C`Length + D`Length);
    directSumRep:= [V!(Eltseq(c) cat Eltseq(d)) : c in CosetRepresentatives(C),
                                                  d in CosetRepresentatives(D)];
    directSumKernel := DirectSum(C`Kernel, D`Kernel);
```

```
    directSumCode := QaryCode(directSumKernel, directSumRep: IsFinalKernel := true);

    directSumCode`PAutSubgroup := DirectProduct(C`PAutSubgroup, D`PAutSubgroup);
    directSumCode`IsPAutGroup := false;

    UpdateMinimumDistanceLowerBound(~directSumCode,
          minPositive([C`MinimumDistanceLowerBound, D`MinimumDistanceLowerBound]));
    UpdateMinimumDistanceUpperBound(~directSumCode,
          minPositive([C`MinimumDistanceUpperBound, D`MinimumDistanceUpperBound]));

    return directSumCode;

end intrinsic;
```

```
/***************************************************************/
/*                                                             */
/* Function name: DirectSum                                    */
/* Parameters: Q                                               */
/* Function description: Given a sequence of q-ary codes Q = [C_1, */
/*   ..., C_r], all defined over the same base field, construct    */
/*   the direct sum of all these q-ary codes C_i, 1 <= i <= r. The */
/*   direct sum is a q-ary code that consists of all vectors of    */
/*   the form (u_1,u_2,...,u_r), where u_i in C_i.              */
/* Input parameters description:                               */
/*   - Q: A sequence of q-ary codes                            */
/* Output parameters description:                              */
/*   - The direct sum code                                     */
/*                                                             */
/* Signature: (<[CodeFld]> Q) -> CodeFld                       */
/*                                                             */
/***************************************************************/
//ALGORITME "Directe" PER A LA FUNCIÓ DirectSum(Q)
intrinsic DirectSumDirect(Q::[CodeFld]) -> CodeFld
{Given a sequence of q-ary codes Q = [C_1,..., C_r], all defined over the same
base field, construct the direct sum of all these q-ary codes C_i, 1 <= i <= r.
The direct sum is a q-ary code that consists of all vectors of the form
(u_1,u_2,...,u_r), where u_i in C_i.}
    r := #Q;
    require r gt 0: "The sequence of codes has to have at least one element";
    if r eq 1 then
        return Q[1];
    end if;
    baseField := Q[1]`BaseField;
    for i in [2..r] do
        require Q[i]`BaseField cmpeq baseField: "Codes must have the same base field";
    end for;

    codeLength := &+[C`Length : C in Q];
    V := VectorSpace(Q[1]`BaseField, codeLength);
    SetQ := <Set(CosetRepresentatives(C)) : C in Q>;
    carProdArray := [Flat([Eltseq(xi) : xi in c]) : c in CartesianProduct(SetQ)];
    directSumRep := [V!c : c in carProdArray];
    directSumKernel := DirectSum([C`Kernel : C in Q]);
    directSumCode := QaryCode(directSumKernel, directSumRep: IsFinalKernel := true);

    upperBoundList := [C`MinimumDistanceUpperBound : C in Q];
    lowerBoundList := [C`MinimumDistanceLowerBound : C in Q];
    UpdateMinimumDistanceLowerBound(~directSumCode, minPositive(lowerBoundList));
    UpdateMinimumDistanceUpperBound(~directSumCode, minPositive(upperBoundList));

    directSumCode`PAutSubgroup := DirectProduct([C`PAutSubgroup : C in Q]);
    directSumCode`IsPAutGroup := false;

    return directSumCode;

end intrinsic;

//ALGORITME "Iteratiu" PER A LA FUNCIÓ DirectSum(Q)
intrinsic DirectSumIterative(Q::[CodeFld]) -> CodeFld
```

```
{Given a sequence of q-ary codes Q = [C_1,..., C_r], all defined over the same
base field, construct the direct sum of all these q-ary codes C_i, 1 <= i <= r.
The direct sum is a q-ary code that consists of all vectors of the form
(u_1,u_2,...,u_r), where u_i in C_i.}
    r := #Q;
    require r gt 0: "The sequence of codes has to have at least one element";
    if r eq 1 then
        return Q[1];
    end if;

    directSumCode := Q[1];
    for i in [2..r] do
        directSumCode := DirectSumKC(directSumCode, Q[i]);
    end for;

    return directSumCode;

end intrinsic;
```

Identificador del codi	$\|C\|$	$\|L\|$	algoritme	2	3	4	5
Cq4n16ker0	16	16	Iteratiu	0.003	0.022	0.411	7.815
			Direct	0.005	0.057	1.057	19.638
CHadq4n16ker3	64	1	Iteratiu	0.000	0.000	0.001	
			Direct	0.000	0.000	0.001	
CHadq4n16ker2a	64	4	Iteratiu	0.000	0.001	0.003	
			Direct	0.001	0.001	0.004	
CHadq4n16ker1b	64	16	Iteratiu	0.001	0.021	0.406	
			Direct	0.003	0.058	1.055	
CHadq4n16ker1c	64	16	Iteratiu	0.001	0.021	0.408	
			Direct	0.004	0.058	1.053	
CHadq4n12ker1	48	12	Iteratiu	0.001	0.008	0.109	1.511
			Direct	0.002	0.023	0.305	4.223
Cq4n4ker2	32	2	Iteratiu	0.000	0.000	0.000	0.001
			Direct	0.001	0.001	0.001	0.001
CHadq4n8ker1	32	8	Iteratiu	0.001	0.003	0.018	0.160
			Direct	0.001	0.007	0.055	0.480
CHadq4n4ker2	16	1	Iteratiu	0.000	0.001	0.001	0.001
			Directe	0.001	0.000	0.000	0.001

Taula 6: Proves de rendiment `DirectSum(Q)`

Repetició	codi cíclic $[2,1,2]_2$		codi $[2,2,1]_2$		codi $(2,2)_4$ amb 2 representants dels cosets	
	Iteratiu	Directe	Iteratiu	Directe	Iteratiu	Directe
2	0.001	0.001	0.000	0.000	0.000	0.001
3	0.001	0.001	0.000	0.001	0.001	0.001
4	0.000	0.001	0.000	0.001	0.001	0.001
5	0.001	0.000	0.000	0.000	0.001	0.001
6	0.001	0.001	0.001	0.000	0.002	0.001
7	0.001	0.001	0.001	0.000	0.002	0.001
8	0.001	0.000	0.001	0.001	0.002	0.001
9	0.001	0.001	0.001	0.001	0.004	0.001
10	0.001	0.001	0.001	0.001	0.071	0.001
11	0.001	0.000	0.001	0.001	0.014	0.001
12	0.001	0.001	0.001	0.000	0.027	0.001
13	0.001	0.001	0.001	0.001	0.055	0.138
14	0.001	0.001	0.002	0.000	0.112	0.289
15	0.001	0.001	0.002	0.000	0.233	0.605
16	0.002	0.001	0.002	0.001	0.484	1.284
17	0.002	0.001	0.002	0.001	1.000	2.734
18	0.002	0.001	0.002	0.001	2.068	5.819
19	0.002	0.001	0.002	0.001		
20	0.002	0.001	0.002	0.001		
21	0.002	0.001	0.002	0.001		
22	0.002	0.001	0.002	0.001		
23	0.002	0.001	0.002	0.001		
24	0.003	0.001	0.003	0.001		
25	0.003	0.000	0.003	0.000		
26	0.003	0.000	0.003	0.001		
27	0.003	0.001	0.003	0.001		
28	0.003	0.001	0.003	0.001		
29	0.003	0.001	0.003	0.001		
30	0.003	0.001	0.003	0.001		

Taula 7: Proves de rendiment `DirectSum(Q)`

Identificador del codi	paràmetre	$\|C\|$	$\|L\|$	Plotkin		Direct	
				BF	KC	BF	KC
CHadq4n8ker1	$(8, 32)_4$	1024	64	0.032	0.001	0.030	0.001
Zero code	$[8, 0, 8]_2$	1	1	0.000	0.000	0.000	0.000
Universe Code	$[2, 2, 1]_3$	81	1	0.001	0.000	0.001	0.000
Cq4n4ker2	$(4, 32)_4$	1024	4	0.001	0.000	0.018	0.000
CHadq4n16ker1c	$(16, 64)_4$	4096	256	0.310	0.001	0.309	0.001
Linear code over GF(3)	$[7, 4, 2]_3$	6561	1	0.028	0.000	0.022	0.000
Linear code over GF(3)	$[7, 3, 2]_3$	729	1	0.003	0.000	0.003	0.000
Two linear code over GF(3)	$[7, 4, 2]_3$ [7, 3, 2]	2187	1	0.011	0.001	0.008	0.001
CHadq4n16ker1b	$(16, 64)_4$	4096	256	0.317	0.001	0.320	0.001
Cq4n16ker0	$(16, 16)_4$	256	256	0.039	0.001	0.039	0.001
CHadq4n16ker1b Cq4n16ker0	$(16, 64)_4$ (16, 16)	1024	256	0.097	0.001	0.091	0.001
CHadq4n16ker2a	$(16, 64)_4$	4096	16	0.098	0.000	0.091	0.000
CHadq4n16ker3t	$(16, 64)_4$	4096		0.022	0.001	0.022	0.001
CHadq4n16ker2a, CHadq4n16ker3t	$(16, 64)_4$ (16, 64)	4096	4	1.233	0.000	1.224	0.000
Cq2n10ker0	$(10, 100)_4$	10000	10000	32.774	0.034	33.005	0.032
Cq2n10ker0, $[10, 4, 1]_2$	(10, 100) $[10, 4, 1]_2$	1600	100	0.024	0.001	0.022	0.001
CHamq2n15ker11	$[15, 11, 3]_2$	4194304	1	28.918	0.000	27.504	0.001
Cq2n20ker5	$(20, 128)_2$	16384	16	0.303	0.000	0.323	0.001
Cq2n20ker0	$(20, 127)_2$	16129	16129	-	0.079	-	0.072
CHamq2n16ker4t	$(16, 2048)_2$	4194304	16384	-	0.073	-	0.064
COsterq2n24ker0	$(24, 136)_2$	18496	18496	-	0.100	-	0.091
Cq3n13ker9c	$(13, 59049)_3$	3486784401	9	-	0.001	-	0.000
COsterq2n18ker3	$(18, 5632)_2$	31719424	495616	-	2.262	-	2.087
COsterq2n25ker7	$(25, 17920)_2$	321126400	19600	-	0.108	-	0.098
Cq4n16ker0	$(16, 16)_4$	256	256	0.039	0.001	0.039	0.001

Taula 8: Proves de rendiment `DirectSum(C, D)` i `PlotkinSum(C, D)`