Punctured extended 1-perfect $\mathbb{Z}_4$-linear codes*

J. Borges, C. Fernández

Departament d'Informàtica
Universitat Autònoma de Barcelona
08193 Bellaterra, Spain

Abstract

Let $\mathcal{C}$ be a 1-perfect additive code. The extended code $\mathcal{C}^\star$ is an extended 1-perfect $\mathbb{Z}_4$-linear or additive non $\mathbb{Z}_4$-linear code. If $\mathcal{C}^\star$ is the extended 1-perfect additive non $\mathbb{Z}_4$-linear code of $\mathcal{C}$ and we puncture a binary coordinate, then $(\mathcal{C}^\star)'$ is isomorphic to $\mathcal{C}$. We will prove that a punctured extended $\mathbb{Z}_4$-linear code is not a 1-perfect additive code up to the extended Hamming code of length 16.

Keywords: Perfect codes, extended codes, quaternary codes, additive codes, $\mathbb{Z}_4$-linear codes.

1 Introduction

In this section we will give the basic definition and results about 1-perfect additive codes and extended 1-perfect additive codes.

1.1 Additive codes

Definition 1 *A binary additive code of length n is a subgroup of $(\mathbb{F}^n, \star)$, where $(\mathbb{F}^n, \star)$ is a translation-invariant Abelian group.*

$\mathbb{F}^n$ is isomorphic to $\mathbb{Z}_2^k \times \mathbb{Z}_4^{\frac{n-k}{2}}$, for some k and $\mathcal{C}$ is called an additive code of type $(k, \frac{n-k}{2})$.

Theorem 2 ([BR99]) *Let $\mathcal{C}$ be a 1-perfect additive code of type $(k, \frac{n-k}{2})$, where $n = 2^t - 1$ and $t \geq 3$. Then, there exists a natural number r, such that $2 \leq r \leq t \leq 2r$ and*

(i) $k = 2^r - 1$; that is, $\mathcal{C}$ is of type $(2^r - 1, 2^{t-1} - 2^{r-1})$,

(ii) $\Omega \cong \mathbb{Z}_2^{2r-t} \times \mathbb{Z}_4^{t-r}$, where Ω is the quotient group $\mathbb{F}^n/\mathcal{C}$.

*This work has been partially supported by spanish CICYT grant TIC2000-0739-C04-01.

Theorem 3 ([PR01]) *Let $\mathcal{C}$ be a binary 1-perfect additive code of type $(2^r - 1, 2^{t-1} - 2^{r-1})$, the kernel of $\mathcal{C}$ has dimension:*

$$K(\mathcal{C}) = \begin{cases} 2^r - r - 1, & \text{if } t = r, \\ 2^{r-1} + 2^{t-1} - r, & \text{if } t \neq r. \end{cases}$$

Theorem 4 ([PR01]) *Let $\mathcal{C}$ be a binary 1-perfect additive code of type $(2^r - 1, 2^{t-1} - 2^{r-1})$, of length $n = 2^t - 1$, where $t \geq 4$, then the rank $r(\mathcal{C})$ of $\mathcal{C}$ is:*

$$r(\mathcal{C}) = n - r = 2^t - r - 1.$$

In the table (1) we can see the parameters of the rank and the kernel of 1-perfect additive codes given in the last theorems.

t	r	type: $(k, \frac{n-k}{2})$	$K(\mathcal{C})$	$r(\mathcal{C})$
2	$1, 2$	$(1, 1), (3, 0)$	$3, 3$	$3, 3$
3	$2, 3$	$(3, 2), (7, 0)$	$4, 4$	$4, 4$
4	$2, 3, 4$	$(3, 6), (7, 4), (15, 0)$	$8, 9, 11$	$13, 12, 11$
5	$3, 4, 5$	$(7, 12), (15, 8), (31, 0)$	$17, 20, 26$	$28, 27, 26$
6	$3, 4, 5, 6$	$(7, 28), (15, 24), (31, 16), (63, 0)$	$33, 36, 43, 57$	$60, 59, 58, 57$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$

(1)

1.2 Extended 1-perfect additive codes

Definition 5 *Let $\mathcal{C}$ be a code of length n. The* extended code *$\mathcal{C}^\star$ of $\mathcal{C}$ is a code of length $n + 1$ obtained from $\mathcal{C}$ by adding the parity check coordinate.*

Theorem 6 ([BPR02]) *If $\mathcal{C}^\star$ is an extended 1-perfect additive code of length $n + 1 = 2^t$, then it is of type $(\alpha + 1, \beta)$, where either*

- *$\alpha + 1 = 0$ and it is a $\mathbb{Z}_4$-linear code or*
- *$\alpha = 2^r - 1$, $2 \leq r \leq t \leq 2r$.*

By Theorem 2, a 1-perfect additive code can be constructed as the kernel of a map $\mathbb{F}^n \longrightarrow \mathbb{Z}_2^\alpha \times \mathbb{Z}_4^\beta$, where $\alpha + 2\beta = n$. For any allowable parameter r and t, code $\mathcal{C}^\star$ could be seen as the kernel of a group homomorphism:

$$\mathbb{F}^{n+1} = \mathbb{Z}_2^{\alpha+1} \times \mathbb{Z}_4^\beta \longrightarrow Z_2^\gamma \times \mathbb{Z}_4^\delta$$

where $\alpha + 1 = 2^r$, $\beta = 2^{t-1} - 2^{r-1}$, $\gamma = 2r - t + 1$ and $\delta = t - r$ (see [BPR02]).

1.2.1 Extended 1-perfect additive non $\mathbb{Z}_4$-linear codes

Theorem 7 ([BPR02]) *For any r and $t \geq 4$ such that $2 \leq r \leq t \leq 2r$ there is exactly one extended 1-perfect additive code $\mathcal{C}^*$ of type $(2^r, 2^{t-1} - 2^{r-1})$, up to coordinate permutation.*

The following theorem gives the rank and the dimension of the kernel of such a 1-perfect additive code.

Theorem 8 ([BPR02]) *Let $\mathcal{C}^\star$ be an extended 1-perfect additive code of type $(2^r, 2^{t-1} - 2^{r-1})$ where $t > 3$, then*

(i) $K(\mathcal{C}^\star) = 2^{r-1} + 2^{t-1} - r$ *if* $t \neq r$ *and* $K(\mathcal{C}^\star) = 2^r - r - 1$ *if* $t = r$.

(ii) $r(\mathcal{C}^*) = 2^t - r - 1$.

In the following tables there are the different values of γ and δ and the values of $R = r(\mathcal{C}^\star)$ and $K = K(\mathcal{C}^\star)$ in each case, for t equal to 4, 5 and the general case.

$t=4$	R	K	R	K	R	K
	11	11	12	9	13	8
γ	4		3		1	
δ	0		1		2	

$t=5$	R	K	R	K	R	K	R	K
	26	26	27	20	28	17	$*$	$*$
γ	6		4		2		0	
δ	0		1		2		3	

t	R	K	R	K	R	K	R	K
	$2^t - t - 1$	$2^t - t - 1$	$2^t - t$	$2^{t-2} + 2^{t-1} - t + 1$	$\cdots$	$\cdots$	$2^t - t - 1 + \delta$	$2^{t-\delta-1} + 2^{t-1} - t + \delta$
γ	$t+1$		$t-1$		$\cdots$		γ	
δ	0		1		$\cdots$		δ	

1.2.2 Extended 1-perfect additive $\mathbb{Z}_4$-linear codes

Theorem 9 ([BPR02]) *Let $\mathcal{C}^\star$ be an extended 1-perfect $\mathbb{Z}_4$-linear code of length $n+1 = 2^t \geq 16$, such that $\mathbb{F}^{n+1}/\mathcal{C}^\star$ is isomorphic to $\mathbb{Z}_2^\gamma \times \mathbb{Z}_4^\delta$ for a fixed $\delta \in \{1, \cdots, \lfloor (t+1)/2 \rfloor\}$ and $\gamma = t + 1 - 2\delta$. Then, $\mathcal{C}^\star$ is unique, up to coordinate permutation.*

Theorem 10 ([BPR02]) *Let $\mathcal{C}^\star$ be an extended 1-perfect $\mathbb{Z}_4$-linear code of length $n+1 = 2^t$ and assume the quotient set is isomorphic to $G = \mathbb{Z}_2^\gamma \times \mathbb{Z}_4^\delta$.*

For $t > 4$, $r(\mathcal{C}^\star) = 2^t - t - 1 + \delta$.

For $t = 4$, either $G = \mathbb{Z}_2^{t-1} \times \mathbb{Z}_4$ and $r(\mathcal{C}^\star) = 2^t - t - 1$, i.e. $\mathcal{C}^\star$ is linear; or $G = \mathbb{Z}_2 \times \mathbb{Z}_4^2$ and $r(\mathcal{C}^\star) = 2^t - t + 1$.

Theorem 11 ([BPR02]) *Let $\mathcal{C}^\star$ be an extended 1-perfect $\mathbb{Z}_4$-linear code of length $n+1=2^t$.*
For $\delta=1$, the dimension of the kernel is $K(\mathcal{C}^\star)=2^{t-1}+t-1$.
For $\delta=2$, the dimension of the kernel is $K(\mathcal{C}^\star)=2^{t-1}-2^{\delta-1}+2=2^{t-1}$.
For $\delta\geq 3$, the dimension of the kernel is $K(\mathcal{C}^\star)=2^{t-1}-2^{\delta-1}+1$.

In the following tables there are the different values of γ and δ and the values of $R=r(\mathcal{C}^\star)$ and $K=K(\mathcal{C}^\star)$ in each case, for t equal to 4, 5 and the general case.

$t=4$	R	K	R	K	R	K
	$*$	$*$	11	11	13	8
γ	4		3		1	
δ	0		1		2	

$t=5$	R	K	R	K	R	K	R	K
	$*$	$*$	27	20	28	16	29	13
γ	6		4		2		0	
δ	0		1		2		3	

t	R	K	R	K	R	K	R	K
	$*$	$*$	2^t-t	$2^{t-1}+t-1$	$\cdots$	$\cdots$	$2^t-t-1+\delta$	$2^{t-1}-2^{\delta-1}+1$
γ	$t+1$		$t-1$		$\cdots$		γ	
δ	0		1		$\cdots$		δ	

2 Punctured extended 1-perfect $\mathbb{Z}_4$-linear codes

Let $\mathcal{C}$ be a 1-perfect additive code. By Theorem 9 the extended code $\mathcal{C}^\star$ is an extended 1-perfect $\mathbb{Z}_4$-linear or additive non $\mathbb{Z}_4$-linear code. If $\mathcal{C}^\star$ is the extended 1-perfect additive non $\mathbb{Z}_4$-linear code of $\mathcal{C}$ and we puncture a binary coordinate, then $(\mathcal{C}^\star)'$ is isomorphic to $\mathcal{C}$. It is not true if we puncture a quaternary coordinate.

The aim of this paper is to prove that a punctured extended $\mathbb{Z}_4$-linear code is not a 1-perfect additive code up to the extended Hamming code of length 16.

Lemma 12 *Let $\mathcal{C}$ be a 1-perfect code and $\mathcal{C}^\star$ the extended code. Then, the rank and the dimension of the kernel of $\mathcal{C}$ and $\mathcal{C}^\star$ coincide.*

Proof: Trivial. ∎

Lemma 13 *Let $\mathcal{C}^\star$ be an extended 1-perfect $\mathbb{Z}_4$-linear code of length $n+1=2^t\geq 16$ such that $\mathbb{F}^{n+1}/\mathcal{C}^\star\cong\mathbb{Z}_2^\gamma\times\mathbb{Z}_4^\delta$, $\gamma+2\delta=t+1$, and assume $\mathcal{C}=(\mathcal{C}^\star)'$ is a 1-perfect additive code. Then, the allowable parameters of t and δ are:*

(i) $t=4,\ \delta=1$,

(ii) $t=4,\ \delta=2$,

(iii) $t=5,\ \delta=1$.

Proof: From Theorems 4 and 10 we obtain:

$$r(\mathcal{C}) = n - r = 2^t - r - 1$$

$$r(\mathcal{C}^\star) = \begin{cases} 2^t - t - 1 \text{ or } 2^t - t + 1 & \text{if } t = 4, \\ 2^t - t - 1 + \delta & \text{if } t > 4. \end{cases}$$

By Lemma 12 $r(\mathcal{C}^\star) = r(\mathcal{C})$, then

- if $t = 4$, either $r = t$ or $r = t - 2$.
- if $t > 4$, $\delta = t - r$.

Now, from Theorems 3 and 11 we obtain:

$$K(\mathcal{C}) = \begin{cases} 2^r - r - 1, & \text{if } t = r, \\ 2^{r-1} + 2^{t-1} - r, & \text{if } t \neq r. \end{cases}$$

$$K(\mathcal{C}^\star) = \begin{cases} 2^{t-1} + t - 1 & \text{if } \delta = 1, \\ 2^{t-1} & \text{if } \delta = 2, \\ 2^{t-1} - 2^{\delta-1} + 1 & \text{if } \delta \geq 3. \end{cases}$$

By Lemma 12 $K(\mathcal{C}) = K(\mathcal{C}^\star)$, hence

- if $t = 4$ and $r = t = 4$, then $2^r - r - 1 = 2^{t-1} + t - 1$ and $\delta = 1$, that corresponds to the case (i).
- if $t = 4$ and $r \neq t$, then $r = t - 2 = 2$, $2^{r-1} + 2^{t-1} - r = 2^{t-1}$ and $\delta = 2$, that corresponds to the case (ii).
- if $t > 4$, then $\delta = t - r$ and there are three cases:

  (1) $2^{r-1} + 2^{t-1} - r = 2^{t-1} + t - 1$ and $\delta = 1$.
  We obtain the equations $2^{r-1} - r = t - 1$ and $1 = t - r$, and hence, $2^{t-2} = 2t - 2$ that has solution if and only if $t = 5$, that correspond to the case (iii).

  (2) $2^{r-1} + 2^{t-1} - r = 2^{t-1}$ and $\delta = 2$.
  $2^{r-1} = r$ if and only if $r = 1, 2$, but for these values of r it is not possible $\delta = t - r$, $t > 4$ and $\delta = 2$.

  (3) $2^{r-1} + 2^{t-1} - r = 2^{t-1} - 2^{\delta-1} + 1$ and $\delta \geq 3$.
  $2^{r-1} - r = -2^{\delta-1} + 1 \leq -3$. But $2^{r-1} \leq r - 3$ has no solution.

∎

Proposition 14 *If $\mathcal{C}^\star$ is an extended 1-perfect $\mathbb{Z}_4$-linear code with parameters $t = 4$ and $\delta = 2$, then the punctured code $(\mathcal{C}^\star)'$ is not a 1-perfect additive code.*

Proof: Assume $\mathcal{C}^\star$ is an extended 1-perfect $\mathbb{Z}_4$-linear code with parameters $t = 4$ and $\delta = 2$. The rank of $\mathcal{C}^\star$ is 13 and the dimension of the kernel is 8. If $\mathcal{C} = (\mathcal{C}^\star)'$ is a 1-perfect additive code then, its rank and dimension of the kernel have the same values than $\mathcal{C}^\star$ and, hence, $\mathcal{C}$ is necessarily of type $(3, 6)$ (see Table (1)).

Program *FindFragments15.c* (see Appendix A.5) construcs the $STS(15)$ associated to $(\mathcal{C}^\star)'$ and computes the pattern array of fragments (see [LeV95]). The pattern array of fragments obtained from $(\mathcal{C}^\star)'$ corresponds to the $STS(15)$ number 3 whereas the pattern array of fragments from the 1-perfect additive code of type $(3, 6)$ corresponds to the $STS(15)$ number 7. As the $STS(15)'s$ obtained from these two codes are not isomorphic, the codes are not isomorphic and, hence, $(\mathcal{C}^\star)'$ is not a 1-perfect additive code. ∎

Proposition 15 *If $\mathcal{C}^\star$ is an extended 1-perfect $\mathbb{Z}_4$-linear code with parameters $t = 5$ and $\delta = 1$, then the punctured code $(\mathcal{C}^\star)'$ is not a 1-perfect additive code.*

Proof: Assume $\mathcal{C}^\star$ is an extended 1-perfect $\mathbb{Z}_4$-linear code with parameters $t = 5$ and $\delta = 1$. The rank of $\mathcal{C}^\star$ is 27 and the dimension of the kernel 20. If $\mathcal{C} = (\mathcal{C}^\star)'$ is a 1-perfect additive code then, its rank and dimension of the kernel have the same values than $\mathcal{C}^\star$ and $\mathcal{C}$ is necessarily of type $(15, 8)$ (see Table (1)).

Program *ExtendedZ4.c* (see Appendix A.7) constructs the set S_1 of weight 3 codewords of the punctured extended 1-perfect $\mathbb{Z}_4$-linear code, $(\mathcal{C}^\star)'$. Program *ExtendedAdd.c* (see Appendix A.6) constructs the set S_2 of weight 3 codewords of the extended 1-perfect non $\mathbb{Z}_4$-linear code with parameters $t = 5$ and $\delta = 1$ puncturing a binary coordinate. Note that, as it is punctured a binary coordinate, this code coincides to the 1-perfect additive code of type $(15, 8)$.

S_1 and S_2 contain 155 codewords of length 31 and weight 3. Using GAP we compute the dimensions of these sets and we obtain $dim(S_1) = 26$ and $dim(S_2) = 27$. As their dimensions are different, S_1 and S_2 are not isomorphic and, hence, the punctured code $(\mathcal{C}^\star)'$ is not a 1-perfect additive code. ∎

Proposition 16 *Let $\mathcal{C}$ be a 1-perfect code of length $n = 2^t - 1$. Let $S_n(\mathcal{C})$ be the $STS(n)$ associated to $\mathcal{C}$. Let $\mathcal{H}_n$ be the Hamming code of length n. Hence, we obtain the following inequation:*

$$r(\mathcal{H}_n) \leq dim(S_n(\mathcal{C})) \leq r(\mathcal{C}).$$

and $r(\mathcal{H}_n) = dim(S_n(\mathcal{C}))$ if and only if $S_n(\mathcal{C})$ is isomorphic to $S_n(\mathcal{H}_n)$.

Proof: Clearly, $dim(S_n) \leq r(\mathcal{C})$. The other inequation and the condition to the equality can be found in ([AJK92, Theorem 8.2.1]) ∎

By the last proposition, if $\mathcal{C}$ is a code of lenght 31, then the dimension of the set of its weight 3 codewords is equals to $r(\mathcal{H}_{31}) = 26$ if and only if this set is isomorphic to the set of weight 3 codewords in $\mathcal{H}_{31}$. In the proof of Proposition 15, we have obtained a code of rank 27; that is, a nonlinear code, but the set of its weight 3 codewords is isomorphic to the set of weight 3 codewords in a linear code.

Theorem 17 *If $\mathcal{C}^\star$ is a binary extended 1-perfect $\mathbb{Z}_4$-linear code of length $n + 1 \geq 16$ then, the punctured code $(\mathcal{C}^\star)'$ 1-perfect is not a 1-perfect additive code up to the case $\mathcal{C}^\star$ equals to the extended of the Hamming code of length 15.*

Proof: Let $\mathcal{C}^\star$ be a binary extended 1-perfect $\mathbb{Z}_4$-linear code of length $n+1 \geq 16$. If $\mathcal{C} = (\mathcal{C}^\star)'$ is a 1-perfect additive code then, by Lemma 13 the allowable parameteres of t and δ are:

(i) $t = 4,\ \delta = 1$,

(ii) $t = 4,\ \delta = 2$,

(iii) $t = 5,\ \delta = 1$.

By Proposition 14, if $\mathcal{C}^\star$ is a binary extended 1-perfect $\mathbb{Z}_4$-linear code with the parameters given in (ii), then the punctured code $(\mathcal{C}^\star)'$ is not a 1-perfect additive code. We obtain the same conclusion with Proposition 15 and the parameters given in (iii). Finally, the parameters given in (i) correspond to the extended of the Hamming code of length 15. ∎

A Source code

A.1 extended.h

```
/*
   Fuctions in Utils.c
*/
void binaryOf(int num,int l, int bin[l]);
void quaternaryOf(int num, int l, int quad[l]);
int wt(int l, int vector[l]);
int qwt(int l, int vector[l]);
int Addwt(int l,int ini,int fin,int vector[l]);
void iniVector(int *vector, int l);
void iniMatrix(int k1, int k2, int matrix[k1][k2]);
void ArrangeCoordinates(int *vector, int ini, int fin);
void printMatrix(int k1, int k2, int matrix[k1][k2]);
void printVector(int l, int vector[l]);
void PrintCode(int C,int N,int code[C][N]);
int sumCoordinates(int l, int vector[l]);
void MultMatrixVector(int r,int c,int M[r][c],int V[c],int R[r]);

/*
   Fuctions in Fragment.c
*/
void CodeToSTS(int N, int C, int B, int code[C][N], int sts[B][3]);
void ConstructTableSTS(int N, int B,int sts[B][3], int Table[N][N]);
void ConstructFragment(int N, int Table[N][N], int fragment[N]);


/*
   Fuctions in Codes.c
*/
void ConstructCodeH15(int code[NCodewords][n]);
void ConstructCodeZ4(int N, int C,int delta, int gamma,
                int code[C][N],int G[(N+1)/2-delta][(N+1)/2]);
void ConstructCodeZ4H(int N, int C,int delta, int gamma,
                int code[C][N],int H[delta+gamma][(N+1)/2]){
void ConstructCodeAddH(int N, int C,int r,int t,
                int code[C][N],int H[r+1][(N+1)/2]);
void GrayImage(int l, int C, int codeZ4[C][l], int codeZ2[C][2*l]);
void MixedGrayImage(int l, int l2,int C, int ini, int fin,
               int codeAdd[C][l], int codeZ2[C][l2]);
void PuncturedCode(int N, int C,int Code1[C][N+1],int Code2[C][N],int c);
```

A.2 Utils.c

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include"extended.h"


/*************************************************
                binaryOf

 From a number 0≤i≤2048, return a vector of
 length 11 of its binary expression.
 *************************************************/
 void binaryOf(int num,int l, int bin[l]){
    int i;

    for (i=0;i<l;i++) bin[i] = 0;
    i = 0;
    do{
       bin[i] = num − 2∗(num/2);;
       num = num/2;
       i += 1;
    }
    while (num > 0);
 }


/*************************************************
                quaternayOf

 From a number 0≤i≤4096, return a vector of
 length 6 of its quaternary expression.
 *************************************************/
 void quaternaryOf(int num, int l, int quad[l]){
    int i;

    for (i=0;i<l;i++) quad[i] = 0;

    i = 0;
    do{
       quad[i] = num − 4∗(num/4);;
       num = num/4;
       i += 1;
    }
```

```
    while (num > 0);
 }


 /*****************************************************
                sumCoordinates

  Gives the sum of the coordinates of a vector of length l
 *****************************************************/
int sumCoordinates(int l, int vector[l]){

   int i, sum=0;

   for(i=0;i<l;i++) sum += vector[i];
   return sum;
}


/*****************************************************
                iniVector

  Inicializes a vector of length l.
 *****************************************************/
void iniVector(int *vector, int l){

   int i;

   for(i=0;i<l;i++) vector[i]=0;

}


/*****************************************************
                iniMatrix

  Inicializes a matrix of dimensio k1xk2
 *****************************************************/
void iniMatrix(int k1, int k2, int matrix[k1][k2]){
  int i,j;

  for(i=0;i<k1;i++)
     for(j=0;j<k2;j++) matrix[i][j] = 0;

}
```

```
/*****************************************************
                printVector

   Prints a vector of length n
 *****************************************************/
 void printVector(int l, int vector[l]){
    int i;
    for (i=0;i<l;i++){
       printf("%d, ", vector[i]);
    }
    printf("\n");
 }


/*****************************************************
                printMatix

   Prints a matrix of dimension k1xk2
 *****************************************************/
void printMatrix(int k1, int k2, int matrix[k1][k2]){
  int i,j;

  for(i=0;i<k1;i++){
     for(j=0;j<k2;j++) printf(" %d",matrix[i][j]);
     printf("\n");
  }
  printf("\n");

}


/*****************************************************
                wt

  Returns the Hamming weight of a vector
 *****************************************************/
int wt(int l, int vector[l]){
  int i;
  int weight=0;

  for(i=0;i<l;i++) weight+=vector[i];
  return weight;
}
```

```
/*****************************************************
                qwt

  Returns the Hamming weight of the binary image of
  a vector in Z4
 *****************************************************/
int qwt(int l, int vector[l]){
  int i;
  int weight=0;

  for(i=0;i<l;i++){
    switch (vector[i]){
       case 0: break;
       case 2: weight += 2;
             break;
       default:weight += 1; /*1 or 3*/
    }
  }
  return weight;
}


/*****************************************************
                Addwt

  Returns the Hamming weight of the binary image of
  a vector in an additive code. The Z4 part of the vector
  corresponds to the coordinates c_i such that
  ini≤i≤fin.
 *****************************************************/
int Addwt(int l,int ini,int fin,int vector[l]){
   int j;
   int weight=0;

   if (ini>0)
      for (j=0;j<ini;j++) weight+=vector[j];
   for(j=ini;j≤fin;j++){
      switch (vector[j]){
         case 0: break;
         case 2: weight += 2;
               break;
         default:weight += 1; /*1 or 3*/
      }
   }
```

```
   if (fin+1<l)
      for (j=fin+1;j<l;j++) weight+= vector[j];
   return weight;
}


/*****************************************************
            ArrangeCoordinates

 Puts the coordinates of a vector between positions
 ini and fin in an decreasing order.
 *****************************************************/
void ArrangeCoordinates(int *vector, int ini, int fin){
   int mid, pos1,pos2,aux;

   pos1 = ini;
   pos2 = fin;
   mid = (vector[pos1]+vector[pos2])/2;
   do{
      while (vector[pos1] < mid) pos1+=1;
      while (vector[pos2] > mid) pos2-=1;
      if (pos1 <= pos2){
         aux = vector[pos1];
         vector[pos1] = vector[pos2];
         vector[pos2] = aux;
         pos1 += 1;
         pos2 -= 1;
      }
   }
   while (pos1 <= pos2);
   if (ini < pos2) ArrangeCoordinates(vector, ini,pos2);
   if (pos1 < fin) ArrangeCoordinates(vector,pos1, fin);

};


/*****************************************************
         MultMatrixVector

   Multiplies a Vector by a Matrix
 *****************************************************/
void MultMatrixVector(int r,int c,int M[r][c],int V[c],int R[r]){
   int i,j;

   iniVector(R,r);
```

```
    for(i=0;i<r;i++)
        for(j=0;j<c;j++)
            R[i]+=M[i][j]*V[j];
}


/*****************************************************
            PrintCode

    Given the matrix of the code we write the codewords
    with the following format:
    V:=FreeLeftModule(GF(2),
    [[c11,c12,...,c1n],[c21,c22,....,c2n],...
     [cm1,cm2,...,cmn]]);
    to be read in GAP.
*****************************************************/
void PrintCode(int C,int N,int code[C][N]){
    int i,j;

    printf("V:=FreeLeftModule(GF(2),[[");
    for(i=0;i<C-1;i++){
        for(j=0;j<N-1;j++){
            if(code[i][j]==0)
                printf("0*Z(2),");
            else printf("Z(2)^0,");
        }
        if(code[i][N-1]==0)
            printf("0*Z(2)],\n[");
        else printf("Z(2)^0],\n[");
    }
    for(j=0;j<N-1;j++){
        if(code[C-1][j]==0)
            printf("0*Z(2),");
        else printf("Z(2)^0,");
    }
    if(code[C-1][N-1]==0)
        printf("0*Z(2)]]);\n");
    else printf("Z(2)^0]]);\n");
}
```

A.3 Fragment.c

```
#include"extended.h"

/*********************************************************
                CodeToSTSN

  Construction of an STS(N) from the codewords of weight 3
  of a code.
*********************************************************/
void CodeToSTSN(int N, int C, int B, int code[C][N], int sts[B][3]){

   int i,j,k,t;
   int weight;

   t = 0;
   for (k=0;k<C;k++){ /*for any codeword in c*/
      weight = wt(N,code[k]);
      if ( weight== 3){
         i = 0;
         for (j=0;j<n;j++){
            if (code[k][j] == 1){
               sts[t][i] = j;
               i += 1;
            }
         }
         t += 1;
      }
   }
}

/*********************************************************
                ConstructTableSTS

  Construction of the product table of the elements of an STS(N)
  if (a,b,c) is a block, then: a*b=b*a=c, a*c=c*a=b, b*c=c*b=a.
  To fill all the table, we define a*a=N (a=0,...,14).
  *********************************************************/
void ConstructTableSTS(int N, int B,int sts[B][3], int Table[N][N]){
   int i,j; /* counters */
   int a,b,c;    /* elements in the STS's blocks */

   iniMatrix(N,N,Table);

   for (i=0;i<N;i++){  /* Specials cases in the table */
```

```
      Table[i][i] = N;
   }
   for (j=0;j<B;j++){
      /* We obtain the elements a,b,c in each block */
      a = sts[j][0];
      b = sts[j][1];
      c = sts[j][2];
      /* We contruct the product table of the elements a,b,c */
      Table[a][b] = c;
      Table[b][a] = c;
      Table[a][c] = b;
      Table[c][a] = b;
      Table[b][c] = a;
      Table[c][b] = a;
   }
}

/*************************************************************
                ConstructFragment

  Construction of the fragment.
 *************************************************************/
void ConstructFragment(int N, int Table[N][N], int fragment[N]){
   int a,b,c; /* elements in each block */
   int e,d;
   int i; /* counters */

   /* Inicialize the fragment */
   for (i=0;i<N;i++) fragment[i] = 0;

   for (a=0;a<N;a++){
      for (b=0;b<N;b++) {
         if (a≠b){
            c = Table[a][b];
            for (d=0;d<N;d++)
               if ((d≠a) && (d≠b) && (d≠c)){
                  e = Table[b][d];
                  if (Table[a][e] == Table[c][d])
                     fragment[a] += 1;
               }
         }
      }
   }
   for (i=0;i<N;i++) fragment[i] = fragment[i]/4;
}
```

A.4 Codes.c

```
#include"extended.h"
#include<math.h>

/*************************************************************
                ConstructCodeH15

   Construct the Hamming code H(11,15).
 *************************************************************/
void ConstructCodeH15(int code[2048][15]){

   int G15[11][15] = {
     {1,1,0,0,1,0,0,0,0,0,0,0,0,0,0},
     {1,0,1,0,0,1,0,0,0,0,0,0,0,0,0},
     {1,0,0,1,0,0,1,0,0,0,0,0,0,0,0},
     {0,1,0,1,0,0,0,1,0,0,0,0,0,0,0},
     {0,1,1,0,0,0,0,0,1,0,0,0,0,0,0},
     {0,0,1,1,0,0,0,0,0,1,0,0,0,0,0},
     {1,1,1,0,0,0,0,0,0,0,1,0,0,0,0},
     {1,1,0,1,0,0,0,0,0,0,0,1,0,0,0},
     {1,0,1,1,0,0,0,0,0,0,0,0,1,0,0},
     {0,1,1,1,0,0,0,0,0,0,0,0,0,1,0},
     {1,1,1,1,0,0,0,0,0,0,0,0,0,0,1},
   };

   int i,j,k,sum;
   int bin[11];

    for(i=0;i<NCodewords;i++){
       binaryOf(11,i,bin);
       for(j=0;j<n;j++){
          sum = 0;
          for(k=0;k<11;k++){
             sum += G15[k][j]*bin[k];
          }
          code[i][j] = sum-2*(sum/2);
       }
    }
}


/*************************************************************
                ConstructCodeZ4
```

```
 Construct an extended Z4 code using its generator matrix G.
 The parameters of the code are: delta and gamma and the
 length of the code is (N+1/2).
 We only will return binary codewords of weight 3
 and length N.
 ************************************************/
void ConstructCodeZ4(int N, int C,int delta,int gamma,
          int code[C][N],int G[(N+1)/2−delta][(N+1)/2],
          int H[gamma+delta][(N+1)/2]){

   int l = (N+1)/2;
   int Q = l−(delta + gamma); /* Number of rows in Z4 */
   int nQ = pow(4,Q);  /*4^Number of rows in Z4*/
   int nB = pow(2,gamma); /*2^Number of rows in Z2*/

   int i,j,k,q,b;
   int quad[Q];
   int bin[gamma];
   int codeZ4[C][l];
   int codeZ2[C][2*l];
   int sum[l], sumt[l];
   int Res[gamma+delta];

   /* Contruction of the Z4-code */
   i = 0;
   for(q=0;q<nQ;q++){
      quaternaryOf(q,Q,quad);
      iniVector(sum,l);
      /* part in Z4*/
      for(j=0;j<l;j++)
         for(k=0;k<Q;k++)
             sum[j] += G[k][j]*quad[k];
      /* part in Z2 */
      for (b=0;b<nB;b++){
         binaryOf(b,gamma,bin);
         for(j=0;j<l;j++){
            sumt[j] = sum[j];
            for(k=0;k<gamma;k++)
               sumt[j] += G[Q+k][j]*2*bin[k];
            sumt[j] = sumt[j]−4*(sumt[j]/4);
         }
         if (qwt(l,sumt)==4 && (sumt[l−1]==1 || sumt[l−1]==2)){
            /* Now, in PuncturedCode, the position to
            be removed has to be the last one */
            for(j=0;j<l;j++)   codeZ4[i][j] = sumt[j];
```

```
            i += 1;
          }
        }
      }

    /* We obtain the binary image*/
    iniMatrix(C,2*l,codeZ2);
    GrayImage(l,C,codeZ4,codeZ2);
    /* We remove the last coordinate*/
    PuncturedCode(N,C,codeZ2,code,N);
}


/*********************************************************
                ConstructCodeZ4H

 Construct an extended Z4 code using its parity check matrix H.
 The parameters of the code are: delta and gamma and the
 length of the code is (N+1/2).
 We only will return binary codewords of weight 3
 and length N.
 *********************************************************/
void ConstructCodeZ4H(int N, int C,int delta,int gamma,
          int code[C][N],int H[delta+gamma][(N+1)/2]){

    int l = (N+1)/2;
    int nQ = pow(4,l/2);  /*4^Number of rows in Z4*/

    int i,j,k,q1,q2;
    int w1;
    int quad1[l/2];
    int quad2[l/2];
    int codeZ4[C][l];
    int codeZ2[C][2*l];
    int V[l];
    int R[gamma+delta]; /*Hv=R, v in Code iff R=0*/

    iniMatrix(C,l,codeZ4);


    /* Contruction of the Z4-code */
    i = 0;
    /* We divide coordinates in 2 group of
    length l/2, because we can work with 4^(l/2)
    and not with 4^l (it's too big)
```

```
    That way, nQ=4^(l/2)*/
    for(q1=0;q1<nQ;q1++){
       quaternaryOf(q1,l/2,quad1);
       if (qwt(l/2,quad1)≤4)
       for(q2=(nQ/2);q2<nQ;q2++){
          /* if q2≥(NQ/2), quad2=xxxxxxx2,
            o xxxxxxx3 , hence, the binary image
            would be: xxxxx...x1x and we will
            punctuared the coordinate (l-2) */
          quaternaryOf(q2,l/2,quad2);
          for(k=0;k<l/2;k++){
             V[k] = quad1[k];
             V[k+l/2] = quad2[k];
          }
          if(qwt(l,V)==4){
             /* If the binary weight of the vector V
             is 4 then, we check if it's a codeword*/
             MultMatrixVector(delta+gamma,l,H,V,R);
             for(j=0;j<gamma;j++)
                R[j] = fmod(R[j],2);
             for(j=gamma;j<gamma+delta;j++)
                R[j] = fmod(R[j],4);
             w1 = wt(gamma+delta,R);
             if(w1==0){  /*V is a codeword*/
                for(j=0;j<l;j++)
                   codeZ4[i][j] = V[j];
                i++;
             }
          }
       }
    }

    /* We obtain the binary image*/
    iniMatrix(C,2*l,codeZ2);
    GrayImage(l,C,codeZ4,codeZ2);
    /* We remove the last coordinate*/
    PuncturedCode(N,C,codeZ2,code,l−2);
}


/*************************************************************
                ConstructCodeAddH

 Construct an extended (not Z4) additive code using its
 parity check matrix H.
```

```
 The parameters of the code are: r and t and the
 length of the code is (N+1/2).
 We only will return binary codewords of weight 4
 and length N.
 ***************************************************/
void ConstructCodeAddH(int N, int C,int r,int t,
          int code[C][N],int H[r+1][(N+1)/2]){

   int delta = t − r;
   int gamma = 2∗r − t + 1;
   int alpha = pow(2,r) − 1;
   int beta = pow(2,t−1) − pow(2,r−1);

   int l = alpha + 1 + beta;
   int l2 = alpha + 1 + 2∗beta ;

   int Q = beta; /∗ Number of rows in Z4 ∗/
   int B = alpha + 1; /∗ Number of rows in Z2 ∗/
   int nQ = pow(4,Q);  /∗4^Number of rows in Z4∗/
   int nB = pow(2,B); /∗2^Number of rows in Z2∗/
   /∗ Note that B+Q = (N+1)/2,  gamma+delta = r+1 ∗/

   int i,j,q,b,h;
   int w1,w4;
   int quad[Q];
   int bin[B];
   int codeAdd[C][l];
   int codeZ2[C][l2];
   int V[l];
   int R[r+1];


   /∗ Contruction of the additive code ∗/
   i = 0;
   for (b=(nB/2);b<nB;b++){
      /∗ part in Z2 ∗/
      /∗ Taking b≥nB we fix a 1 in coordinate B-1∗/
      /∗ We will punctured the code in this coordinate∗/
      binaryOf(b,B,bin);
      if (wt(B,bin)≤4){
         /∗we fix the first coordinate equals 1∗/
         /∗we will punctuared this coordinate∗/
         for(j=0;j<B;j++) V[j] = bin[j];
         for(q=0;q<nQ;q++){
            quaternaryOf(q,Q,quad);
```

```
            for(j=0;j<Q;j++) V[B+j] = quad[j];
            w4 = Addwt(l,16,23,V);
            if (w4==4){  /*V is a vector of weight 4*/
               MultMatrixVector(r+1,B+Q,H,V,R);
               for(j=0;j<gamma;j++)
                  R[j] = fmod(R[j],2);
               for(j=gamma;j<r+1;j++) /*r+1=delta+gamma*/
                  R[j] = fmod(R[j],4);
               w1 = wt(r+1,R);
               if(w1==0){ /*V is a codeword*/
                  for(j=0;j<l;j++) codeAdd[i][j] = V[j];
                  i ++;
               }
            }
         }
      }

   }
   /* We obtain the binary image*/
   /* the Z4 part is from pos. alpha+1 to the end of the vector*/
   MixedGrayImage(l,l2,C,alpha+1,l-1,codeAdd,codeZ2);

   /* We remove the coordinate B-1
   because we have fixed a 1 in it */
   PuncturedCode(N,C,codeZ2,code,B-1);
}


/*********************************************************
                GrayImage

  Given a Z4-code, this function construct the binary image
  of such code via the Gray map: 0->00, 1->01, 2->11, 3->10

 *********************************************************/
void GrayImage(int l, int C, int codeZ4[C][l], int codeZ2[C][2*l]){

   int i,j;

   for(i=0;i<C;i++){
      for(j=0;j<l;j++){
         switch (codeZ4[i][j]){
            case 0: codeZ2[i][2*j]=0;
                  codeZ2[i][2*j+1]=0;
                  break;
```

```
            case 1: codeZ2[i][2*j]=0;
                    codeZ2[i][2*j+1]=1;
                    break;
            case 2: codeZ2[i][2*j]=1;
                    codeZ2[i][2*j+1]=1;
                    break;
            default:codeZ2[i][2*j]=1;
                    codeZ2[i][2*j+1]=0;
          }
        }
    }
}


/*************************************************************
                MixedGrayImage

   Given a additive code, this function construct the binary image
   of such code via the Gray map: 0->00, 1->01, 2->11, 3->10 in
   the Z4 part and the identity in the Z2 part.
   Coordinates in Z4 are those being beetwen ini and fin:
   C=(c0,c1,c2,...c(n-1)) c_ini≤ci≤c_fin, ci in Z4

 *************************************************************/
void MixedGrayImage(int l, int l2,int C, int ini, int fin,
           int codeAdd[C][l], int codeZ2[C][l2]){

   int i,j;
   int pos=0;

   for(i=0;i<C;i++){
      if (ini>0)
         for (j=0;j<ini;j++) codeZ2[i][j]=codeAdd[i][j];
      for(j=ini;j≤fin;j++){
         pos = ini + 2*(j−ini);
         switch (codeAdd[i][j]){
            case 0: codeZ2[i][pos]=0;
                    codeZ2[i][pos+1]=0;
                    break;
            case 1: codeZ2[i][pos]=0;
                    codeZ2[i][pos+1]=1;
                    break;
            case 2: codeZ2[i][pos]=1;
                    codeZ2[i][pos+1]=1;
                    break;
```

```
            default:codeZ2[i][pos]=1;
                    codeZ2[i][pos+1]=0;
            }
        }
        if (fin+1<l)
            for (j=fin+1;j<l;j++)
                codeZ2[i][j+(fin−ini+1)]=codeAdd[i][j];
    }
}


/*********************************************************
                PuncturedCode

    Code2 is obtained from code1 by removing
    the coordinate c

 *********************************************************/
void PuncturedCode(int N, int C,int Code1[C][N+1],int Code2[C][N],int c){
    int i,j;

    iniMatrix(C,N,Code2);

    for(i=0;i<C;i++) {
        for(j=0;j<c;j++) Code2[i][j] = Code1[i][j];
        for(j=c;j<N;j++) Code2[i][j] = Code1[i][j+1];
    }
}
```

A.5 FindFragment15.c

```
#include<math.h>
#include<stdio.h>
#include<stdlib.h>
#include"extended.h"

 int main(){

   /*****************************/
   /* Parameters of the code */
    int N = 15; // Length of the code
    int C = 2048; // Number of codewords in the code
    int B = 35; // Number of blocks in the STS
    int delta = 2;
    int gamma = 1;
    //Dimension of the generator matrix:
    //    (N+1)/2 - delta)x((N+1)/2)*/
    int G[6][8] = {
     {1,3,3,1,0,0,0,0},
     {1,0,2,0,1,0,0,0},
     {2,3,2,0,0,1,0,0},
     {2,0,1,0,0,0,1,0},
     {3,3,1,0,0,0,0,1},
     {1,3,0,0,0,0,0,0},
    };

   /*****************************/

    int code[C][N];
    int sts[B][3];
    int Table[N][N];
    int fragment[N];
    int i,j;


    iniMatrix(C,N,code);
    iniMatrix(B,3,sts);
    iniMatrix(N,N,Table);
    iniVector(fragment,N);

    ConstructCodeZ4(N,C,delta,gamma,code,G);


    /*We obtain the STS(15) from the code
```

```
    and construct the table of the STS*/
    CodeToSTSN(N,C,B,code,sts);
    ConstructTableSTS(N,B,sts,Table);
    /*Construction of the fragment*/
    ConstructFragment(N,Table,fragment);

    ArrangeCoordinates(fragment,0,N − 1);
    printVector(N,fragment);

    return 0;
}
```

A.6 ExtendedAdd.c

```
#include<math.h>
#include<stdio.h>
#include<stdlib.h>
#include"extended.h"

 int main(){

   /*****************************/
    /* Parameters of the additive code */
    int N = 31; // Length of the code
    int B = 155; // Number of blocks in the STS
    int r = 4;
    int t = 5;

    int delta = t − r;
    int gamma = 2∗r − t + 1;
    int alpha = pow(2,r) − 1;
    int beta = pow(2,t−1) − pow(2,r−1);

     /∗Dimension of the generator matrix:
        (alpha+1+beta-(delta+gamma))x(alpha+1+beta)∗/
    int H[5][24] = {
     {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},
     {0,0,0,0,1,1,1,1,0,0,0,0,1,1,1,1,0,0,0,0,1,1,1,1},
     {0,0,1,1,0,0,1,1,0,0,1,1,0,0,1,1,0,0,1,1,0,0,1,1},
     {0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1},
     {0,0,0,0,0,0,0,0,2,2,2,2,2,2,2,2,1,3,1,3,1,3,1,3},
    };

   /*****************************/

    int codeAdd[B][N]; //contains only weight 3 codewords
    int sts[B][3];
    int i,j;

    iniMatrix(B,N,codeAdd);

    ConstructCodeAddH(N,B,r,t,codeAdd,H);
    PrintCode(B,N,codeAdd);

    return 0;
}
```

A.7 ExtendedZ4.c

```
#include<math.h>
#include<stdio.h>
#include<stdlib.h>
#include"extended.h"

 int main(){

   /*****************************/
    /* Parameters of the code */
    int N = 31; // Length of the code
    int B = 155; // Number of blocks in the STS
    int delta = 1;
    int gamma = 4;
    /*Dimension of the generator matrix:
       ((N+1)/2 - delta)x((N+1)/2)*/
    int G[15][16] = {
     {1,3,0,0,3,1,0,0,0,0,0,0,0,0,0,0},
     {1,0,3,0,3,0,1,0,0,0,0,0,0,0,0,0},
     {2,3,3,0,3,0,0,1,0,0,0,0,0,0,0,0},
     {1,3,3,1,0,0,0,0,0,0,0,0,0,0,0,0},
     {1,3,0,0,0,0,0,0,3,1,0,0,0,0,0,0},
     {1,0,3,0,0,0,0,0,3,0,1,0,0,0,0,0},
     {2,3,3,0,0,0,0,0,3,0,0,1,0,0,0,0},
     {1,0,0,0,3,0,0,0,3,0,0,0,1,0,0,0},
     {2,3,0,0,3,0,0,0,3,0,0,0,0,1,0,0},
     {2,0,3,0,3,0,0,0,3,0,0,0,0,0,1,0},
     {3,3,3,0,3,0,0,0,3,0,0,0,0,0,0,1},
     {1,3,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
     {1,0,3,0,0,0,0,0,0,0,0,0,0,0,0,0},
     {1,0,0,0,3,0,0,0,0,0,0,0,0,0,0,0},
     {1,0,0,0,0,0,0,0,3,0,0,0,0,0,0,0},
    };

   int H[5][16] = {
      {0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1},
      {0,0,0,0,1,1,1,1,0,0,0,0,1,1,1,1},
      {0,0,1,1,0,0,1,1,0,0,1,1,0,0,1,1},
      {0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1},
      {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},
   };

   /*****************************/
    int code4[B][N]; //contains only weight 3 codewords
```

```
    int sts[B][3];
    int i,j;
    int Res[5];


    iniMatrix(B,N,code4);
    iniMatrix(B,3,sts);
    ConstructCodeZ4H(N,B,delta,gamma,code4,H);
   /* It would be also correct
   ConstructCodeZ4(N,B,delta,gamma,code4,G);
   that use the generator matrix G*/

    PrintCode(B,N,code4);

    return 0;
}
```

References

[AJK92] E.F. Assmus Jr and J.D. Key. *Designs and their codes*. Cambridge University Press, 1992.

[BPR02] J. Borges, K.P. Phelps, and J. Rifà. The rank and the kernel of 1-perfect additive codes and extended 1-perfect $\mathbb{Z}_4$-linear codes. *Submitted to IEEE Transactions on Information Theory*, 2002.

[BR99] J. Borges and J. Rifà. A characterization of 1-perfect additive codes. *IEEE Transactions on Information Theory 45, pp. 1688-1697*, 1999.

[LeV95] J.M. LeVan. *Designs and codes*. PhD thesis, Auburn University, 1995.

[PR01] K. T. Phelps and J. Rifà. Combinatorial structure of binary 1-perfect additive codes. *Preprint n.488. CRM. Spain*, 2001.