Computing an Invariant of a Linear Code

Mijail Borges-Quintana[1(✉)], Miguel Ángel Borges-Trenard[2], Edgar Martínez-Moro[3(✉)], and Gustavo Torres-Guerrero[1]

[1] Department of Mathematics, Faculty of Natural and Exact Sciences, University of Oriente, Santiago de Cuba, Cuba
mijail@uo.edu.cu, gtorresguerrero85@gmail.com
[2] Doctorate in Mathematics Education, University Antonio Nariño, Bogotá, Colombia
borgestrenard2014@gmail.com
[3] Institute of Mathematics IMUVa, University of Valladolid, Valladolid, Castilla, Spain
edgar.martinez@uva.es

Abstract. In this work we present an efficient algorithm that generates the leader codewords of a linear code in an incremental form. On the other hand, using the set of leader codewords we define a transformation that remains invariant only if the codes are equivalent which is used as a signature for checking the code equivalence problem. An upper bound on the weight of the codewords is imposed to this algorithm in order to get a smallest set that can be also used as a signature for the 'Code Equivalence Problem'.

Keywords: Leader codewords · Code equivalence · Coset leaders

1 Introduction

In this work we are interested in the mathematical aspects of the set of leader codewords of a linear code related with two main issues, its computation and getting a signature for the 'Code Equivalence Problem'. In [3] this set is defined for binary codes and it vis given an algorithm for its computation in an incremental form based on the Gröbner representation [9] of the code. The extension of those results for general linear codes is analyzed in [5].

We formulate a kind of Möller's algorithm for Gröbner representation techniques that generates the leader codewords in an incremental form. Nevertheless, we state and proof the correctness of the algorithm without the need of using Gröbner basis. An upper bound on the weight of the codewords is imposed the algorithm in order to get only those leader codewords bounded by a given weight. We also show how can be used a suitable subset of the leader codewords in the 'Code Equivalence Problem', i.e. the problem of determining whether two given

E. Martínez-Moro—Partially supported by the Spanish State Research Agency (AEI) under Grants MTM2015-65764-C3-1, PGC2018-096446-B-C21.

D. Slamanig et al. (Eds.): MACIS 2019, LNCS 11989, pp. 218–233, 2020.
https://doi.org/10.1007/978-3-030-43120-4_17

linear codes are permutation-equivalent. If they are, we also want to recover this permutation group. In [10] the authors proved that this problem is not NP-complete but also that it is at least as hard as the Graph Isomorphism Problem. On the other hand, the support splitting algorithm [11] solves the computational version of the problem in polynomial time for all but an exponentially small proportion of the instances. In that paper it is stated that the main difficulty in the implementation of the algorithm lies in the choice of the invariant since usually the computation rapidly becomes intractable when its size grows.

Note that the role played by the Gröbner representation in the equivalence of codes was introduced in [4]. The set of leader codewords proposed in this paper is a structure which is considerably smaller that the invariant proposed in [4]. Despite of this, this set grows fast as the size of the code increase; so we impose an upper bound on the weight of the codewords to be included and prove that is enough to consider this subset of leader codewords as invariant. We use this subset for finding the permutation between equivalent codes. Note also that it can be used in any algorithm based on partitions and refinements like those in [8,11]. In particular, we have adapted the support splitting algorithm by defining a specific signature corresponding to this subset as invariant.

The structure of the paper is as follows. In Sect. 2 we present some preliminary facts and notations. In Sect. 3 we define the set of leader codewords and describe the algorithm. Section 4 provides a formal proof that this subset is an invariant for the code and we show how it can be used for finding the permutation group between equivalent codes. The algorithm is described and formalized in Sect. 5. Finally in Sect. 6 we present some experimental results.

2 Preliminaries

2.1 Linear Codes

From now on we shall denote by $\mathbb{F}_q$ the finite field with $q = p^m$ elements, p a prime. A *linear code* $\mathcal{C}$ over $\mathbb{F}_q$ of length n and dimension k is a k-dimensional subspace of $\mathbb{F}_q^n$. We will call the vectors $\mathbf{v}$ in $\mathbb{F}_q^n$ words and those $\mathbf{v} \in \mathcal{C}$, codewords. For every word $\mathbf{v} \in \mathbb{F}_q^n$ its *support* is defined as $\mathrm{supp}(\mathbf{v}) = \{i \mid v_i \neq 0\}$ and its *Hamming weight*, denoted by $\mathrm{w_H}(\mathbf{v})$ as the cardinality of $\mathrm{supp}(\mathbf{v})$ and the *Hamming distance* $\mathrm{d_H}(\mathbf{x}, \mathbf{y})$ between two words $\mathbf{x},\ \mathbf{y} \in \mathbb{F}_q^n$ is $\mathrm{d_H}(\mathbf{x}, \mathbf{y}) = \mathrm{w_H}(\mathbf{x} - \mathbf{y})$. The *minimum distance* $\mathrm{d}(\mathcal{C})$ of a linear code $\mathcal{C}$ is defined as the minimum weight among all nonzero codewords.

The set of words of minimal Hamming weight in all the cosets of $\mathbb{F}_q^n/\mathcal{C}$ is the *set of coset leaders* of the code $\mathcal{C}$ in $\mathbb{F}_q^n$ and we will denote it by $\mathrm{CL}(\mathcal{C})$. $\mathrm{CL}(\mathbf{y})$ will denote the subset of coset leaders corresponding to the coset $\mathbf{y}+\mathcal{C}$. Given a coset $\mathbf{y}+\mathcal{C}$ we define the *weight of the coset* $\mathrm{w_H}(\mathbf{y}+\mathcal{C})$ as the smallest Hamming weight among all vectors in the coset, or equivalently the weight of one of its leaders. It is well known that given $t = \lfloor \frac{d(\mathcal{C})-1}{2} \rfloor$ where $\lfloor \cdot \rfloor$ denotes the greatest integer function then every coset of weight at most t has a unique coset leader.

2.2 The Weak Order Ideal of the Coset Leaders

Let $f(X)$ be an irreducible polynomial over $\mathbb{F}_p$ of degree m and β be a root of $f(X)$, then any element $a \in \mathbb{F}_q$ can be represented as $a_1 + a_2\beta + \ldots + a_m\beta^{m-1}$ with $a_i \in \mathbb{F}_p$ for $i \in \{1, \ldots, m\}$.

Definition 1. *We define the* generalized support *of a vector* $\mathbf{v} = (\mathbf{v}_1, \ldots, \mathbf{v}_n) \in \mathbb{F}_q^n$ *as the support of the nm-tuple given by the concatenations of the p-adic expansion of each component* $\mathbf{v}_i = v_{i_1} + v_{i_2}\beta + \ldots + v_{i_m}\beta^{m-1}$ *of* $\mathbf{v}$. *That is* $\mathrm{supp}_{\mathrm{gen}}(\mathbf{v}) = (\mathrm{supp}((v_{i_1}, \ldots, v_{i_m})) : i = 1, \ldots, n)$, *and* $\mathrm{supp}_{\mathrm{gen}}(\mathbf{v})[i] = \mathrm{supp}((v_{i_1}, \ldots, v_{i_m}))$. *We will say that* $i_j \in \mathrm{supp}_{\mathrm{gen}}(\mathbf{v})$ *if the corresponding* v_{i_j} *is not zero.*

The set $\mathrm{Can}(\mathbb{F}_q, f) = \left\{\mathbf{e}_{ij} = \beta^{j-1}\mathbf{e}_i : i = 1, \ldots, n;\ j = 1, \ldots, m\right\}$ represents the canonical basis of $(\mathbb{F}_q^n, +)$. We state the following connection between $\mathbb{F}_q^n$ and $\mathbb{N}^{nm}$:

$$\begin{aligned} \Delta : \mathbb{F}_q^n &\to \mathbb{N}^{nm} \\ \mathbf{v} &\mapsto (\psi(v_{i_j}) : i = 1, \ldots, n,\ j = 1, \ldots, m), \end{aligned}$$

where the mapping $\psi : \mathbb{F}_p \to \mathbb{N}$ is defined as $k \cdot 1_{\mathbb{F}_p} \mapsto k \bmod p$. On the other hand we define the mapping $\nabla : \mathbb{N}^{nm} \to \mathbb{F}_q^n$ as $\mathbf{a} \mapsto (a_{m(i-1)+1} + a_{m(i-1)+2}\beta + \ldots + a_{m(i-1)+m}\beta^{m-1})$, $i = 1, \ldots, n$.

Definition 2. *Given* $\mathbf{x}, \mathbf{y} \in (\mathbb{F}_q^n, +)$, $\mathbf{x} = \sum_{i,j} x_{i_j}\mathbf{e}_{ij}$, $\mathbf{y} = \sum_{i,j} y_{i_j}\mathbf{e}_{ij}$, *we say* $\mathbf{x} \subset \mathbf{y}$ *if* $\psi(x_{i_j}) \leq \psi(y_{i_j})$ *for all* $i \in \{1, \ldots, n\}$ *and* $j \in \{1, \ldots, m\}$.

The map Δ relates orders on $\mathbb{F}_q^n$ with orders on $\mathbb{N}^{nm}$, and vice versa. An *admissible order* on $(\mathbb{N}^{nm}, +)$ is a total order $<$ on $\mathbb{N}^{nm}$ satisfying the following two conditions

1. $\mathbf{0} < \mathbf{x}$, for all $\mathbf{x} \in \mathbb{N}^{nm}$, $\mathbf{x} \neq \mathbf{0}$.
2. If $\mathbf{x} < \mathbf{y}$, then $\mathbf{x} + \mathbf{z} < \mathbf{y} + \mathbf{z}$, for all $\mathbf{z} \in \mathbb{N}^{nm}$.

In particular, any admissible order on $(\mathbb{N}^{nm}, +)$, (lexicographical, degree lexicographical, degree reverse lexicographical ...) induces an order on $(\mathbb{F}_q^n, +)$. A representation of a word $\mathbf{v}$ as an nm-tuple over $\mathbb{N}$ is said to be in *standard form* if $\Delta(\nabla(\mathbf{v})) = \mathbf{v}$. We will denote the standard form of $\mathbf{v}$ as $\mathrm{SF}(\mathbf{v}, f)$ (note that $\nabla(\mathbf{v}) = \nabla(\mathrm{SF}(\mathbf{v}, f))$). Therefore, $\mathbf{v}$ is in standard form if $\mathbf{v} = \mathrm{SF}(\mathbf{v}, f)$ (we will also say $\mathbf{v} \in \mathrm{SF}(\mathbb{F}_q^n, f)$). In shake of brevity, from now on we will consider the polynomial f fixed and we will use $\mathrm{Can}(\mathbb{F}_q)$ and $\mathrm{SF}(\mathbb{F}_q^n)$ instead of $\mathrm{Can}(\mathbb{F}_q, f)$ and $\mathrm{SF}(\mathbb{F}_q^n, f)$ respectively.

Definition 3. *A subset* $\mathcal{O}$ *of* $\mathbb{N}^k$ *is an order ideal if for all* $\mathbf{w} \in \mathcal{O}$ *and* $\mathbf{v} \in \mathbb{N}^k$ *s.t.* $\mathbf{v}_i \leq \mathbf{w}_i$, $i = 1, \ldots, k$, *then* $\mathbf{v} \in \mathcal{O}$.

In the same fashion as the previous definition, we say that a subset $\mathcal{S}$ of $\mathbb{F}_q^n$ is an order ideal if $\Delta(\mathcal{S})$ is an order ideal in $\mathbb{N}^{nm}$. It is easy to check that an equivalent definition for the order ideal would be that for all $\mathbf{w} \in \mathcal{S}$, and for all $i_j \in \mathrm{supp}_{\mathrm{gen}}(\mathbf{w})$, and $\mathbf{v} \in \mathbb{F}_q^n$ s.t. $\mathbf{w} = \mathbf{v} + \mathbf{e}_{ij}$ we have $\mathbf{v} \in \mathcal{S}$. If we change it slightly and instead of for all $i_j \in \mathrm{supp}_{\mathrm{gen}}(\mathbf{w})$ the condition is satisfied at least for one $i_j \in \mathrm{supp}_{\mathrm{gen}}(\mathbf{w})$ we say that the set $\mathcal{S}$ is a *weak order ideal*. More formally,

Definition 4. *A subset* $\mathcal{S}$ *of* $\mathbb{F}_q^n$ *is a weak order ideal if for all* $\mathbf{w} \in \mathcal{S} \setminus \mathbf{0}$ *there exists a* $i_j \in \mathrm{supp}_{\mathrm{gen}}(\mathbf{w})$ *such that for* $\mathbf{v} \in \mathbb{F}_q^n$ *with* $\mathbf{w} = \mathbf{v} + \mathbf{e}_{ij}$ *then* $\mathbf{v} \in \mathcal{S}$.

In the above situation we will say that the word $\mathbf{w}$ is an *ancestor* of the word $\mathbf{v}$, and that $\mathbf{v}$ is a *descendant* of $\mathbf{w}$. In non binary case a coset leader could be an ancestor of another coset leader or an ancestor of a word at Hamming distance 1 to a coset leader (this last case is not possible in the binary case).

The first idea that allows us to compute incrementally the set of all coset leaders for a linear code was introduced in [4] using the additive structure of $\mathbb{F}_q^n$ and the set of canonical generators $\mathrm{Can}(\mathbb{F}_q)$. Unfortunately in [4] most of the chosen coset representatives may not be coset leaders if the weight of the coset is greater than t. In order to incrementally generate all coset leaders starting from $\mathbf{0}$ adding elements in $\mathrm{Can}(\mathbb{F}_q)$, we must consider words with weight one more than the previous chosen coset leader (see [5]).

Definition 5. *Given* $\prec_1$ *an admissible order on* $(\mathbb{N}^{nm}, +)$ *we define the weight compatible order* $\prec$ *on* $(\mathbb{F}_q^n, +)$ *associated to* $\prec_1$ *as the ordering given by*

1. $\mathbf{x} \prec \mathbf{y}$ *if* $\mathrm{w_H}(\mathbf{x}) < \mathrm{w_H}(\mathbf{y})$ *or*
2. *if* $\mathrm{w_H}(\mathbf{x}) = \mathrm{w_H}(\mathbf{y})$ *then* $\Delta(\mathbf{x}) \prec_1 \Delta(\mathbf{y})$.

In other words, the words in $\mathbb{F}_q^n$ are ordered according their Hamming weights and the order $\prec_1$ break ties. These class of orders is a subset of the class of monotone α-orderings in [7]. In fact we will need a little more than monotonicity, we will also need the following condition: for every pair $\mathbf{v}, \mathbf{w} \in \mathrm{SF}(\mathbb{F}_q^n)$ such that $\mathbf{v} \subset \mathbf{w}$ one has that $\mathbf{v} \prec \mathbf{w}$. Note that this last condition is indeed true for a weight compatible order. In addition, for any weight compatible order $\prec$ every strictly decreasing sequence terminates (due to the finiteness of the set $\mathbb{F}_q^n$). In the binary case the behavior of the coset leaders can be translated to the fact that the set of coset leader is an order ideal of $\mathbb{F}_2^n$; whereas, for non binary linear codes this is no longer true even if we try to use the characterization of order ideals given in [6], where order ideals do not need to be associated with admissible orders.

Definition 6. *We define* the weak order ideal of the coset leaders *of a linear code* $\mathcal{C}$ *as the set* $\mathcal{O}(\mathcal{C})$ *of elements in* $\mathbb{F}_q^n$ *verifying the following items,*

1. $\mathbf{0} \in \mathcal{O}(\mathcal{C})$.
2. *If* $\mathbf{v} \in \mathcal{O}(\mathcal{C})$ *and* $\mathrm{w}_H(\mathbf{v}) = \mathrm{w}_H(\mathbf{v} + \mathcal{C})$ *then*
$$\left\{\mathbf{v} + \mathbf{e}_{ij} \mid \Delta(\mathbf{v}) + \Delta(\mathbf{e}_{ij}) \in \mathrm{SF}(\mathbb{F}_q^n)\right\} \subset \mathcal{O}(\mathcal{C}).$$
3. *If* $\mathbf{v} \in \mathcal{O}(\mathcal{C})$ *and* $\mathrm{w}_H(\mathbf{v}) = \mathrm{w}_H(\mathbf{v} + \mathcal{C}) + 1$ *then*
$$\left\{\mathbf{v} + \mathbf{e}_{ij} \mid i \in \mathrm{supp}(\mathbf{v}),\, \Delta(\mathbf{v}) + \Delta(\mathbf{e}_{ij}) \in \mathrm{SF}(\mathbb{F}_q^n)\,,\, \mathbf{v} - \mathbf{v}_i \in \mathrm{CL}(\mathcal{C})\right\} \subset \mathcal{O}(\mathcal{C}).$$

Note that it is clear by items 2 and 3 in the definition above that $\mathcal{O}(\mathcal{C})$ is a weak order ideal. Note also that the definition of the set $\mathcal{O}(\mathcal{C})$ also gives an algorithmic process to built this set, which result very important to construct the set $\mathrm{CL}(\mathcal{C})$ taking into account that $\mathrm{CL}(\mathcal{C}) \subset \mathcal{O}(\mathcal{C})$. The following two theorems show the connections between the set of coset leaders and the weak order ideal of the coset leaders.

Theorem 1 (See [5]). *Let* $\mathbf{w} \in \mathbb{F}_q^n$. *If there exists* $i \in 1, \ldots, n$ *s.t.* $\mathbf{w} - \mathbf{w}_i \in \mathrm{CL}(\mathcal{C})$ *then* $\mathbf{w} \in \mathcal{O}(\mathcal{C})$.

Theorem 2 (See [5]). *Let* $\mathbf{w} \in \mathbb{F}_q^n$ *and* $\mathbf{w} \in \mathrm{CL}(\mathcal{C})$ *then* $\mathbf{w} \in \mathcal{O}(\mathcal{C})$.

3 Leader Codewords of Linear Codes

Definition 7. *The set of* leader codewords *of a linear code* $\mathcal{C}$ *is defined as*

$$\mathrm{L}(\mathcal{C}) = \left\{ \begin{array}{c} \mathbf{v}_1 + \mathbf{e}_{ij} - \mathbf{v}_2 \in \mathcal{C} \setminus \{\mathbf{0}\} \mid \Delta(\mathbf{v}_1) + \Delta(\mathbf{e}_{ij}) \in \mathrm{SF}(\mathbb{F}_q^n), \\ \mathbf{v}_2 \in \mathrm{CL}(\mathcal{C}) \text{ and } \mathbf{v}_1 - \mathbf{v}_{1i} \in \mathrm{CL}(\mathcal{C}) \end{array} \right\}.$$

Note that the definition is a bit more elaborated that the one for binary codes in [3] due to the fact that in the general case not all coset leaders need to be ancestors of coset leaders. The name of leader codewords comes from the fact that one could compute all coset leaders of a corresponding word knowing the set $\mathrm{L}(\mathcal{C})$ adapting [3, Algorithm 3]. Theorem 1 guarantees that $\mathbf{w} \in \mathcal{O}(\mathcal{C})$ provided that $\mathbf{w} - \mathbf{w}_i \in \mathrm{CL}(\mathcal{C})$ for some i, then the associated set of leader codewords may be computed as $\{\mathbf{w} - \mathbf{v} : \mathbf{w} \in \mathcal{O}(\mathcal{C}), \mathbf{w} - \mathbf{w}_i \in \mathrm{CL}(\mathcal{C}), \mathbf{v} \in \mathrm{CL}(\mathbf{w}) \text{ and } \mathbf{v} \neq \mathbf{w}\}$.

3.1 Computing Algorithm

In [3] it is presented a Möller's like algorithm for computing the leader codewords for binary linear codes. Given a weight compatible ordering $\prec$, it is introduced an incremental form of generating the set of leader codewords. The generation of these elements is based on the construction of an object `List` (a crucial object in a Möller-like algorithm). The object `List` for general linear codes is related exactly with the computation of the set $\mathcal{O}(\mathcal{C})$; i.e. `List` is the smallest ordered set of elements in $\mathbb{F}_q^n$ verifying the following properties:

1. $\mathbf{0} \in$ `List`.
2. Criterion 1: If $\mathbf{v} \in$ `List` and $\mathrm{w}_H(\mathbf{v}) = \mathrm{w}_H(\mathbf{v} + \mathcal{C})$ then
$$\left\{\mathbf{v} + \mathbf{e}_{ij} \mid \Delta(\mathbf{v}) + \Delta(\mathbf{e}_{ij}) \in \mathrm{SF}(\mathbb{F}_q^n)\right\} \subset \texttt{List}.$$
3. Criterion 2: If $\mathbf{v} \in$ `List` and $\mathrm{w}_H(\mathbf{v}) = \mathrm{w}_H(\mathbf{v} + \mathcal{C}) + 1$ then
$$\left\{\mathbf{v} + \mathbf{e}_{ij} \mid i \in \mathrm{supp}(\mathbf{v}),\ \Delta(\mathbf{v}) + \Delta(\mathbf{e}_{ij}) \in \mathrm{SF}(\mathbb{F}_q^n),\right.$$
$$\left.\mathbf{v} - \mathbf{v}_i \in \mathrm{CL}(\mathcal{C})\right\} \subset \texttt{List}.$$

Given a weight compatible order $\prec$ and a linear code $\mathcal{C}$, the algorithm will incrementally generate all elements in `List` and also all coset leaders, starting from the zero codeword in `List`. Then Theorem 1 guarantees that

$$\mathbf{w} \in \texttt{List} \text{ provided that } \mathbf{w} - \mathbf{w}_i \in \mathrm{CL}(\mathcal{C}) \text{ for some } i, \tag{1}$$

and the associated set of leader codewords may be computed as $\{\mathbf{w} - \mathbf{v} : \mathbf{v} \in \mathrm{CL}(\mathbf{w}) \text{ and } \mathbf{v} \neq \mathbf{w}\}$.

3.2 Computing up to a Given Level

Let $\mathcal{Q}$ be a set of elements in $\mathbb{F}_q^n$. We will call a *level of weight* k to the set $\mathcal{Q}'$ such that $\mathcal{Q}' = \{\mathbf{v} \in \mathcal{Q} \mid \mathrm{w}_H(\mathbf{v}) = k\}$. We can get a partition of the set $\mathcal{Q}$ ordered by the weight of each level $0 \leq k_1 < k_2 < \ldots < k_s$. We will refer to the i-th set $\mathcal{Q}_i$ in this partition by the level of weight k_i of $\mathcal{Q}$ and we will denote as $\mathcal{Q}_{[i]}$ to the set of all words up to the level i.

As it was discussed in the previous section, the leader codewords of a linear code $\mathcal{C}$ are generated in an incremental form according to a weight compatible order, so we can set an upper bound if we only want the leader codewords up to a given level. The following proposition establishes a connection between the weight of the elements belonging to `List` and the weight of their corresponding leader codewords.

Proposition 1. *Let* $\mathbf{c} \in \mathrm{L}(\mathcal{C})$ *and* $\mathbf{w} \in$ `List` *the least element w.r.t to* $\prec$ *such that* $\mathbf{c} = \mathbf{w} - \mathbf{v}$, $\mathbf{w} - \mathbf{w}_i \in \mathrm{CL}(\mathcal{C})$ *for some* $i \in 1, \ldots, n$, $\mathbf{v} \in \mathrm{CL}(\mathcal{C})$. *Then* $2w_H(\mathbf{w}) - 1 \leq w_H(\mathbf{c})$.

Proof. Since $\mathbf{c} = \mathbf{w} - \mathbf{v}$ and $\mathbf{v} \in \mathrm{CL}(\mathcal{C})$, we have $\mathbf{w} \in \mathbf{v} + \mathcal{C}$. Then $\mathrm{w}_H(\mathbf{v}) \leq \mathrm{w}_H(\mathbf{w})$. If we suppose $\mathrm{w}_H(\mathbf{v}) = \mathrm{w}_H(\mathbf{w}) - 2$ then $\mathbf{c} = (\mathbf{w}_i - \mathbf{v}) - (-(\mathbf{w} - \mathbf{w}_i)) = \mathbf{a} - \mathbf{b}$, where $\mathbf{a} - \mathbf{a}_i = -\mathbf{v} \in \mathrm{CL}(\mathcal{C})$, $\mathbf{b} = -(\mathbf{w} - \mathbf{w}_i) \in \mathrm{CL}(\mathcal{C})$. Now, $\mathrm{w}_H(\mathbf{a}) \leq \mathrm{w}_H(\mathbf{w}_i) + \mathrm{w}_H(\mathbf{v}) = \mathrm{w}_H(\mathbf{w}) - 1$. This is $\mathrm{w}_H(\mathbf{a}) < \mathrm{w}_H(\mathbf{w})$ and so $\mathbf{a} \prec \mathbf{w}$. Finally, by (1), $\mathbf{a} - \mathbf{a}_i \in \mathrm{CL}(\mathcal{C})$ implies $\mathbf{a} \in$ `List`, which is a contradiction because $\mathbf{w}$ is the least element in `List` to obtain $\mathbf{c}$.

Therefore, $\mathrm{w}_H(\mathbf{v}) \geq \mathrm{w}_H(\mathbf{w}) - 1$, from where it is obtained $2\mathrm{w}_H(\mathbf{w}) - 1 \leq \mathrm{w}_H(\mathbf{c})$. □

Remark 1. As a direct consequence of the previous result we have that, in order to compute all leader codewords up to a weight k, it is enough to stop the algorithm in the first element of `List` of weight t such that $2t - 1 > k$.

Algorithm 1 below summarizes the aspects discussed above. There are three functions needed to understand the algorithm:

- InsertNexts[$\mathbf{t}$, `List`] inserts all sums $\mathbf{t} + \mathbf{e}_{ij}$ in `List`, where $\Delta(\mathbf{v}) + \Delta(\mathbf{e}_{ij}) \in \mathrm{SF}(\mathbb{F}_q^n)$, keeping the increasing order $\prec$ in `List`.
- NextTerm[`List`] returns the first element from `List` and deletes it from that set.
- Member[obj, G] returns the position j of obj in G, if $obj \in G$, and false otherwise.

Proposition 2. *Algorithm 1 computes the set of leader codewords of a linear code* $\mathcal{C}$ *up to a given level.*

Algorithm 1: Computation of the leader codewords up to a given level

input : A weight compatible ordering $\prec$, a parity check matrix H of a code $\mathcal{C}$ and the level k.
output: $\mathrm{L}(\mathcal{C})_{[k]}$.

```
List ← [0]; r ← 0; CL(C) ← ∅; S ← ∅; L(C) ← ∅; k' ← 0; w_{k'} ← 0;
w_k ← ∞; Stop ← false;
while List ≠ ∅ and Stop ≠ true do
    t ← NextTerm[List];
    if 2w_H(t) − 1 ≤ w_k then
        s ← tH^T;
        j ← Member[s, S];
        if j ≠ false then
            if w_H(t) = w_H(CL(C)[j][1]) then // Criterion 1 in List
                CL(C)[j] ← CL(C)[j] ∪ {t};
                List ← InsertNext[t, List];
            end if
            if w_H(t) = w_H(CL(C)[j][1]) + 1 then // Criterion 2 in
            List
                for i ∈ supp(t) : t − t_i ∈ CL(C) do
                    List ← InsertNext[t, List];
                end for
            end if
            for i ∈ supp(t) : t − t_i ∈ CL(C) do
                for t' ∈ CL(C)[j] and (t ≠ t') do
                    if w_H(t − t') > w_{k'} then
                        k' ← k' + 1; w_{k'} ← w_H(t − t');
                        if k' = k then // L(C) has reached the level k
                            w_k ← w_{k'};
                        end if
                    end if
                    if w_H(t − t') ≤ w_k then
                        L(C) ← L(C) ∪ {t − t'};
                    end if
                end for
            end for
        else
            r ← r + 1; CL(C)[r] ← {t}; S[r] ← s;
            List = InsertNext[t, List] // Criterion 1 in List;
        end if
    else
        Stop ← true;
    end if
end while
return L(C)
```

Proof (Of Proposition 2). Let us first prove that all the words inserted in `List` satisfy the desired properties pointed in Sect. 3.1. By Step 1, $\mathbf{0} \in$ `List`, verifying the first property, then in Step 5 the syndrome (an element of the coset) of $\mathbf{t} =$ `NextTerm`[`List`] is computed and based on the outcome of Step 6 we have two possible cases,

1. If $j =$ `false` then the coset $\mathcal{C} + \mathbf{t}$ has not yet been considered, therefore it is created taking $\mathbf{t}$ as a representative of minimal weight. Step 32 guarantees Criterion 1 in the second property.
2. On the other hand, if $j \neq$ `false`, the coset $\mathcal{C}+\mathbf{t}_j$ has been created and in case of $\mathrm{w}_H(\mathbf{t}) = \mathrm{w}_H(\mathbf{t}_j)$ Step 10 guarantees Criterion 1. If $\mathrm{w}_H(\mathbf{t}) = \mathrm{w}_H(\mathbf{t}_j) + 1$ then Step 13 and Step 14 verify Criterion 2 in the third property of `List`.

Therefore Algorithm 1 constructs `List` fulfilling the required properties. Furthermore, in `List` is included the set $\mathcal{O}(\mathcal{C})$, then by Theorem 2, `List` contains all coset leaders, thus Step 9 and Step 31 assure the computation of the whole set of coset leaders. From Step 19 to Step 24 the algorithm keeps track of the current level of $\mathrm{L}(\mathcal{C})$ and the weight associated with that level. Finally, Step 25 and Step 26 create the set $\mathrm{L}(\mathcal{C})$ of leader codewords according to Definition 7. Meanwhile, the second stop condition of the loop (Step 2) given by Proposition 1 prevents from continuing when the current weight is greater than the given weight for the desired level k. □

Of course note that if no level k is specified then Algorithm 1 computes the whole set of leader codewords.

4 $\mathrm{L}(\mathcal{C})$ as an Invariant for Linear Codes

It is clear that if two codes $\mathcal{C}, \mathcal{C}'$ are permutation equivalent so that for a given $\sigma \in S_n$ we have that $\mathcal{C}' = \sigma(\mathcal{C})$, then $\mathrm{L}(\mathcal{C}') = \sigma(\mathrm{L}(\mathcal{C}))$. In [4, Theorem 3] it is shown that two linear codes are equivalent if their so called Matphi structure are equivalent. These Matphi structures depend also on the cosets determined by the codes, but the size of this object is bigger than the set of leader codewords. The following result establishes that the set of leader codewords is also an invariant.

Theorem 3. *Let $\mathcal{C}$, $\mathcal{C}'$ be linear codes and $\sigma \in S_n$. Then $\mathcal{C}' = \sigma(\mathcal{C})$ if and only if $\mathrm{L}(\mathcal{C}') = \sigma(\mathrm{L}(\mathcal{C}))$.*

Proof. Let $\mathcal{C}' = \sigma(\mathcal{C})$ for $\sigma \in S_n$, in order to prove $\mathrm{L}(\mathcal{C}') = \sigma(\mathrm{L}(\mathcal{C}))$ it is enough to prove that $\sigma(\mathrm{L}(\mathcal{C})) \subset \mathrm{L}(\mathcal{C}')$. Let $\mathbf{c} \in \mathrm{L}(\mathcal{C})$, then $\mathbf{c} = \mathbf{v}_1 + \mathbf{e}_{ij} - \mathbf{v}_2$, $\Delta(\mathbf{v}) + \Delta(\mathbf{e}_{ij}) \in \mathrm{SF}(\mathbb{F}_q^n)$, $\mathbf{v}_2 \in \mathrm{CL}(\mathcal{C})$ and $\mathbf{v}_1 - \mathbf{v}_{1i} \in \mathrm{CL}(\mathcal{C})$. Thus, $\mathcal{C}' = \sigma(\mathcal{C})$ implies $\sigma(\mathbf{v}_1) - \sigma(\mathbf{v}_{1i}) = \sigma(\mathbf{v}_1 - \mathbf{v}_{1i}) \in \mathrm{CL}(\mathcal{C}')$, $\sigma(\mathbf{v}_2) \in \mathrm{CL}(\mathcal{C}')$ and $\Delta(\sigma(\mathbf{v}_1)) + \Delta(\mathbf{e}_{\sigma(i)j}) \in \mathrm{SF}(\mathbb{F}_q^n)$. Then, $\mathbf{c}' = \sigma(\mathbf{c}) = \sigma(\mathbf{v}_1) + \sigma(\mathbf{e}_{ij}) - \sigma(\mathbf{v}_2) = \sigma(\mathbf{v}_1) + \mathbf{e}_{\sigma(i)j} - \sigma(\mathbf{v}_2)$. Therefore, $\mathbf{c}' \in \mathrm{L}(\mathcal{C}')$.

Now, let us suppose that $\mathrm{L}(\mathcal{C}') = \sigma(\mathrm{L}(\mathcal{C}))$ and let $\mathbf{c} \in \mathcal{C}$. In [5] it was proved that the set $\mathrm{L}(\mathcal{C})$ is a test set for $\mathcal{C}$. This means, there exist $\mathbf{c}_1, \ldots, \mathbf{c}_k$, $\mathbf{c}_i \in LC$, $i = 1, \ldots, k$ such that $\mathbf{c} = \mathbf{c}_1 + \ldots + \mathbf{c}_k$. That is, we have

$$\sigma(\mathbf{c}) = \sigma(\mathbf{c}_1) + \ldots + \sigma(\mathbf{c}_k). \tag{2}$$

But $\sigma(\mathbf{c}_i) \in \mathrm{L}(\mathcal{C}')$, $i = 1, \ldots, k$, then, taking into account (2) we obtain $\sigma(\mathbf{c}) \in \mathcal{C}'$. □

Remark 2. A mapping is an invariant for a code means that it remains invariant under a permutation. The previous theorem shows that the set of leader codewords $\mathrm{L}(\mathcal{C})$ may give a very strong invariant in the sense that it is preserved if and only if the codes are equivalent. Due to its prohibitive size as the code length increases we take the subset $\mathrm{L}(\mathcal{C})_{[2]}$ and for this we have $\mathrm{L}(\mathcal{C}')_{[2]} = \sigma(\mathrm{L}(\mathcal{C})_{[2]})$ provided that $\mathcal{C}' = \sigma(\mathcal{C})$.

The following lemma allow us to state Theorem 4 in order to use the set of leader codewords up to a given level as invariant.

Lemma 1. *Let $\mathcal{C} = \langle \mathcal{B} \rangle$ and $\mathcal{C}' = \langle \mathcal{B}' \rangle$ two codes over $\mathbb{F}_q$ with spanning sets $\mathcal{B}$ and $\mathcal{B}'$. If there exists $\sigma \in S_n$ such that $\mathcal{B}' = \sigma(\mathcal{B})$ then $\mathcal{C}' = \sigma(\mathcal{C})$.*

Proof. Let $c' \in \mathcal{C}'$. Then $c' = \sum_{\alpha_i \in \mathbb{F}_q} \alpha_i \beta_i' = \sum \alpha_i \sigma(\beta_i) = \sigma\left(\sum \alpha_i \beta_i\right)$ and hence $c' \in \sigma(\mathcal{C})$. On the other hand, let $c \in \mathcal{C}$. Then $c = \sum_{\alpha_i \in \mathbb{F}_q} \alpha_i \beta_i$ and $\sigma(c) = \sum \alpha_i \sigma(\beta_i) = \sum \alpha_i \beta_i'$. Therefore $\sigma(c) \in \mathcal{C}'$. □

Theorem 4. *Let $\mathcal{C}$ and $\mathcal{C}'$ linear codes of $\mathbb{F}_q^n$ such that $dim(\mathcal{C}) = dim(\mathcal{C}')$ and $k = \min_s\{s \in \mathbb{N} \mid \mathcal{C} = \langle \mathrm{L}(\mathcal{C})_{[s]} \rangle\}$. For $m \geq k$, for any $\sigma \in S_n$ such that $\sigma\left(\mathrm{L}(\mathcal{C})_{[m]}\right) = \mathrm{L}(\mathcal{C}')_{[m]}$ then $\mathcal{C}' = \sigma(\mathcal{C})$.*

Proof. It is a consequence of the fact that $\mathrm{L}(\mathcal{C})_{[m]}$ is a spanning set of $\mathcal{C}$ for $m \geq k$, Lemma 1 and $\dim(\mathcal{C}) = \dim(\mathcal{C}')$. Note that, by applying the lemma, $\mathcal{C} \sim \sigma(\mathcal{C})$. On the other hand, $\sigma(\mathcal{C})$ is a subspace of $\mathcal{C}'$ of the same dimension of $\mathcal{C}'$, so $\sigma(\mathcal{C}) = \mathcal{C}'$. □

Note that all codewords of minimum weight are leader codewords. Moreover, $L(C)_{[1]}$ is exactly this set of codewords. In case of codes that are generated by this set, $k = 1$ in Theorem 4 and it is enough to use $L(C)_{[1]}$ to compute the candidates permutations.

5 Finding the Permutation

The idea of using the subset $\mathrm{L}(\mathcal{C})_{[2]}$ of the set $\mathrm{L}(\mathcal{C})$ as an invariant can be applied for finding the permutations between equivalent codes and it can be used in any algorithm based on partitions and refinements like [8,11]. In particular, we have specified the algorithm described in [11] by defining a specific signature corresponding to $\mathrm{L}(\mathcal{C})_{[2]}$. We have changed a little the definition of signature but keeping the central idea. The construction of the partition and the refinement process based on the signature follow similar procedures. A description of related algorithms for code equivalence is done in [12].

5.1 The Proposed Signature

One way of defining signatures for codes is by using an invariant, we are going to introduce a signature based on the set $\mathrm{L}(\mathcal{C})_{[2]}$.

Definition 8 ([11]). *A signature S over a set Ω maps a code $\mathcal{C}$ of length n and an element $i \in I_n = \{1, \ldots, n\}$ into an element of Ω and is such that for all permutations $\sigma \in S_n$, $S(\mathcal{C}, i) = S(\sigma(\mathcal{C}), \sigma(i))$.*

Let $\mathbb{Z}[y_0, \ldots, y_n]$ be the polynomial ring of the $n+1$ variables $y_0, \ldots, y_n$ over the integers. We define a signature over $\Omega = \mathbb{Z}[y_0, \ldots, y_n] \times \mathbb{Z}[y_0, \ldots, y_n]$ which depends on the numbers of assignments of positions already done. Note at the beginning no assignment has been done yet.

Let $J \subset I_n$, $J = \{j_1, \ldots, j_s\}$ be the assignments of positions we assumed have been done to the set $J' \subset I_n$, $J' = \{j'_1, \ldots, j'_s\}$, J may be equal to the empty set and $s = 0$. Then for all permutations $\sigma \in S_n$, such that $\sigma(j_i) = j'_i$, $i = 1, \ldots, s$, we define for $i \in I_n \setminus J$

$$\mathrm{SLC}_s(\mathcal{C}, i) = (a_{i0}y_0 + \ldots + a_{is}y_0y_1 \cdots y_s, b_{i0}y_0 + \ldots + b_{is}y_0y_1 \cdots y_s),$$

where the first component $a_{i0}y_0 + \ldots + a_{is}y_0y_1 \cdots y_s$ stands for the subset $\mathrm{L}(\mathcal{C})_1$ of $\mathrm{L}(\mathcal{C})_{[2]}$ and the second component $b_{i0}y_0 + \ldots + b_{is}y_0y_1 \cdots y_s$ stands for the subset $\mathrm{L}(\mathcal{C})_2$ of $\mathrm{L}(\mathcal{C})_{[2]}$. Specifically a_{ik}, $k \in 0, \ldots, s$ means that there are a_{ik} elements $\mathbf{c} \in \mathrm{L}(\mathcal{C})_1$ with $\mathbf{c}_i \neq 0$ and others exactly k positions from J which are not zero. Similarly, b_{ik}, $k \in 0, \ldots, s$ means that there are b_{ik} elements $\mathbf{c} \in \mathrm{L}(\mathcal{C})_2$ with $\mathbf{c}_i \neq 0$ and other exactly k positions from J which are not zero.

Note that $\mathrm{SLC}_s(\mathcal{C}, i)$ counts the interactions between the position i and the set of positions J already assigned in the subsets $\mathrm{L}(\mathcal{C})_1$ and $\mathrm{L}(\mathcal{C})_2$ of $\mathrm{L}(\mathcal{C})_{[2]}$. As it is expected, for all permutations $\sigma \in S_n$, such that $\sigma(j_i) = j'_i$, $i = 1, \ldots, s$, $\mathrm{SLC}_s(\mathcal{C}, i) = \mathrm{SLC}_s(\sigma(\mathcal{C}), \sigma(i))$ which is guaranteed by Theorem 3.

The $(\mathcal{C}, \mathrm{SLC}_s)$-partition (see [11]) is $\mathcal{P}(\mathcal{C}, \mathrm{SLC}_s) = \{J_e : e \in \Omega\}$, where $J_e = \{i \in I_n \setminus J : \mathrm{SLC}_s(\mathcal{C}, i) = e\}$. Note that the partition corresponding to a permutation is such that $\mathcal{P}(\sigma(\mathcal{C}), \mathrm{SLC}_s) = \{\sigma(J_e) : e \in \Omega\}$, for all permutations $\sigma \in S_n$, such that $\sigma(j_i) = j'_i$, $i = 1, \ldots, s$.

5.2 Refining the Partition

Given the linear codes $\mathcal{C}$ and $\mathcal{C}'$ and a subset of s positions J already assigned to J', such that $\mathrm{SLC}_i(\mathcal{C}, j_{i+1}) = \mathrm{SLC}_i(\mathcal{C}', j'_{i+1})$, $i = 0, \ldots, s-1$.

We compute the partitions $\mathcal{P}(\mathcal{C}, \mathrm{SLC}_s) = \{J_e : e \in \Omega\}$ and $\mathcal{P}(\mathcal{C}', \mathrm{SLC}_s) = \{J'_e : e \in \Omega\}$ and then we take into account that a position from J_e must be transformed into a position of J'_e, so the next assignment is decided. For example, we may take the J_{e_1} subset of minimal cardinal and then the position $i \in J_{e_1}$ of minimal absolute value. Once a new position is chosen we select its image j such that $\mathrm{SLC}_s(\mathcal{C}', j) = \mathrm{SLC}_s(\mathcal{C}, i)$ and then $J = J \cup \{i\}$, $J' = J' \cup \{j\}$, $s = s + 1$.

In this process it is possible to detect some contradictions which means no permutation will be found by this path. For example, it is clear that the cardinal of the partitions for $\mathcal{C}$ and $\mathcal{C}'$ must be the same, and also $\mid J_e \mid = \mid J'_e \mid$ for all $e \in \Omega$.

5.3 Computing Algorithm

Proposition 3. *Algorithm 2 computes a permutation $\sigma \in S_n$ between the codes $\mathcal{C}$ and $\mathcal{C}'$, that is $\mathcal{C}' = \sigma(\mathcal{C})$. If no permutation is found then these codes are not permutation equivalent.*

Algorithm 2: Computing the permutation

function `PermutationEquivCodes`
  input : $\mathcal{C}, \mathcal{C}'$ and a weight compatible ordering $\prec$
  output: A permutation $\sigma \in S_n$, such that $\mathcal{C}' = \sigma(\mathcal{C})$
  $J \leftarrow \emptyset$; $J' \leftarrow \emptyset$; $s \leftarrow 0$
  Compute $\mathrm{L}(\mathcal{C})_{[2]}$ and $\mathrm{L}(\mathcal{C}')_{[2]}$ using $\prec$ as described in Section 3
  `FindPermutation`$(\mathrm{L}(\mathcal{C})_{[2]}, \mathrm{L}(\mathcal{C}')_{[2]}, J, J', s)$
  if no permutation found **then**
    return $\mathcal{C}, \mathcal{C}'$ are not permutation equivalent codes
  end if
  return $\sigma \leftarrow \sigma(J_i) = J'_i$, $i = 1, \ldots, n$
end func

function `FindPermutation`
  input : $\mathrm{L}(\mathcal{C})_{[2]}, \mathrm{L}(\mathcal{C}')_{[2]}$ and $J, J' \subset I_n$ where $s = |J| = |J'|$
  output: J, J' such that $\sigma(J_i) = J'_i$, $i = 1, \ldots, n$ if a permutation is found. No permutation found is returned otherwise
  if $|J| = n$ **then**
    return J, J'
  end if
  $\mathcal{P} \leftarrow \{J_e : \mathrm{SLC}_s(\mathcal{C}, i) = e,\ i \in I_n \setminus J,\ e \in \Omega\}$ `// use` $\mathrm{L}(\mathcal{C})_{[2]}$ `as invariant`
  $\mathcal{P}' \leftarrow \{J'_e : \mathrm{SLC}_s(\mathcal{C}', i) = e,\ i \in I_n \setminus J',\ e \in \Omega\}$ `// to compute` $\mathrm{SLC}_s(\mathcal{C}, i)$
  if $|J_e| = |J'_e|$ for all $J_e \in \mathcal{P}, J'_e \in \mathcal{P}'$ **then**
    $J \leftarrow J \cup \{i\}, i \in J_{e_1}$ `//` J_{e_1}, i `chosen randomly or by an heuristic`
    $J^* \leftarrow J'_{e_1}$ such that $J'_{e_1} \in \mathcal{P}'$
    while no permutation found **and** $J^* \neq \emptyset$ **do**
      $J' \leftarrow J' \cup \{j\}$, $j \in J^*$
      `FindPermutation`$(\mathrm{L}(\mathcal{C})_{[2]}, \mathrm{L}(\mathcal{C}')_{[2]}, J, J', s+1)$
      if a permutation were found **then**
        return J, J'
      end if
      $J' \leftarrow J' \setminus \{j\}$; $J^* \leftarrow J^* \setminus \{j\}$
    end while
    $J \leftarrow J \setminus \{i\}$
  end if
  return no permutation found
end func

Proof (Of Proposition 3). It is clear that if there exists a permutation between two codes $\mathcal{C}$ and $\mathcal{C}'$, by Theorem 3, this permutation transforms $\mathrm{L}(\mathcal{C})$ into $\mathrm{L}(\mathcal{C}')$ and then defines the same signatures and partitions. Thus one of those permutations will be found by Algorithm 2. The process is finite because there is a finite number of permutations and therefore the process of analyzing different assignments following the signatures and partitions is finite. □

Note that in Algorithm 2 the function `PermutationEquivCodes` do the initializations. Then the sets $\mathrm{L}(\mathcal{C})_{[2]}$ and $\mathrm{L}(\mathcal{C}')_{[2]}$ are computed (also they can be loaded from a precomputed database) and then a call to the recursive function `FindPermutation` is made which follows a refinement process following an n-ary tree structure, where a permutation is found when a node of level n is reached.

Finding All the Permutations. Given a linear code $\mathcal{C}$ of length n, the subgroup of all elements σ of S_n such that $\sigma(\mathcal{C}) = \mathcal{C}$ is called the permutation automorphism group of $\mathcal{C}$. Note that if the permutation automorphism group is nontrivial and if $\mathcal{C}'$ is permutation equivalent to $\mathcal{C}$, then several permutations could satisfy $\mathcal{C}' = \sigma(\mathcal{C})$. Algorithm 2 can be easily modified to compute all those permutations. This can be achieved if a list of pairs J, J' is returned instead of a single pair and before each successful return statement those pairs are added to this list. An n-ary tree transversal is made, where at each node of level n one permutation is considered. After that step an expurgation process should be carried out since some invalid permutations could be introduced because $\sigma(\mathrm{L}(\mathcal{C})_{[2]}) = \mathrm{L}(\mathcal{C}')_{[2]}$ may not be sufficient to guarantee that $\sigma(\mathcal{C}) = \mathcal{C}'$.

Example 1 (Toy Example). Consider the binary codes $\mathcal{C} = \langle (0,1,0,0,1), (1,1,0,1,0), (0,1,1,0,0)\rangle$ and $\mathcal{C}' = \langle (1,1,0,0,0), (1,0,1,0,1), (1,0,0,1,0)\rangle$ and

$$\begin{aligned}
\mathrm{L}(\mathcal{C})_1 &= \{(0,1,1,0,0), (0,1,0,0,1), (0,0,1,0,1)\},\\
\mathrm{L}(\mathcal{C})_2 &= \{(1,1,0,1,0), (1,0,1,1,0), (1,0,0,1,1)\},\\
\mathrm{L}(\mathcal{C}')_1 &= \{(1,1,0,0,0), (1,0,0,1,0), (0,1,0,1,0)\},\\
\mathrm{L}(\mathcal{C}')_2 &= \{(1,0,1,0,1), (0,1,1,0,1), (0,0,1,1,1)\}.
\end{aligned}$$

Note that $\mathcal{C}' = \sigma(\mathcal{C})$ with $\sigma = (1,3,4,5,2)$ and thus $\mathrm{L}(\mathcal{C}')_{[2]} = \sigma(\mathrm{L}(\mathcal{C})_{[2]})$. On the first call to `FindPermutation` we get

$$\begin{aligned}
&\mathrm{SLC}_0(\mathcal{C},1) = (0,3y_0), \quad \mathrm{SLC}_0(\mathcal{C},2) = (2y_0,y_0),\\
&\mathrm{SLC}_0(\mathcal{C},3) = (2y_0,y_0), \quad \mathrm{SLC}_0(\mathcal{C},4) = (0,3y_0), \quad \mathrm{SLC}_0(\mathcal{C},5) = (2y_0,y_0),\\
&\mathrm{SLC}_0(\mathcal{C}',1) = (2y_0,y_0), \quad \mathrm{SLC}_0(\mathcal{C}',2) = (2y_0,y_0),\\
&\mathrm{SLC}_0(\mathcal{C}',3) = (0,3y_0), \quad \mathrm{SLC}_0(\mathcal{C}',4) = (2y_0,y_0), \quad \mathrm{SLC}_0(\mathcal{C}',5) = (0,3y_0).
\end{aligned}$$

Now with $e_1 = (0,3y_0), e_2 = (2y_0,y_0)$ such that $e_1, e_2 \in \Omega$ we get

$$\begin{aligned}
\mathcal{P}(\mathcal{C},\mathrm{SLC}_0) &= \{J_{e_1} = \{1,4\}, J_{e_2} = \{2,3,5\}\},\\
\mathcal{P}'(\mathcal{C}',\mathrm{SLC}_0) &= \{J'_{e_1} = \{3,5\}, J'_{e_2} = \{1,2,4\}\}.
\end{aligned}$$

At this point is verified that $|J_{e_1}| = |J'_{e_1}|$ and $|J_{e_2}| = |J'_{e_2}|$ and a coordinate must be chosen, that is, $i \in J_e$ such that $J_e \in \mathcal{P}$. Recall that this can be done at random or following some heuristics. Let us take the one with minimal cardinal and the position with minimal absolute value. Then $J = J \cup \{1\} \Rightarrow J = \{1\}$ and we may try with each element in J'_{e_1}, since $\sigma(1) \in J'_{e_1}$ provided that $\mathrm{SLC}_0(\mathcal{C}, 1) = \mathrm{SLC}_0(\mathcal{C}', \sigma(1))$. Starting with the minimum value $J' = J' \cup \{3\} \Rightarrow J' = \{3\}$, then a recursive call is made, meaning that $\sigma(1) = 3$. Note that if no permutation is found through this path, a new selection must be made following a tree structure. In the new call we get

$$\begin{array}{ll} \mathrm{SLC}_1(\mathcal{C}, 2) = (0, 2y_0 + y_0y_1), & \mathrm{SLC}_1(\mathcal{C}, 3) = (0, 2y_0 + y_0y_1), \\ \mathrm{SLC}_1(\mathcal{C}, 4) = (0, 3y_0y_1), & \mathrm{SLC}_1(\mathcal{C}, 5) = (0, 2y_0 + y_0y_1), \\ \mathrm{SLC}_1(\mathcal{C}', 1) = (0, 2y_0 + y_0y_1), & \mathrm{SLC}_1(\mathcal{C}', 2) = (0, 2y_0 + y_0y_1), \\ \mathrm{SLC}_1(\mathcal{C}', 4) = (0, 2y_0 + y_0y_1), & \mathrm{SLC}_1(\mathcal{C}', 5) = (0, y_0y_1), \end{array}$$

$$\mathcal{P}(\mathcal{C}, \mathrm{SLC}_1) = \{J_{e_1} = \{4\}, J_{e_2} = \{2, 3, 5\}\},$$
$$\mathcal{P}'(\mathcal{C}', \mathrm{SLC}_1) = \{J'_{e_1} = \{5\}, J'_{e_2} = \{1, 2, 4\}\},$$

where $J = J \cup \{4\} \Rightarrow J = \{1, 4\}$ and $J' = J' \cup \{5\} \Rightarrow J' = \{3, 5\}$, meaning that $\sigma(1) = 3$ and $\sigma(4) = 5$. Following this refining procedure is obtained J and J' such that $\sigma(j_i) = j'_i$, $i = 1, \ldots, 5$.

6 Experimental Results

The algorithms in this paper were implemented in C++ using the GNU operating system based gcc compiler and performed using the high performance computing capabilities provided at University of Oriente, Cuba (http://www.uo.edu.cu). In Table 1 we show the advantage of choosing the set of leader codewords only up to the second level. A significant difference can be noticed in the execution time, since for level 2 there is a relatively slight change as the number of cosets increase,

Table 1. Execution time and number of leader codewords

Codes	# Cosets	Level 2		All levels	
		Time (sec.)	Num.	Time (sec.)	Num.
$\mathbb{F}_2\,[12, 6, 4]$	64	0.0001	28	0.0310	50
$\mathbb{F}_2\,[15, 8, 3]$	128	0.0001	9	0.1400	127
$\mathbb{F}_2\,[18, 10, 3]$	256	0.0150	16	0.6560	328
$\mathbb{F}_3\,[12, 6, 3]$	729	0.0620	16	6.9690	568
$\mathbb{F}_3\,[12, 5, 3]$	2187	0.0620	12	42.2510	472
$\mathbb{F}_3\,[13, 6, 4]$	2187	0.1250	28	57.8660	520
$\mathbb{F}_4\,[11, 5, 4]$	4096	0.2970	42	271.0270	816
$\mathbb{F}_4\,[12, 6, 3]$	4096	0.4100	15	401.0000	2435
$\mathbb{F}_4\,[14, 7, 4]$	16384	1.1720	27	10942.7280	4564

compared with the fast growth in computing the whole set. On the other hand, the number of leader codewords up to the second level remains stable and much more smaller (see Fig. 1).

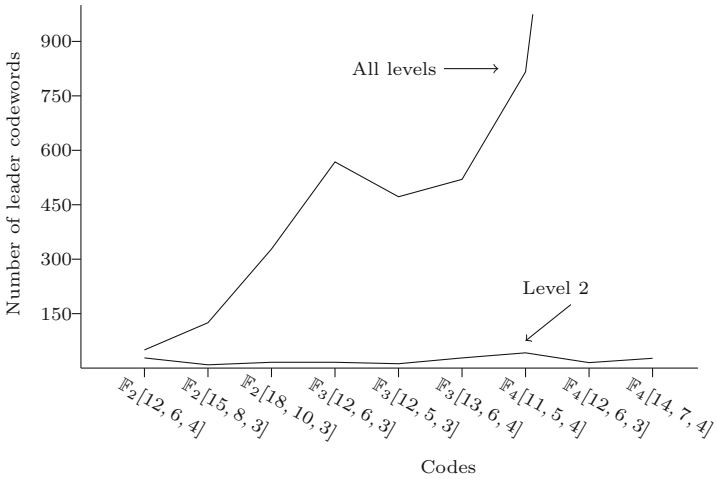

Fig. 1. Number of leader codewords up to the 2nd and all levels

In order to evaluate the performance of the algorithm for finding the permutation between two linear codes, a code is generated first at random and then is applied a permutation generated at random too. For these two codes we compute the set of the leader codewords up to the second level, and then, they are used as input for the algorithm that will give as output the first valid permutation.

Table 2 is shows execution times for the leader codewords up to the second level, only for the generated code, and in a different column is showed the time

Table 2. Execution time in seconds to find the first permutation between two linear codes randomly permuted

Codes ($\mathcal{C}$)	Cosets	$\mathrm{L}(\mathcal{C})_{[2]}$	1st permutation
$\mathbb{F}_2\,[15, 7, 3]$	256	0.031	0.015
$\mathbb{F}_2\,[21, 12, 3]$	512	0.093	0.047
$\mathbb{F}_2\,[29, 18, 3]$	2048	0.328	0.078
$\mathbb{F}_2\,[34, 19, 4]$	32768	12.688	0.063
$\mathbb{F}_3\,[18, 8, 4]$	59049	22.891	0.078
$\mathbb{F}_3\,[23, 12, 5]$	177147	96.395	0.078
$\mathbb{F}_3\,[32, 20, 5]$	531441	567.770	0.240
$\mathbb{F}_4\,[20, 10, 5]$	1048576	647.020	0.060
$\mathbb{F}_4\,[26, 15, 5]$	4194304	3001.800	0.360
$\mathbb{F}_4\,[30, 18, 5]$	16777216	9402.400	0.140

Table 3. Execution time in seconds to find all permutations

Codes	Permutations	First (sec.)	All (sec.)
$\mathbb{F}_2\,[15, 7, 3]$	8	0.015	0.031
$\mathbb{F}_2\,[29, 18, 3]$	96	0.078	0.265
$\mathbb{F}_2\,[21, 12, 3]$	144	0.047	0.271
$\mathbb{F}_4\,[30, 18, 5]$	720	0.140	0.830
$\mathbb{F}_2\,[15, 7, 3]$	768	0.015	1.218
$\mathbb{F}_3\,[18, 8, 4]$	1536	0.015	2.828
$\mathbb{F}_3\,[23, 12, 4]$	3456	0.042	8.636
$\mathbb{F}_2\,[34, 19, 4]$	4608	0.063	9.000
$\mathbb{F}_4\,[30, 18, 5]$	10080	0.024	9.376

consumed by Algorithm 1. The codes are generated increasing the number of cosets as before, but this time these numbers are much more greater, showing the advantage of using the selected invariant $\mathrm{L}(\mathcal{C})_{[2]}$. Note that the time used by the algorithm to get the first correct permutation is significantly smaller than the time spent in computing the invariant.

Finally Table 3 shows the execution times for the algorithm adapted to compute all the permutations and it is compared with the timing of the first permutation obtained. The time for finding the first permutation depends on how the elements are chosen in each refinement stage. This explain the fluctuating time according to the increasing number of elements in the permutation group.

Some Comments on Complexity Issues. Authors would like to emphasize that the main goal of this paper is the study the computation of leader codewords and the properties related to the permutation equivalent problem from the mathematical point of view. Some of the experiments are devoted to show the possibility of using part of the set of leader codewords instead of the whole set and to compare this two instance.

Algorithm 1 for computing the leader codewords is efficient because its computational complexity is linear on the size of the weak order ideal of the code, and because of the nature of the leader codewords this can not be improved much more. Also we adapted the algorithm to compute the set up to a given weight. Anyway the computation of this set becomes intractable when the code length increase, particularly, the redundancy of the code (the number of cosets).

On the other hand, we used the SSA Algorithm and we construct with the leader codewords a signature in order to use the scheme of this algorithm. The main limitation is the high complexity of computing the invariant, the set of leader codewords up to a given level. For complexity issues regarding the SSA Algorithm and other problems related with the code equivalence, we recommend [13], which examines also complexity issues of SSA, [1] it can be used as a comparison for works related to this problem, [2], useful reference and a standard in these works to compare against.

References

1. Babai, L., Codenotti, P., Grochow, J.A., Qiao, Y.: Code equivalence and group isomorphism. In: Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 1395–1408. Society for Industrial and Applied Mathematics (2011)
2. Betten, A., Braun, M., Fripertinger, H., Kerber, A., Kohnert, A., Wassermann, A.: Error-Correcting Linear Codes: Classification by Isometry and Applications, vol. 18. Springer, Heidelberg (2006). https://doi.org/10.1007/3-540-31703-1
3. Borges-Quintana, M., Borges-Trenard, M., Márquez-Corbella, I., Martínez-Moro, E.: Computing coset leaders and leader codewords of binary codes. J. Algebra Appl. **14**(8), 19 (2015)
4. Borges-Quintana, M., Borges-Trenard, M., Martínez-Moro, E.: On a Gröbner bases structure associated to linear codes. J. Discrete Math. Sci. Cryptogr. **10**(2), 151–191 (2007)
5. Borges-Quintana, M., Borges-Trenard, M., Martínez-Moro, E.: On the weak order ideal associated to linear codes. Math. Comput. Sci. **12**(3), 339–347 (2018)
6. Braun, G., Pokutta, S.: A polyhedral characterization of border bases. SIAM J. Discrete Math. **30**(1), 239–265 (2016)
7. Helleseth, T., Kløve, T., Levenshtein, V.I.: Error-correction capability of binary linear codes. IEEE Trans. Inf. Theory **51**(4), 1408–1423 (2005)
8. Leon, J.S.: Computing automorphism groups of error-correcting codes. IEEE Trans. Inform. Theory **28**, 496–511 (1982)
9. Mora, T.: Solving Polynomial Equation Systems II: Macaulay's Paradigm and Gröbner Technology. Cambridge University Press, Cambridge (2005)
10. Petrank, E., Roth, R.: Is code equivalence easy to decide? IEEE Trans. Inform. Theory **43**(5), 1602–1604 (1997)
11. Sendier, N.: Finding the permutation between equivalent linear codes: the support splitting algorithm. IEEE Trans. Inform. Theory **46**(4), 1193–1203 (2000)
12. Sendier, N., Simos, D.: How easy is code equivalence over $\mathbb{F}_q$? In: International Workshop on Coding and Cryptography (2013). https://www.rocq.inria.fr/secret/PUBLICATIONS/codeq3.pdf
13. Sendrier, N., Simos, D.E.: The hardness of code equivalence over $\mathbb{F}_q$ and its application to code-based cryptography. In: Gaborit, P. (ed.) PQCrypto 2013. LNCS, vol. 7932, pp. 203–216. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38616-9_14