mathematics

MDPI

Article

How to Find the Equivalence Classes in a Set of Linear Codes in Practice?

Stefka Bouyuklieva [1,*] and Iliya Bouyukliev [2]

[1] Faculty of Mathematics and Informatics, St. Cyril and St. Methodius University of Veliko Tarnovo, 5000 Veliko Tarnovo, Bulgaria
[2] Institute of Mathematics and Informatics, Bulgarian Academy of Sciences, 5000 Veliko Tarnovo, Bulgaria; iliyab@math.bas.bg
* Correspondence: stefka@ts.uni-vt.bg

Abstract: An algorithm for equivalence of linear codes over finite fields is presented. Its main advantage is that it can extract exactly one representative from each equivalence class among a large number of linear codes. It can also be used as a test for isomorphism of binary matrices. The algorithm is implemented in the program LCEQUIVALENCE, which is designed to obtain the inequivalent codes in a set of linear codes over a finite field with $q < 64$ elements. This program is a module of the free software package QEXTNEWEDITION for constructing, classifying and studying linear codes.

Keywords: linear code; code equivalence; canonical form; graph isomorphism problem

MSC: 94B05; 05A18

1. Introduction

Solving classification problems is of interest for both practical and theoretical reasons. Already, Felix Klein in his Erlanger Programm (1872) [1] described mathematics as the study of properties of sets that remain invariant under certain specified groups of transformations. The classification of codes with given parameters and/or properties is one of the main problems in Coding theory. Two equivalent codes are considered to be the same code, and one can replace the other if it is more convenient in the corresponding practical application. For example, we say that there is only one Reed–Muller $[32, 16, 8]$ binary code, but in fact there are many different codes with these parameters and they are all equivalent to each other. Some algorithms for construction and/or classification of linear codes use the set of all inequivalent codes with smaller parameters and/or given properties. For more theoretical and practical issues of the classification problem we refer to the book of Kaski and Östergård, *Classification Algorithms for Codes and Designs* [2]. The equivalence test is the main part in any classification algorithm.

In this paper, we describe the algorithm implemented in the program LCEQUIVALENCE, which can be downloaded from the website [3]. The program uses as input a file with generator matrices of linear codes over a finite field $\mathbb{F}_q$ with $q < 64$ elements and gives as output a maximum set of inequivalent codes among them. Moreover, it calculates the order of the automorphism group of a codes and presents the orbits of the coordinates with respect to this group.

Although there are many classification results, the known algorithms for equivalence of linear codes that are implemented in a software are the algorithm of J. Leon [4] (implemented in MAGMA [5] and GAP [6]), the algorithm of Thomas Feulner [7] (implemented in SAGEMATH [8]) and Bouyukliev's algorithm [9] (implemented in Q-EXTENSION). An algorithm was proposed by N. Sendrier in [10], but his algorithm has mostly theoretical value. There are many papers considering the complexity of the code equivalence problem (we distinguish the papers [11,12]). Each of the respective programs has its advantages and

check for updates

Citation: Bouyuklieva, S.; Bouyukliev, I. How to Find the Equivalence Classes in a Set of Linear Codes in Practice? *Mathematics* **2024**, *12*, 328. https://doi.org/10.3390/math12020328

Academic Editor: Raúl M. Falcón

Received: 16 December 2023
Revised: 14 January 2024
Accepted: 17 January 2024
Published: 19 January 2024

limitations, and in our opinion it is good to have more choice so that users can test and use the software that is most suitable for their particular research.

Leon's algorithm checks two codes for equivalence and computes automorphism groups. It is implemented in the computer algebra system MAGMA, which is not freely available, and to compare more codes, a program must be written for it. The same algorithm is implemented in the free computer algebra system GAP, but only for the binary case, i.e., for linear codes over $\mathbb{F}_2$. Feulner's program can only be used via the computer algebra system SAGEMATH (as far as we know), so if one wants to use this algorithm one must be familiar with that system. The algorithm implemented in Q-EXTENSION works with codes over fields with up to 5 elements.

As main advantages of the program LCEQUIVALENCE, we can point out the following: (1) it can be used to find the inequivalent among a huge number of linear codes; (2) it works for codes over prime and composite fields with $q < 64$ elements; (3) the limits on the length and dimension of the considered codes depend only on the used hardware and the computational time (not on the algorithm). As far as we know, there is no other program for the equivalence test of a large number of linear codes. The algorithms implemented in popular packages for computations with linear codes, such as MAGMA and GAP package GUAVA, check pairs of codes for equivalence, and if the researcher needs to test more codes, he/she has to write a program to use this check.

The main idea of the algorithm is to associate each code (regardless of what field it is over) with a binary matrix, so that two codes are equivalent if and only if the corresponding binary matrices are isomorphic. A similar idea was used in [9], but the problem there is that not every automorphism of the binary matrix used is an automorphism of the code, therefore additional verification is needed.

Algebraically, an equivalence relation can be considered in terms of the action of a finite group G on a set of objects Ω. Then, the equivalence classes coincide with the orbits under this action, so two objects are equivalent if and only if they belong to the same orbit of G on Ω [2]. The efficiency of some algorithms depends mostly on the order of the group G, and if it is too large, the equivalence test becomes a very hard problem.

We use an algorithm that involves obtaining canonical forms for the objects using a canonical representative map. To check whether $X \cong Y$ we only need to compare their canonical forms. This approach is used in [9,13]. In most cases, algorithms of this type are more efficient and faster.

The paper is organized as follows. In Section 2, we present some important definitions that we need in our research. In Section 3, we discuss the representation of the linear codes as binary matrices. This representation is very important for the algorithm which is described in the following Section 4. Some computational results are shown in Section 5. In the end, before the references, we give a small conclusion.

2. Preliminaries

In this section, we present the most important definitions that we need in our work. By $\mathbb{F}_q^n$, we denote the n-dimensional vector space over the finite field $\mathbb{F}_q$ with q elements. As a metric, we use the *Hamming distance* between two vectors of $\mathbb{F}_q^n$, which is defined as the number of coordinates in which they differ. Any k-dimensional linear subspace of $\mathbb{F}_q^n$ is called a linear code of length n and dimension k. The vectors in a code are called codewords, and the minimum among the distances between two different codewords is called the minimum distance of the code. If a linear code over $\mathbb{F}_q$ has length n, dimension k and minimum distance d, we say that it is an $[n, k, d]_q$ code. Any matrix of rank k whose rows are codewords from C is called its *generator matrix*.

Definition 1. *We say that two linear q-ary codes C_1 and C_2 of the same length and dimension are equivalentif the codewords of C_2 can be obtained from the codewords of C_1 via a finite sequence of transformations of the following types:*

(1) Permutation of coordinate positions;

(2) *Multiplication of the elements in a given position by a non-zero element of the field;*
(3) *Application of a field automorphism to the elements in all coordinate positions.*

This definition is well motivated as the transformations (1)–(3) preserve the Hamming distance and the linearity (for more details see [2], Chapter 7.3). It is based on the action of the semilinear isometries group $\mathcal{M}_n^*(q) = \mathrm{Mon}_n(\mathbb{F}_q^*) \rtimes \mathrm{Aut}(\mathbb{F}_q) \leq \Gamma_n(\mathbb{F}_q)$ on the vector space $\mathbb{F}_q^n$, where $\Gamma_n(\mathbb{F}_q)$ is the set of all semilinear mappings, i.e., the general semilinear group $\mathrm{Mon}_n(\mathbb{F}_q^*)$ is the group of all monomial $n \times n$ matrices over $\mathbb{F}_q$, and $\mathrm{Aut}(\mathbb{F}_q)$ is the automorphisms group of the field $\mathbb{F}_q$. Note that the group $\mathrm{Mon}_n(\mathbb{F}_q^*)$ is isomorphic to the wreath product $\mathbb{F}_q^* \wr \mathcal{S}_n$, and any monomial matrix is a product of a permutation matrix P and a diagonal matrix $D = \mathrm{diag}(d_1, \ldots, d_n)$, where $d_i \in \mathbb{F}_q^*$, $i = 1, \ldots, n$. Linear q-ary codes C_1 and C_2 of the same length n are equivalent whenever $C_2 = C_1 T$ for some $T \in \mathcal{M}_n^*(q)$. If $CT = C$ for an element $T \in \mathcal{M}_n^*(q)$, then T is called an automorphism of the code C. The set of all automorphisms of C form a group denoted by $\mathrm{Aut}(C)$.

Any element $T \in \mathcal{M}_n^*(q)$ can be written as $T = PD\tau$ where P is a permutation matrix (permutation part), D is a diagonal matrix (diagonal part) and $\tau \in \mathrm{Aut}(\mathbb{F}_q)$. Note that in the case of prime q, $\mathcal{M}_n^*(q) = \mathrm{Mon}_n(\mathbb{F}_q^*)$, and if $q = 2$, then $\mathcal{M}_n^*(q) \cong \mathcal{S}_n$, where S_n is the symmetric group of degree n.

To construct all inequivalent codes with given parameters means to have one representative of each equivalence class. To easily make a distinction between the equivalence classes, we use the concept for a canonical representative, selected on the base of some specific conditions.

Let G be a group acting on a set Ω. This action defines an equivalence relation such that the equivalence classes are the G-orbits in Ω.

Definition 2. *A canonical representative map for the action of the group G on the set Ω is a function $\rho : \Omega \to \Omega$ that satisfies the following two properties:*

1. *For all $X \in \Omega$ it holds that $\rho(X) \cong X$;*
2. *For all $X, Y \in \Omega$ it holds that $X \cong Y$ implies $\rho(X) = \rho(Y)$.*

For $X \in \Omega$, $\rho(X)$ is called the canonical form of X, and $\rho(X)$ is the canonical representative of its equivalence class with respect to ρ.

In our case, the set Ω consists of linear codes, and we can take for a canonical representative of one equivalence class a code which is more convenient for our purposes.

In addition, we use integer-valued invariants. An invariant over the set $\mathbb{E}$ (we use the set of integers $\mathbb{Z}$) is a mapping $f : \Omega \to \mathbb{E}$ such that $f(a) = f(b)$ if a and b are in the same orbit with respect to the action of the group G. We use two different actions:

(1) We take Ω to be the set of all linear $[n, k]_q$ codes (with additional restrictions if needed, for example, only self-orthogonal codes with these parameters or codes with a given minimum and dual distance, etc.) and $G = \mathcal{M}_n^*(q)$.
(2) For the second action, we take Ω to be the set of all codewords of a linear code C, and the group $G = \mathrm{Aut}(C)$. The main invariant that we use in this case is the weight of the codewords.

Finding the canonical form is generally not difficult, but since it is time-consuming, it is important which canonical representation we choose and how we design the algorithm to calculate it. In this work, we do not present a new algorithm for computing canonical forms, but use the algorithm algorithm described in [9]. If the coordinates are previously partitioned according to suitable invariants, the algorithm works much faster. An invariant of the coordinates of C is a function $f\colon N = \{1, 2, \ldots, n\} \to \mathbb{Z}$, such that if i and j are in the same orbit with respect to $\mathrm{Aut}(C)$ then $f(i) = f(j)$, $i, j \in N$. The code C and the invariant f define a partition $\pi = \{N_1, N_2, \ldots, N_l\}$ of the coordinate set N, such that $N_i \cap N_j = \emptyset$ for $i \neq j$, $N = N_1 \cup N_2 \cup \cdots \cup N_l$, and two coordinates i, j are in the same subset of $N \iff f(i) = f(j)$. The description of some very effective invariants and the process of their application are detailed in [9,13].

To define pseudoorbits and coloring, we consider a set of invariants for the codewords of C. The automorphism group $\mathrm{Aut}(C)$ acts on the code and partitions the codewords into orbits. All codewords with the same value of an invariant f define a set which consists of one or more orbits called a pseudoorbit. The values of f give an ordering of the pseudoorbits and a coloring of the codewords (the codeword $u \in C$ has color $f(u)$). The most natural invariant for a codeword is its weight.

As already mentioned, we associate with each q-ary linear code a binary matrix such that two codes are equivalent if and only if the corresponding binary matrices are isomorphic. Therefore, we need a definition for isomorphism of binary matrices.

Definition 3. *Two binary matrices of the same size are isomorphic if the rows of the second one can be obtained from the rows of the first one by a permutation of the columns.*

We also use a similar definition for the equivalence of integer matrices.

Definition 4. *Two integer matrices A and B of the same size are isomorphic ($A \cong B$) if the rows of the second one can be obtained from the rows of the first one by a permutation of the columns.*

Any permutation of the columns of an integer (or binary) matrix A which maps the rows of A into the rows of the same matrix is called an automorphism of A. The set of all automorphisms of A is a subgroup of the symmetric group $\mathcal{S}_n$, denoted by $\mathrm{Aut}(A)$.

3. Representing the Objects

To check codes for equivalence, we need (1) to find a proper set $M(C)$ of codewords, (2) to set a binary matrix $GM(C)$ corresponding to $M(C)$ (if the code is not binary) and (3) to compute the canonical form of $GM(C)$. The set $M(C)$ of codewords of the code C must have the following properties:

- $M(C)$ generates the code C;
- $M(C)$ is stable with respect to $Aut(C)$;
- If $C' \cong C''$ and $\sigma(C') = C''$ then $\sigma(M(C')) \equiv M(C'')$, $\sigma \in \mathcal{M}_n^*$.

We begin with an algorithm for finding a proper set $M(C)$:

1. Initialization: $M(C) = \emptyset$;
2. generate the set D of all codewords with smallest not considered weight;
3. Find and order pseoudoorbits $\{O_1, O_2, \ldots, O_l\}$ of D according to the value of the applied invariant;
4. For r from 1 to l do the following:
   If $\mathrm{rank}(M(C) \bigcup O_r) > \mathrm{rank}(M(C))$, then $M(C) = M(C) \bigcup O_r$;
5. If $\mathrm{rank}(M(C)) < \mathrm{rank}(C)$, then go to point 2 else return.

Since the set $M(C)$ is stable under the action of the automorphism group of the code, together with each vector it contains all its proportional vectors. Let Mt be a $(q-1)s \times n$ matrix whose rows are the codewords from the set $M(C)$. For any column v of this matrix we add all its proportional column vectors and thus construct the matrix Mt_{ext} with the same number of $(q-1)s$ rows but $(q-1)n$ columns. If the codes C' and C'' are equivalent then there is a monomial matrix $M \in \mathrm{Mon}_n(\mathbb{F}_q^*)$ and an automorphism of the field ϕ such that $vM\phi \in C''$ for any codeword $v \in C'$. It turns out that $Mt' \cdot M\phi = P \cdot Mt''$, where P is a permutation matrix that permutes the rows of Mt''. Since in the definition for equivalence of integer matrices we use only permutations (resp. permutation matrices), we consider the matrices Mt'_{ext} and Mt''_{ext}. If we map the monomial matrix M to the permutation $(q-1)n \times (q-1)n$ matrix P_M by replacing any element with a $(q-1) \times (q-1)$ matrix in the following way:

$$0 \mapsto \mathrm{circ}(00\ldots0), \quad \alpha^i \mapsto \mathrm{circ}(0\ldots0\underbrace{1}_{i}0\ldots0) \text{ for } i = 0, 1, \ldots, q-2, \tag{1}$$

we have $Mt'_{ext} \cdot P_M\phi = P \cdot Mt''_{ext}$. This gives us that

$$P_M^T \cdot Mt'^T_{ext}\phi = Mt''^T_{ext} \cdot P^T \Rightarrow Mt'^T_{ext} \cdot (P^T)^{-1}\phi = (P_M^T)^{-1} \cdot Mt''^T_{ext}.$$

This means that instead of the matrices Mt'_{ext} and Mt''_{ext}, we can use their transpose matrices, which is important if $s < n$.

Example 1. *The ternary codes* $C' = \langle \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 2 \end{pmatrix} \rangle$ *and* $C'' = \langle \begin{pmatrix} 1 & 0 & 2 \\ 0 & 1 & 2 \end{pmatrix} \rangle$ *are equivalent and the monomial matrix* $M = \begin{pmatrix} 0 & 0 & 2 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$ *maps the codewords of* C' *to codewords of* C''*. We take*

$$Mt' = \begin{pmatrix} 1 & 0 & 1 \\ 2 & 0 & 2 \\ 0 & 1 & 2 \\ 0 & 2 & 1 \\ 1 & 1 & 0 \\ 2 & 2 & 0 \end{pmatrix}, \quad Mt'' = \begin{pmatrix} 1 & 0 & 2 \\ 2 & 0 & 1 \\ 0 & 1 & 2 \\ 0 & 2 & 1 \\ 1 & 2 & 0 \\ 2 & 1 & 0 \end{pmatrix}, \quad Mt' \begin{pmatrix} 0 & 0 & 2 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} Mt'',$$

$$Mt'_{ext} = \begin{pmatrix} 1 & 2 & 0 & 0 & 1 & 2 \\ 2 & 1 & 0 & 0 & 2 & 1 \\ 0 & 0 & 1 & 2 & 2 & 1 \\ 0 & 0 & 2 & 1 & 1 & 2 \\ 1 & 2 & 1 & 2 & 0 & 0 \\ 2 & 1 & 2 & 1 & 0 & 0 \end{pmatrix}, \quad Mt''_{ext} = \begin{pmatrix} 1 & 2 & 0 & 0 & 2 & 1 \\ 2 & 1 & 0 & 0 & 1 & 2 \\ 0 & 0 & 1 & 2 & 2 & 1 \\ 0 & 0 & 2 & 1 & 1 & 2 \\ 1 & 2 & 2 & 1 & 0 & 0 \\ 2 & 1 & 1 & 2 & 0 & 0 \end{pmatrix}.$$

We take only one representative of each class of proportional vectors and thus obtain the set $M'(C)$. To define the binary matrix $GM(C)$ that corresponds to the set $M'(C)$, we consider the cases $q = 2$ and $q \geq 3$ separately.

Consider first the binary case. Let $GM(C)$ be a binary matrix whose rows are the codewords from $M(C)$.

Theorem 1. *The binary* $[n, k]$ *codes* C_1 *and* C_2 *are equivalent if and only if the binary matrices* $GM(C_1)$ *and* $GM(C_2)$ *are isomorphic.*

Proof. Let the binary codes C_1 and C_2 be equivalent. Hence, there is a permutation $\sigma \in \mathcal{S}_n$ such that $\sigma(C_1) = C_2$. It follows that $\sigma(M(C_1)) = M(C_2)$, therefore the matrices $GM(C_1)$ and $GM(C_2)$ are isomorphic.

Let $GM(C_1) \cong GM(C_2)$. Then, there is a permutation $\tau \in \mathcal{S}_n$ such that the matrices $\tau(GM(C_1))$ and $GM(C_2)$ have the same rows but are ordered differently. Hence, τ maps the codewords from $M(C_2)$ to codewords in $M(C_2)$, therefore the codes C_1 and C_2 are equivalent. □

Representing a linear code over a field with more than two elements as a binary matrix is already introduced in [9,14], but here we present a different approach which is more general and more suitable for codes over fields with $q \geq 5$ elements.

Let C be a q-ary linear code for with $q = p^m \geq 3$, where p is a prime, and let α be a primitive element of $\mathbb{F}_q$. We map each element of $\mathbb{F}_q$ to a circular binary matrix of order $q - 1$ in the same way as shown in (1).

Consider the matrix $Mt'(C)$ whose rows are the codewords from $M'(C)$ for the q-ary linear code C of length n. Replacing all elements in $Mt(C)$ using (1), we obtain the matrix $GM(C)$ of size $(q-1)s \times (q-1)n$ where $s = |M'(C)|$. Note that this presentation includes also the proportional vectors of the codewords in $M'(C)$ and $(q-1)s = |M(C)|$.

To construct the needed binary matrix, we use some additional matrices that depend on the field. Denote the $(q-1)\times(q-1)$ circulant $\mathrm{circ}(010\ldots0)$ by L and let

$$A^* = I_{q-1} + 2(L + L^p + \cdots + L^{p^{m-1}}).$$

The automorphism group of this matrix is $\mathrm{Aut}(A^*) = \langle\sigma,\tau\rangle < \mathcal{S}_{q-1}$, where $\sigma = (0,1,\ldots,q-2)$ is the presented cycle of length $q-1$, and $\tau = \Omega_1\ldots\Omega_t$, where Ω_i corresponds to the i-th ciclotomic class modulo $q-1$. This group is isomorphic to $\mathbb{F}_q^* \rtimes \mathrm{Aut}(\mathbb{F}_q)$ which is the automorphism group of the trivial $[1,1]_q$ code. We prove this isomorphism in detail in the following lemma.

Lemma 1. *$Aut(A^*) \cong \mathbb{F}_q^* \rtimes Aut(\mathbb{F}_q)$.*

Proof. Let us label the rows and the columns of the matrix A^* from 0 to $q-2$. We first consider the case of a prime $q \geq 3$. Then the i-th row of A^* is

$$(0,\ldots,0,\underbrace{1}_{i},\underbrace{2}_{i+1},0,\ldots,0),\ \text{for } 0 \leq i \leq q-3,$$

and $(2,0,\ldots,0,1)$ if $i = q-2$. The j-th column is $(1,0,\ldots,0,2)^T$ if $j = 0$, and

$$(0,\ldots,0,\underbrace{2}_{j-1},\underbrace{1}_{j},0,\ldots,0),\ \text{if } 1 \leq j \leq q-2.$$

Now, take a permutation $\tau \in \mathrm{Aut}(A^*) \leq \mathcal{S}_{q-1}$. If $\tau(0) = j$, then the column with number 0 goes to the j-th column. This column contains only one coordinate equal to 1 and this 1 is on the j-th row which contains only one coordinate equals to 2, namely the $j+1$-th coordinate. Hence, $\tau(1) = j+1$. Considering the $j+1$-th column and the positions of the coordinates equal to 1 and 2, we have $\tau(i) = j+i \pmod{q-1}$. Hence, $\tau = (0,1,\ldots,q-2)^j$ and $\mathrm{Aut}(A^*) = \langle(0,1,\ldots,q-2)\rangle \cong \mathbb{F}_q^*$.

Let us now consider the general case when $q = p^m$ for a prime p and integer $m \geq 1$. In this case, if $A^* = (a_{ij})$, $0 \leq i,j \leq q-2$, then $a_{ii} = 1$ for all $i = 0,1,\ldots,q-2$, and $a_{ij} = 2$ if $j = i + p^s \pmod{q-1}$, $s = 0,1,\ldots,m-1$. We can consider any automorphism $\tau \in Aut(A^*) \leq \mathcal{S}_{q-1}$ as a permutation of the nonzero elements of the field. If $\tau(\alpha^0) = \alpha^i$, the row with number 0 goes to the i-th row. This means that

$$\{\alpha,\alpha^p,\ldots,\alpha^{p^{m-1}}\} \xrightarrow{\tau} \{\alpha^{i+1},\alpha^{i+p},\ldots,\alpha^{i+p^{m-1}}\}$$

(these are the positions of the coordinates equal to 2 in these two rows). But then, $\tau(\alpha) = \alpha^{i+p^{s_1}}$ and the number 1 row goes to row with number $i+p^{s_1}$ and therefore

$$\{\alpha^2,\alpha^{1+p},\ldots,\alpha^{1+p^{m-1}}\} \xrightarrow{\tau} \{\alpha^{1+i+p^{s_1}},\alpha^{i+p^{s_1}+p},\ldots,\alpha^{i+p^{s_1}+p^{m-1}}\}.$$

Hence, $\tau(\alpha^2) = \alpha^{i+p^{s_1}+p^{s_2}}$. Then,

$$\{\alpha^3,\alpha^{2+p},\ldots,\alpha^{2+p^{m-1}}\} \xrightarrow{\tau} \{\alpha^{1+i+p^{s_1}+p^{s_2}},\alpha^{i+p^{s_1}+p^{s_2}+p},\ldots,\alpha^{i+p^{s_1}+p^{s_2}+p^{m-1}}\}.$$

Thus, $\tau(\alpha^3) = \alpha^{i+p^{s_1}+p^{s_2}+p^{s_3}}$. In this way we obtain that $\tau(\alpha^p) = \alpha^{i+p^{s_1}+p^{s_2}+\cdots+p^{s_p}}$. But at the same time, $\tau(\alpha^p) = \alpha^{i+p^r}$ and so $\alpha^{i+p^{s_1}+p^{s_2}+\cdots+p^{s_p}} = \alpha^{i+p^r}$. It follows that

$$\alpha^{p^{s_1}+p^{s_2}+\cdots+p^{s_p}-p^r} = 1 \ \Rightarrow\ p^m - 1 \mid p^{s_1}+p^{s_2}+\cdots+p^{s_p}-p^r.$$

But $s_j \leq m-1$, $j = 1,\ldots,p$, and $r \geq 0$, thus $p^{s_1}+p^{s_2}+\cdots+p^{s_p}-p^r \leq p^m-1$. Hence, $p^{s_1}+p^{s_2}+\cdots+p^{s_p}-p^r = 0$ or p^m-1. It turns out that $s_1 = s_2 = \cdots = s_p = s$, $r = s+1$, and $\tau(\alpha^j) = \alpha^{i+jp^s}$ for $j = 0,1,\ldots,q-2$ which means that $\tau = \sigma^i\phi^s \in G$, where

$\sigma = (0, 1, 2, \ldots, q-2)$ and ϕ is the Frobenius automorphism defined by $\phi(a) = a^p$, $a \in \mathbb{F}_q$. It follows that $\mathrm{Aut}(A^*) \cong \langle \sigma, \phi \rangle \cong \mathbb{F}_q^* \rtimes \mathrm{Aut}(\mathbb{F}_q)$. □

We give two examples with the matrices A^*.

Example 2. *Consider the cases $q = 5$ and $q = 8$:*

(1) *If $q = 5$ we have* $A^* = \mathrm{circ}(1200) = \begin{pmatrix} 1 & 2 & 0 & 0 \\ 0 & 1 & 2 & 0 \\ 0 & 0 & 1 & 2 \\ 2 & 0 & 0 & 1 \end{pmatrix}$, $Aut(A^*) = \langle (0123) \rangle < \mathcal{S}_4$;

(2) *If $q = 8$ then* $A^* = \mathrm{circ}(1220200) = \begin{pmatrix} 1 & 2 & 2 & 0 & 2 & 0 & 0 \\ 0 & 1 & 2 & 2 & 0 & 2 & 0 \\ 0 & 0 & 1 & 2 & 2 & 0 & 2 \\ 2 & 0 & 0 & 1 & 2 & 2 & 0 \\ 0 & 2 & 0 & 0 & 1 & 2 & 2 \\ 2 & 0 & 2 & 0 & 0 & 1 & 2 \\ 2 & 2 & 0 & 2 & 0 & 0 & 1 \end{pmatrix}$,

$Aut(A^*) = \langle (0123456), (124)(365) \rangle < \mathcal{S}_7$, $|Aut(A^*)| = 21$.

Next, we replace the elements 0, 1 and 2 in the matrix A^* with binary column-vectors with two coordinates as follows:

$$0 \mapsto \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \ 1 \mapsto \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \ 2 \mapsto \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

and in this way from the matrix A^* we construct the matrix B'. To have a matrix with an automorphism group which is isomorphic to $\mathrm{Aut}(A^*)$, we expand B' with the columns of the matrix $I_{q-1} \otimes \binom{1}{1}$, and denote the obtained matrix by B^*.

Example 3. *Consider again the case $q = 5$. Then,*

$$B^* = \begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Recall that $\mathrm{Aut}(A^*) = \langle \sigma, \tau \rangle$. Let us map $\sigma \in \mathcal{S}_{q-1}$ to the permutation $\sigma' = (0, 1, \ldots, q-2)(q-1, q, \ldots, 2q-3) \in \mathcal{S}_{2q-2}$, and τ to $\tau' = \Omega_1 \ldots \Omega_t \Omega'_1 \ldots \Omega'_t \in \mathcal{S}_{2q-2}$ where τ' is defined as follows: if $\Omega_i = (j_1, \ldots, j_{m_i})$ then $\Omega'_i = (q-1+j_1, \ldots, q-1+j_{m_i})$, $i = 1, \ldots, t$. This map defines an isomorphism between $\mathrm{Aut}(A^*)$ and $\mathrm{Aut}(B^*)$, and so $\mathrm{Aut}(B^*) \cong \mathbb{F}_q^* \rtimes \mathrm{Aut}(\mathbb{F}_q)$.

We need two more square matrices, namely the $(q-1)n \times (q-1)n$ matrix $A = I_n \otimes A^*$ and the $2(q-1)n \times 2(q-1)n$ matrix $B = (I_n \otimes B' | I_{(q-1)n} \otimes \binom{1}{1})$. The automorphism groups of these two matrices are isomorphic to $\mathcal{M}_n^*(q)$.

Now expand the matrix $GM(C)$ by adding the rows of the matrix A. In this way we obtain the matrix $GM_A(C) = \binom{GM(C)}{A}$ with $(q-1)(s+n)$ rows and $(q-1)n$ columns.

Example 4. *Consider the* $[2,1]_5$ *code* C *with a generator matrix* $(1\ 2)$. *For this code*

$$GM_A(C) = \begin{pmatrix} 1&0&0&0&0&1&0&0\\ 0&1&0&0&0&0&1&0\\ 0&0&1&0&0&0&0&1\\ 0&0&0&1&1&0&0&0\\ 1&2&0&0&0&0&0&0\\ 0&1&2&0&0&0&0&0\\ 0&0&1&2&0&0&0&0\\ 2&0&0&1&0&0&0&0\\ 0&0&0&0&1&2&0&0\\ 0&0&0&0&0&1&2&0\\ 0&0&0&0&0&0&1&2\\ 0&0&0&0&2&0&0&1 \end{pmatrix}$$

The automorphism group of the matrix A is isomorphic to $\mathcal{M}_n^*(q)$ and therefore $\mathrm{Aut}(GM_A(C)) < \mathrm{Aut}(A)$ and consists of those automorphisms of A which belong to $\mathrm{Aut}(GM(C))$.

Example 5. *Consider the code from Example 4. If we swap the two coordinates and then multiply the second coordinate by 4, we obtain the same code. Hence, the monomial matrix* $\begin{pmatrix} 0&4\\ 1&0 \end{pmatrix}$ *belongs to the automorphism group of this code. We map this monomial matrix to the permutation matrix*

$$Q_A = \left(\begin{array}{cccc|cccc} 0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&1\\ 0&0&0&0&1&0&0&0\\ 0&0&0&0&0&1&0&0\\ \hline 1&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&0\\ 0&0&0&1&0&0&0&0 \end{array}\right).$$

It is easy to see that the matrix $GM_A(C)Q_A$ *has the same rows as the matrix* $GM_A(C)$ *and so* $Q_A \in Aut(GM_A(C))$.

The matrices $GM(C)$ and $GM_A(C)$ are related to the matrix $GM_B(C)$ which we construct in the following way: (1) we glue the $(q-1)s \times (q-1)n$ zero matrix to the matrix $GM(C)$; (2) we add the rows of B. The obtained matrix has $(q-1)(s+2n)$ rows and $2(q-1)n$ columns.

Example 6. *Consider the same* $[2,1]_5$ *code* C *with a generator matrix* $(1\ 2)$. *For this code*

$$GM_B(C) = \left(\begin{array}{cccccccc|cccccccc} 1&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&1&0&0&0&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&1&0&0&0&0&0&0&0&0\\ 0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0\\ \hline 1&1&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0&1&0&0&0&0&0&0&0\\ 0&1&1&0&0&0&0&0&0&1&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&0&0&1&0&0&0&0&0&0\\ 0&0&1&1&0&0&0&0&0&0&1&0&0&0&0&0\\ 0&0&0&1&0&0&0&0&0&0&1&0&0&0&0&0\\ 1&0&0&1&0&0&0&0&0&0&0&1&0&0&0&0\\ 1&0&0&0&0&0&0&0&0&0&0&1&0&0&0&0\\ 0&0&0&0&1&1&0&0&0&0&0&0&1&0&0&0\\ 0&0&0&0&0&1&0&0&0&0&0&0&1&0&0&0\\ 0&0&0&0&0&1&1&0&0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&1&0&0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&1&1&0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&1&0&0&0&0&0&0&1&0\\ 0&0&0&0&1&0&0&1&0&0&0&0&0&0&0&1\\ 0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&1 \end{array}\right)$$

Theorem 2. *The q-ary $[n,k]$ codes C_1 and C_2 are equivalent if and only if the binary matrices $GM_B(C_1)$ and $GM_B(C_2)$ are isomorphic.*

Proof. In converting the matrix $Mt(C)$ to the matrix $GM_B(C)$, we replace each column with a group of $(q-1)$ columns and then extend the obtained matrix with extra zero columns and the rows of the matrix B. Therefore, there is a one-to-one correspondence between the permutations of the coordinates of C and the permutations of the corresponding groups of columns in $GM_B(C)$. More precisely, if $\tau \in \mathcal{S}_n$ is a permutation of the n coordinates of C, it goes to the permutation $\tau_B \in \mathcal{S}_{2(q-1)n}$ of the columns of $GM_B(C)$ such that if $\tau(j) = i_j$ then $\tau_B((q-1)j - j_1) = (q-1)i_j - j_1$ and $\tau_B((q-1)(n+j) - j_1) = (q-1)(n+i_j) - j_1$ for $1 \le j \le n$, $0 \le j_1 \le q-2$. Moreover, the multiplication of a coordinate by α^i can be considered as applying the permutation $\sigma = (0, 1, \ldots, q-2)$ on the corresponding group of $2(q-1)$ columns i times. The automorphism of the field applied to all coordinates of all codewords also can be considered as a permutation of the columns of $GM_B(C)$.

The above observation shows that if $\varphi \in \mathcal{M}_n(q)^*$ maps the code C_1 to C_2, then the corresponding to φ permutation of the columns of $GM_B(C_1)$ maps the rows of this matrix to the rows of $GM_B(C_2)$. Note that ϕ can be considered as a permutation followed by multiplications of the coordinates of the code by nonzero elements of the field and eventually applying an automorphism of the field to all coordinates of all codewords.

Conversely, if we take a permutation π of the columns of $GM_B(C_1)$ that maps the rows of this matrix to the rows of $GM_B(C_2)$, then $\pi \in \mathrm{Aut}(B) \cong \mathcal{M}_n^*(q)$ and π maps the rows of $GM'(C_1)$ to the rows of $GM'(C_2)$. □

Corollary 1. *$Aut(C) \cong Aut(GM_B(C))$.*

Example 7. *The automorphism group of the code $C = \langle (1,2) \rangle \subset \mathbb{F}_5^2$ is*

$$Aut(C) = \{\beta I_2, \beta \begin{pmatrix} 0 & 4 \\ 1 & 0 \end{pmatrix}, \beta \in \mathbb{F}_5^*\}, \ \ |Aut(C)| = 8.$$

According to Corollary 1, $Aut(C) \cong Aut(GM_B(C))$. Indeed,

$$Aut(GM_B(C)) = \langle (1,2,3,4)(5,6,7,8)(9,10,11,12)(13,14,15,16),$$

$$(1,7,3,5)(2,8,4,6)(9,15,11,13)(10,16,12,14) \rangle \cong Aut(C).$$

Calculating $|Aut(GM(C))| = 384$, $|Aut(GM_A(C))| = 8$, $|Aut(GM_{B'}(C))| = 48$, where $GM_{B'}(C)$ is the matrix $GM_B(C)$ without the last $n(q-1)$ columns, we see that the extra rows and columns are important to remove the automorphisms of the matrices that do not correspond to automorphisms of the code.

4. The Algorithm

The classification problem we want to solve is the following: given a set of linear q-ary codes of length n and dimension k, take only one representative of each equivalence class. We present the algorithm that gives a solution of the problem. As an input, we have a family of linear codes presented by a generator matrix. The set of these matrices is denoted by W. As an output, we have one representative of each equivalence class with respect to the defined (in the previous sections) equivalence relation in the set of linear codes. Generator matrices of these representatives compose the set U. The algorithm follows these steps:

1. We find a generating set $M(C)$ for each code C. In most cases this is the set of codewords with minimum weight. If this set has a rank smaller than k, we add the codewords with another weight, etc. In some cases we also use other invariants to get a smaller and more convenient generating set.
2. We take the binary matrix $GM_B(C)$ as it is described in the previous section. According to Theorem 2, two codes C_1 and C_2 are equivalent if and only if $GM_B(C_1) \cong GM_B(C_2)$.

3. We find the canonical form of the matrix $GM_B(C)$ as it is described in [9].
4. We compare the canonical forms of the matrices. If two matrices have the same canonical form, they are isomorphic and the corresponding codes are equivalent. If these matrices have different canonical forms, the codes are not equivalent.

A pseudocode of Algorithm 1 is given below.

Algorithm 1 Procedure *Find_Inequivalent_Codes*

Input: A set W of generator matrices of linear codes.
Output: A maximal set $U \subseteq W$ of generator matrices of inequivalent codes.

```
    var
    HA: array of integers;
    Hf: file;
    not_found: boolean;
 1: Initialize Hf;
 2: HA[i] ← 0 for all i = 0, ..., size;
 3: for each generator matrix G ∈ W do
 4:    Find generating set M(C) of the corresponding linear code C;
 5:    construct the corresponding binary matrix GM_B(C);
 6:    find the canonical form A the matrix GM_B(C) and its hash value h(A);
 7:    call the procedure SEARCH_IN_HASH_TABLE with parameters HA, A, h(A),
       Hf {the procedure gives the value of the boolean variable not_found which is true
       or false}
 8:    if not_found then
 9:       U ← U ∪ {G}
10:    end if
11: end for
```

For fast search between a large number of similar objects (in our case, binary matrices in canonical form), an abstract data structure known as a hash map is used (more information about hash tables and hash maps can be found in [15,16]). It consists of three components: a hash array, a hash function and a hash table or file. The basic idea is that when we get a new code, we first compare its hash with the hash values of already considered codes. If the new hash is different from all already computed, the new code is not equivalent to either of the previous codes. If the hash is equal to the hash value of another code, then we compare the corresponding binary matrices.

Let us introduce the hashing we use in a little more detail. A hash array $HA[]$ can be considered as an array of non-negative integers. To any object, the hash function associates an integer, which can be considered as an index in the hash array $HA[]$. Each element of the array $HA[i]$ (if $HA[i] \neq 0$) is a pointer (a positive integer) to the location of the object in a hash file or table. In the beginning, all values in the array HA are 0s. We define a hash file as an addressable memory of a large size that consists of cells. Objects or records (that contain an object) can be written in these cells. A binary file which has direct access, or a dynamic structure (if enough RAM is available), can serve as the hash file (hash table).

We use a non-cryptographic hash function h that maps an object of arbitrary size to a positive integer not larger than the size of the hash array. If the number of different matrices we examine is greater than the size of the hash array, there will be at least two matrices with the same hash value, which is called a collision. The hash function h must meet the following requirements:

- The function should be easily and quickly calculated.
- It should provide a uniform distribution in the hash array. In this case, the number of collisions will be smaller. Even with a very good hash function, collisions are not excluded. The collision handling technique which we use is known as Separate Chaining (Open Hashing). It is usually implemented using linked lists. A nonzero cell of the hash array points to the first object in a linked list of objects that have the

same hash value (if no matches with the same hash value are found, the list consists of one element).

An implementation of this approach is presented in Algorithm 2.

The total complexity of the main algorithm depends on the size of the set of codes (the set W consists of their generator matrices) but is difficult to determine because it depends on many subalgorithms. For codes with large dimension, one of the most expensive parts is the generating a proper set $M(C)$ of codewords. In this case, a more efficient approach than the standard brute force algorithm (with Gray code) gives the Brouwer–Zimmermann algorithm and its modifications [17]. The construction of the binary matrix $GM_B(C)$, which corresponds to the set $M(C)$, despite its theoretical complexity, practically does not affect the execution time of the algorithm. Finding the canonical form of a binary matrix represents the solution to the graph isomorphism problem, since in this case a bipartite graph is considered. The computation time required depends on whether the binary matrix corresponds to a regular structure such as a combinatorial design and does not particularly depend on the order of the automorphism group. The use of a canonical form reduces the isomorphism problem between binary matrices to a matrix comparison problem which has complexity fixed by the matrix parameters and negligible execution time. In the case of a large number m of non-equivalent matrices, if a hash map is not used, the complexity of comparing a matrix with the others becomes linear with respect to m, and when a hash map is used, the complexity is reduced to a constant.

Algorithm 2 Procedure SEARCH_IN_HASH_TABLE

Input: $hashkey = h(A)$, the matrix A, the integer array HA, the file Hf
Output: not_found: boolean;

```
    type
      structure MATREC {
        matrix M;
        integer Next}        // next matrix with the same hashkey
    var
       AMATREC, BMATREC of type MATREC;
      curent_position, old_position: integer;
 1: AMATREC.M ← A;
 2: AMATREC.next ← 0;
 3: if HA[hashkey] = 0 then
 4:   append AMATREC to the file Hf;
 5:   HA[hashkey] ← the position of AMATREC in the file Hf;
 6:   return not_found ← true
 7: else
 8:   curent_position ← HA[hashkey];
 9:   while curent_position ≠ 0 do
10:     old_position ← curent_position;
11:     read from curent_position the structure BMATREC {from the file Hf};
12:     curent_position ← BMATREC.next
13:     if AMATREC.M = BMATREC.M then
14:       return not_found ← false
15:     end if
16:   end while
17:   append AMATREC to the file Hf;
18:   curent_position ← the first position of AMATREC in the file Hf;
19:   BMATREC.Next ← curent_position;
20:   write from old_position the structure BMATREC to the file Hf
21:   return not_found ← true;
22: end if
```

5. Computational Results

We made some experiments with codes with different lengths and dimensions over different finite fields.

In Table 1, we present the execution times for obtaining inequivalent codes. In the table, we have presented results in the following cases, which we consider to be significant:

- Finding the inequivalent codes in a set of randomly constructed linear codes. This is useful for the process of generating linear codes of a given type and/or given parameters. During the intermediate steps in the generation process, a large amount of codes is usually constructed, from which only the inequivalent ones should be taken and the procedure continued with them. As an example, we can mention one of the methods for isomorph-free exhaustive generation, namely isomorph rejection based on recorded objects [2].
- One of the most important classes of linear codes for theoretical and practical reasons is the class of self-dual codes. A self-dual code over $\mathbb{F}_q$ is a linear code which coincides with its orthogonal complement in the vector space $\mathbb{F}_q^n$ with respect to a given inner product [18]. The study of these codes involves the generation and classification of self-dual codes over a given finite field with prescribed length, minimum distance and other parameters such as a weight enumerator or an automorphism group (see, for example, [19]). Therefore, we have included in the table results on this type of codes over fields with 3, 5 and 7 elements.
- The quasi-cyclic codes have nontrivial automorphism groups that contain as a subgroup a cyclic group of a given order (see [18] for more details about these codes). This is the reason why we also consider these types of codes and include a special option in the LCEQUIVALENCE program to generate random quasi-cyclic codes.

Table 1. Execution times (in seconds) for obtaining inequivalent codes in a set of randomly constructed linear codes with the program LCEQUIVALENCE.

n	k	q	Number of Codes	Number of Inequivalent Codes	Time in Seconds	Comment
20	10	2	100,000	65,566	39.47	random
40	20	2	10,000	10,000	89.25	random
120	40	2	100	100	336.98	quasi-cyclic
120	40	2	100	100	214.34	random
28	14	3	6931	6931	15,386.21	self-dual
60	16	4	100	100	231.30	random
5000	10	4	1000	1000	153.91	random
16	8	5	535	535	33.187	self-dual
12	6	7	64	64	0.30	self-dual
40	20	7	2	1	12.44	self-dual
56	28	7	2	1	25,161.18	self-dual
15	3	9	1,000,000	4115	621.85	quasi-cyclic
20	10	16	1000	1000	9.87	random
20	10	17	1000	1000	17.33	random
20	10	23	1000	1000	110.38	random
1000	5	31	100	100	110.39	random
10	3	32	10,000	9999	167.81	random
20	10	61	100	100	80.59	random

Table 1 is structured as follows. The first three columns contain the parameters n (length), k (dimension) and q (size of the field) of the constructed $[n,k]_q$ codes. In the fourth column we present the number of constructed codes that will be checked for equivalence. The next column contains the number of the inequivalent codes in the given set of codes. The execution time is shown in the sixth column. In the last column we comment on whether the constructed codes are of special types, for example self-dual or quasi-cyclic.

The computations were executed on a WINDOWS 11 OS in a single core of an INTEL XEON GOLD 5118 CPU with a 2.30 GHz clock frequency.

The program LCEQUIVALENCE has two options for constructing random codes—for given positive integers ℓ, n, k, d and q, (1) constructing ℓ random linear $[n, k, \geq d]_q$ codes and (2) constructing ℓ random quasi-cyclic $[n, k, \geq d]_q$ codes. We have provided this option so that we can test a large number of random codes for equivalence. The self-dual codes that we check for equivalence are taken from the Database of self-dual codes by Masaaki Harada and Akihiro Munemasa [20].

As input we use generator matrices of the considered codes. The program then goes through the four steps presented in Section 4 and takes exactly one representative from each equivalence class. Generator matrices of these representatives, as well as additional information about them (parameters, automorphism groups, orbits, etc.), are written to a file.

6. Conclusions

This paper is devoted to the algorithm for equivalence of linear codes over finite fields implemented in the program LCEQUIVALENCE, which is a module of the software package QEXTNEWEDITION v1.1 [3]. The program takes as input a file with generator matrices of linear codes and gives as output a file with generator matrices for the maximum set of inequivalent among the input codes. Using this program, one can test for equivalence codes over fields with $q < 64$ elements. As can be seen in Table 1, the file may contain large number of codes. As far as we know, there is no other program for the equivalence test of a large number of linear codes. The algorithms implemented in the popular packages that deal with linear codes check pairs of codes for equivalence.

Instead of linear codes, the program LCEQUIVALENCE can test binary matrices for equivalence. This means that it can be used for equivalence (isomophism) testing of objects that can be represented by binary matrices including graphs [14].

The program also gives the orders and generators of the automorphism groups of the corresponding binary matrices. Optionally, by marking the corresponding option, the program can also write additional information to the output file, such as a permutation that leads one matrix to its equivalent, orbits, weight distributions of the codes, etc.

No installation of the program is required. It is only necessary for the user to create a directory and download a version of the program that corresponds to the operating system used—LINUX or WINDOWS. The interface is very simple. The user can choose from the following seven options:

1. Find inequivalent codes.
2. Find inequivalent matrices.
3. What to print in the output file about codes?
4. What to print in the output file about matrices?
5. Change the name of the input file.
6. Random codes (will be written in a file with name EXAM in the directory RES_DIR0).
7. Random quasi-cyclic codes (will be written in a file with name EXAM in the directory RES_DIR0).

As an input the program LCEQUIVALENCE uses a text file with generator matrices of linear codes in the form described in the previous version of Q-EXTENSION. There is no limit to the number of codes in the input file.

Author Contributions: Conceptualization, I.B.; methodology, I.B.; software, I.B.; validation, I.B. and S.B.; formal analysis, S.B.; resources, I.B. and S.B.; data curation, I.B.; writing—original draft preparation, S.B.; writing—review and editing, S.B.; project administration, I.B. and S.B.; funding acquisition, I.B. and S.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research was partially supported by Bulgarian National Science Fund grant number KP-06-H62/2/13.12.2022.

Data Availability Statement: No new data were created or analyzed in this study. Data sharing is not applicable to this article.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Klein, F. Das Erlanger Programm (1872). In *Vergleichende Betrachtungen über Neuere Geometrische Forschungen*, 3rd ed.; Harri Deutsch: Frankfurt am Main, Germany, 1997.
2. Kaski, P.; Östergård, P. *Classification Algorithms for Codes and Designs*; Springer: Berlin/Heidelberg, Germany, 2006.
3. Bouyukliev, I. QextNewEdition–LCequivalence Module. Available online: http://www.moi.math.bas.bg/moiuser/~data/Software/QextNewEditionLCequiv.html (accessed on 15 December 2023).
4. Leon, J. Computing automorphism groups of error-correcting codes. *IEEE Trans. Inform. Theory* **1982**, *28*, 496–511. [CrossRef]
5. Bosma, W.; Cannon, J.; Playoust, C. The Magma algebra system I: The user language. *J. Symb. Comput.* **1997**, *24*, 235–265. [CrossRef]
6. The GAP Group: GAP–Groups, Algorithms, and Programming, Version 4.12.2. 2022. Available online: https://www.gap-system.org (accessed on 15 December 2023).
7. Feulner, T. The automorphism groups of linear codes and canonical representatives of their semilinear isometry classes. *Adv. Math. Commun.* **2009**, *3*, 363–383. [CrossRef]
8. SageMath. Open-Source Mathematical Software System. Available online: https://www.sagemath.org/ (accessed on 15 December 2023).
9. Bouyukliev, I. About the code equivalence. In *Advances in Coding Theory and Cryptology*; Shaska, T., Huffman, W., Joyner, D., Ustimenko, V., Eds.; World Scientific Publishing: Singapore, 2007; pp. 126–151.
10. Sendrier, N. Finding the permutation between equivalent linear codes: The support splitting algorithm. *IEEE Trans. Inform. Theory* **2000**, *46*, 1193–1203. [CrossRef]
11. Petrank, E.; Roth, R. Is code equivalence easy to decide? *IEEE Trans. Inform. Theory* **1997**, *43*, 1602–1604. [CrossRef]
12. Sendrier, N.; Simos, D. How easy is code equivalence over $\mathbb{F}_q$? In Proceedings of the 8th International Workshop on Coding Theory and Cryptography WCC 2013, Bergen, Norway, 15–19 April 2013.
13. McKay, B.; Piperno, A. Practical graph isomorphism, II. *J. Symb. Comput.* **2014**, *60*, 94–112. [CrossRef]
14. Bouyukliev, I.; Dzhumalieva-Stoeva, M. Representing equivalence problems for combinatorial objects. *Serdica J. Comput.* **2014**, *8*, 327–354. [CrossRef]
15. Sedgewick, R.; Wayne, K. *Algorithms*, 4th ed.; Addison-Wesley Professional: Boston, MA, USA, 2011; Volume 1.
16. Cormen, T.H.; Leiserson, C.E.; Rivest, R.L.; Stein, C. *Chapter 11: Hash Tables, in Introduction to Algorithms*, 4th ed.; MIT Press: Cambridge, MA, USA; McGraw-Hill: New York, NY, USA, 2022.
17. Bouyuklieva, S.; Bouyukliev, I. An Extension of the Brouwer–Zimmermann Algorithm for Calculating the Minimum Weight of a Linear Code. *Mathematics* **2021**, *9*, 2354. [CrossRef]
18. Huffman, W.C.; Pless, V. *Fundamentals of Error-Correcting Codes*; Cambridge University Press: Cambridge, UK, 2003.
19. Huffman, W.C. On the classification and enumeration of self-dual codes. *Finite Fields Appl.* **2005**, *11*, 451–490. [CrossRef]
20. Harada, M.; Munemasa, A. Database of Self-Dual Codes. Available online: https://www.math.is.tohoku.ac.jp/~munemasa/selfdualcodes.htm (accessed on 11 January 2024).