Characteristic vector and weight distribution of a linear code

Iliya Bouyukliev1 · Stefka Bouyuklieva2 · Tatsuya Maruta3 · Paskal Piperkov1

Received: 15 December 2019 / Accepted: 24 September 2020 / Published online: 29 October 2020
© Springer Science+Business Media, LLC, part of Springer Nature 2020

Abstract
An algorithm for computing the weight distribution of a linear \([n, k]\) code over a finite field \(\mathbb{F}_q\) is developed. The codes are represented by their characteristic vector with respect to a given generator matrix and a generator matrix of the \(k\)-dimensional simplex code \(S_{q,k}\).

Keywords Linear code · Weight distribution · Walsh transform

Mathematics Subject Classification (2010) 94B05 · 15B36 · 11Y16

1 Introduction

Many problems in coding theory require efficient computing of the weight distribution of a given linear code. Some sufficient conditions for a linear code to be good or proper for error detection are expressed in terms of the weight distribution [13]. The weight distribution of the hull of a code provides a signature and the same signature computed for any permutation-equivalent code will allow the reconstruction of the permutation [31]. The weight distributions of codes can be used to compute some characteristics of the Boolean and vectorial Boolean functions [10].

Stefka Bouyuklieva
stefka@ts.uni-vt.bg

Iliya Bouyukliev
iliyab@math.bas.bg

Tatsuya Maruta
maruta@mi.s.osakafu-u.ac.jp

Paskal Piperkov
ppiperkov@math.bas.bg

1 Institute of Mathematics and Informatics, Bulgarian Academy of Sciences, P.O. Box 323, Veliko Tarnovo, Bulgaria
2 Faculty of Mathematics and Informatics, St. Cyril and St. Methodius University of Veliko Tarnovo, Veliko Tarnovo, Bulgaria
3 Department of Mathematical Sciences, Osaka Prefecture University, Sakai, Osaka, 599-8531, Japan
In 1978, Berlekamp, McEliece, and van Tilborg [3] proved that two fundamental problems in coding theory, namely maximum-likelihood decoding and computation of the weight distribution, are NP-hard for the class of binary linear codes. The formal statement of the corresponding decision problem is [33]

**Problem:** WEIGHT DISTRIBUTION

**Instance** A binary $m \times n$ matrix $H$ and an integer $w > 0$.

**Question:** Is there a vector $x \in \mathbb{F}_2^n$ of weight $w$, such that $Hx^T = 0$?

Berlekamp, McEliece, and van Tilborg [3] proved that this problem is NP-complete. Nevertheless, many algorithms for calculating the weight distribution have been developed. Some of them are implemented in the software systems related to Coding theory, such as MAGMA, GUAVA, Q-EXTENSION, etc. [1, 7, 8]. The main idea in the common algorithms is to obtain all linear combinations of the basis vectors and to calculate their weights. The efficient algorithms generate all codewords in a sequence, where any codeword is obtained from the previous one by adding only one codeword. They usually use $q$-ary Gray codes (see for example [18]) or an additional matrix (see [9]). The complexity of these algorithms is $O(nq^k)$ for a fixed $q$. Katsman and Tsfasman in [25] proposed a geometric method based on algebraic-geometric codes. Some methods use matroids and Tutte polynomials, geometric lattices [23], or Gröbner bases [5, 16, 28, 30]. The algorithm in [6] is based on the idea of an ideal associated to a binary code, and its main aim is to compute the set of coset leaders, but the algorithms in that paper can be easily reformulated to compute the weight distribution of codes over different finite fields. Bellini and Sala in [2] provided a deterministic algorithm to compute the weight and distance distributions of a binary nonlinear code, which takes advantage of fast Fourier techniques. The binary code in [2] is represented as a set of Boolean functions in numerical normal form (NNF). Efficient calculation of the weight distribution for linear codes over large finite fields is given in [19].

We propose an algorithm for computing the weight distribution of a linear code based on a generalized Walsh-Hadamard transform. The linear codes here are represented by their characteristic vector $\chi$. We obtain a vector whose coordinates are all non-zero weights in the code, by multiplying a special (recursively constructed) integer matrix by $\chi^T$. The complexity for this multiplication is $O(kq^k)$, where $k$ is the dimension of the considered code.

In the binary case, we compute the weight distribution by using algorithms for fast Walsh transform which are easy for implementation [15]. For codes over prime fields with $p > 2$ elements we use an integer matrix of size $\theta(p, k) \times \theta(p, k)$ where $\theta(p, k) = \frac{p^k - 1}{p-1}$. The weight distribution in this case can also be obtained by applying the generalized Walsh transform but then one has to use a $p^k \times p^k$ matrix [24]. For codes over a composite field with $q = p^m$ elements, $m > 1$, we use the trace map and take their images over the prime field $\mathbb{F}_p$.

The considered algorithms are related to butterfly networks and diagrams. A detailed description of the binary butterfly network is presented in [29]. These types of algorithms have very efficient natural implementations with SIMD model of parallelization especially with the CUDA platform [21]. The speedup of the parallel implementation of Walsh-Hadamard transform in GPU can be seen in [4].

We implemented the presented algorithm in a C/C++ program without special optimizations. As input we used randomly generated linear codes with lengths 300, 3000, 30000 and different dimensions over finite fields with 2, 3, 4, 5, and 7 elements. The results of our experiments show that the presented approach is very efficient for codes with large
length and small rate. Such codes are useful to distinguish vectorial Boolean functions up to equivalence (for different types of equivalences) [11, 14].

The paper is organized as follows. In Section 2 we define the basic concepts and prove important assertions that we use in the paper. In Section 3 we present a butterfly algorithm for computing the weight distribution of a binary linear code. Section 4 is devoted to linear codes over a prime field with $p > 2$ elements. In Section 5 we prove some theorems that give the connection of the weight distribution of a linear $[n, k]$ code over a composite finite field with characteristic $p$ and $q = p^m$ elements, with the weight distribution of a linear $[(q - 1)n, mk]$ code over $\mathbb{F}_p$. Section 6 presents the complexity of the algorithms and some experimental results.

2 Preliminaries

Let $\mathbb{F}_q = \{0, \alpha_1 = 1, \alpha_2, \ldots, \alpha_{q-1}\}$ be a field with $q$ elements, and $\mathbb{F}_q^n$ be the $n$-dimensional vector space over $\mathbb{F}_q$. Every $k$-dimensional subspace $C$ of $\mathbb{F}_q^n$ is called a $q$-ary linear $[n, k]$ code (or an $[n, k, q]$ code). The parameters $n$ and $k$ are called length and dimension of $C$, respectively, and the vectors in $C$ are called codewords. The (Hamming) weight $wt(v)$ of a vector $v \in \mathbb{F}_q^n$ is the number of its non-zero coordinates. The smallest weight of a non-zero codeword is called the minimum weight of the code. If $A_i$ is the number of codewords of weight $i$ in $C$, $i = 0, 1, \ldots, n$, then the sequence $(A_0, A_1, \ldots, A_n)$ is called the weight distribution of $C$, and the polynomial $W_C(y) = \sum_{i=0}^{n} A_i y^i$ is the weight enumerator of the code. Obviously, for any linear code $A_0 = 1$ and $A_i = 0$ for $i = 1, \ldots, d - 1$, where $d$ is the minimum weight.

Any $k \times n$ matrix $G$, whose rows form a basis of $C$, is called a generator matrix of the code. The $q$-ary simplex code $S_{q,k}$ is a linear code over $\mathbb{F}_q$ generated by a $k \times \theta(q, k)$ matrix $G_k$ having as columns a maximal set of nonproportional vectors from the vector space $\mathbb{F}_q^k$, $\theta(q, k) = (q^k - 1)/(q - 1)$. In other words, the columns of the matrix represent all points in the projective geometry $PG(k - 1, q)$. For more information about linear codes and their parameters we refer to [20, 23, 27].

Let $C$ be a $k$-dimensional linear code over $\mathbb{F}_q$ and $G$ be a generator matrix of $C$. Without loss of generality we can suppose that $G$ has no zero columns (otherwise we will remove the zero columns).

**Definition 1** The characteristic vector of the code $C$ with respect to its generator matrix $G$ is the vector

$$\chi(C, G) = (\chi_1, \chi_2, \ldots, \chi_{\theta(q, k)}) \in \mathbb{Z}^{\theta(q, k)}$$

(1)

where $\chi_u$ is the number of the columns of $G$ that are equal or proportional to the $u$-th column of $G_k$, $u = 1, \ldots, \theta(q, k)$.

When $C$ and $G$ are clear from the context, we will write briefly $\chi$. Note that $\sum_{u=1}^{\theta(q, k)} \chi_u = n$, where $n$ is the length of $C$.

A code $C$ can have different characteristic vectors depending on the chosen generator matrices of $C$ and the considered generator matrix $G_k$ of the simplex code $S_{q,k}$. If we permute the columns of the matrix $G$ we will obtain a permutation equivalent code to $C$ having the same characteristic vector. Moreover, from a characteristic vector one can restore the columns of the generator matrix $G$, possibly in a different order and/or multiplied by
nonzero elements of the field. This is not a problem for us because equivalent codes have the same weight distributions.

All codewords of the code are the linear combinations of the rows of a given generator matrix $G$. We can easily obtain all nonzero codewords of $C$ using the multiplication

$$
\begin{pmatrix}
G_k^T \\
\alpha_2 G_k^T \\
\vdots \\
\alpha_{q-1} G_k^T
\end{pmatrix} G = \begin{pmatrix}
G_k^T \\
\alpha_2 G_k^T \\
\vdots \\
\alpha_{q-1} G_k^T
\end{pmatrix}.
$$

(2)

To know the weight distribution of the code $C$, it is enough to compute the weights of the rows of the matrix $G_k^T G$.

Further, we consider the matrices $M_k = G_k^T G_k$, $k \in \mathbb{N}$. We denote by $N(M_k)$ the matrix obtained from $M_k$ by replacing all nonzero elements by 1. This matrix is the complement of the incidence matrix of points and hyperplanes in $PG(k - 1, q)$ (for more details on the correspondence between linear codes and multisets of points in a projective geometry see e.g. [12]).

**Lemma 1** Let $C$ be an $[n, k]_q$ code, $G$ be its generator matrix and $\chi$ be the characteristic vector of $C$ with respect to $G$. Then the Hamming weight of the $i$-th row of the matrix $G_k^T G$ (multiplication over $\mathbb{F}_q$) is equal to the $i$-th coordinate of the column vector $N(M_k) \chi^T$ (multiplication over $\mathbb{Z}$), $i = 1, \ldots, \theta(q, k)$.

**Proof** Let $\theta = \theta(q, k)$ for short, and $s_1, \ldots, s_\theta$ be the columns of $G_k$. Since $M_k = (m_{ij}) = G_k^T G_k$, then $m_{ij} = s_i \cdot s_j \in \mathbb{F}_q$, where $x \cdot y = x_1 y_1 + \cdots + x_k y_k \in \mathbb{F}_q$ is the Euclidean inner product of the vectors $x, y \in \mathbb{F}_q^k$ over $\mathbb{F}_q$. Similarly, $v_{ij} = s_i \cdot b_j$, where $b_1, \ldots, b_n$ are the columns of $G$, and $G_k G = (v_{ij})$. From the definition of the characteristic vector $\chi$ we know that $\chi_1$ of the columns of $G$ are proportional to $s_1, \chi_2$ columns are proportional to $s_2$, etc.

For $a \in \mathbb{F}_q$ we define $\mathcal{N}(a) = 0$ if $a = 0$ and $\mathcal{N}(a) = 1$ otherwise. If $v_i = (v_{i1}, \ldots, v_{in})$ is the $i$-th row of the matrix $G_k G$, we have if $\overline{n}[\log_2 j] = 0$

$$w_i = \text{wt}(v_i) = \sum_{j=1}^{n} \mathcal{N}(s_i \cdot b_j) = \sum_{j=1}^{n} \mathcal{N}(s_i \cdot s_{uj})$$

$$= \sum_{j=1}^{\theta} \chi_j \mathcal{N}(s_i \cdot s_j) = \sum_{j=1}^{\theta} \mathcal{N}(m_{ij}) \chi_j = u_i,$$

where $u_i$ is the $i$-th coordinate of $N(M_k) \chi^T$. 

Lemma 1 and (2) show that the coordinates of the vector $N(M_k) \chi^T$ are all weights in a maximal set of codewords in the code $C$ with the following properties: (1) no two codewords in the set are proportional, and (2) any codeword of $C$ is proportional to a codeword belonging to the set. Hence using this matrix by vector multiplication we can obtain the weight distribution of $C$ without calculating all codewords.

To develop a fast algorithm for the proposed matrix by vector multiplication, we use a modified Walsh-Hadamard transform. Let $h(x) = h(x_1, \ldots, x_k)$ be a Boolean function

(&) Springer
in $k$ variables. Discrete Walsh–Hadamard transform $\hat{h}$ of $h$ is the integer valued function $\hat{h} : \mathbb{F}_2^k \rightarrow \mathbb{Z}$, defined by

$$\hat{h}(\omega) = \sum_{x \in \mathbb{F}_2^k} h(x)(-1)^{x \cdot \omega}, \quad \omega \in \mathbb{F}_2^k$$

(3)

where $x \cdot \omega$ is the Euclidean inner product. This transform is equivalent to the multiplication of the truth table of $h$ by the matrix $H_k = \otimes^k \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$. The Kronecker power of a matrix can be represented as a product of sparse matrices [17] that leads to the more effective butterfly algorithm for calculation (fast transform).

3 Binary codes

There is a method based on the fast Walsh-Hadamard transform for the computation of the weight distribution of a given binary linear code. The complexity of this computation is $O(k2^k)$ [24].

In this case the columns of a generator matrix of the simplex code $S_k$ are all nonzero vectors from $\mathbb{F}_2^k$. We take $G_k = (1^T \cdot \cdot \cdot 2^k - 1^T)$, where $\bar{u}$ is the binary representation of the integer $u$, considered as a vector with $k$ coordinates, $1 \leq u \leq 2^k - 1$.

If $C$ is an $[n, k, d]$ binary linear code with a characteristic vector $\chi_C$, then

$$M_k \chi_C^T = (G_k^T G_k) \chi_C^T = \begin{pmatrix} w_1 \\ \vdots \\ w_{2^k-1} \end{pmatrix},$$

where $w_1, \ldots, w_{2^k-1}$ are the weights of all nonzero codewords in $C$.

If $\overline{M}_k = \begin{pmatrix} 0 & 0 & \cdots & 0 \\ 0^T & M_k \end{pmatrix}$, then the matrix $J - 2\overline{M}_k$ is a square $\pm 1$ matrix of order $2^k$, which is equal to the Hadamard matrix of Sylvester type $H_k = \otimes^k \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$ (by $J$ we denote the all 1’s matrix of the corresponding size). It follows that

$$H_k \overline{\chi}_C^T = (J - 2\overline{M}_k) \overline{\chi}_C^T = \begin{pmatrix} n \\ \vdots \\ n - 2w_1 \\ \vdots \\ n - 2w_{2^k-1} \end{pmatrix} = \hat{\chi}_C,$$

where $\hat{\chi}_C$ is the Walsh transform of $\overline{\chi}_C = (0, \chi_C)$, if we consider this characteristic vector as a truth table of a Boolean function. Hence we can obtain the weight distribution of $C$ after applying the Walsh-Hadamard transform on its characteristic vector. Algorithm 1 presents the pseudo code of the corresponding butterfly implementation.
Algorithm 1: Butterfly algorithm for fast walsh transform.

Input: The extended characteristic vector $\bar{x}_C$ with length $2^k$
Output: An updated array $W$ – the result of the transform
1: $j := 1; W := \bar{x}(C)$;
2: while $j < 2^k$ do
3: for $u := 0$ to $2^k - 1$ do
4: if $\bar{u}[\log_2 j] = 0$ then
5: $tt := W[u]$;
6: $W[u] := W[u] + W[u + j]$;
7: $W[u + j] := tt - W[u + j]$;
8: end if
9: end for
10: $j := 2j$;
11: end while

For more details on the butterfly algorithms and diagrams we refer to [22]. The algorithm is very suitable for parallel realization, especially with CUDA parallel computing platform.

4 Codes over prime fields

We define a sequence of matrices $G_k$ as follows:

$$G_1 = (1), \quad G_k = \begin{pmatrix} 0 & 1 & \ldots & p - 1 & 1 \\ G_{k-1} & G_{k-1} & \ldots & G_{k-1} & 0^T \end{pmatrix}, \quad k \in \mathbb{Z}, k \geq 2, \quad (4)$$

where $u = (u, \ldots, u) = u(1, 1, \ldots, 1) = u.1, u = 0, 1, \ldots, p - 1$. The size of the matrix $G_k$ is $k \times \theta(p, k)$. All columns in $G_k$ are pairwise linearly independent, so $G_k$ is a generator matrix of $S_{p,k}$.

Let $C$ be a linear $[n, k, d]$ code over the prime field $\mathbb{F}_p = \{0, 1, \ldots, p - 1\}$ with a characteristic vector $\chi$ with respect to its generator matrix $G$ and the matrix $G_k$ as defined in (4). To obtain the weight distribution of $C$, we need to calculate the product $\mathcal{N}(M_k)\chi^T$.

Using (4) we obtain a recurrence relation for the matrices $M_k$ as follows:

$$M_k = \begin{pmatrix} M_{k-1} & M_{k-1} & \ldots & M_{k-1} & 0^T \\ M_{k-1} & M_{k-1} + J & \ldots & M_{k-1} + (p - 1)J & 1^T \\ M_{k-1} & M_{k-1} + 2J & \ldots & M_{k-1} + 2(p - 1)J & 2^T \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ M_{k-1} & M_{k-1} + (p - 1)J & \ldots & M_{k-1} + (p - 1)^2J & (p - 1)^T \end{pmatrix}, \quad (5)$$

$k \in \mathbb{Z}, k \geq 2, M_1 = (1)$. The matrix $J$ in the above formula is the $\theta(p, k - 1) \times \theta(p, k - 1)$ matrix with all elements equal to 1. The form of the matrix $G_k$ is chosen so to have only additions of matrices in the recurrence relation (5). Denote $\theta(p, k)$ by $\theta_k, k \in \mathbb{N}$. Then $\theta_k = p\theta_{k-1} + 1$ for $k \geq 2$. Unfortunately, there is no comfortable recurrence relation for the
matrices $\mathcal{N}(M_k)$. To overcome this, we introduce the matrices $M_k^{[x]} = \begin{pmatrix} m_1^{[x]} \\ \vdots \\ m_N^{[x]} \end{pmatrix}$, where $m_i^{[x]} = (\omega_0(i), \omega_1(i), \ldots, \omega_{p-1}(i)) \in \mathbb{Z}^p$, $m_i$ is the $i$-th row of $M_k$, and $\omega_u(i) = \sum \{\chi_j : m_{ij} = u, 1 \leq j \leq \theta_k\}$, $u = 0, 1, \ldots, p - 1$. The vector $m_i^{[x]}$ shows how many times each of the elements of the field $\mathbb{F}_p$ occurs in the $i$-th row $v_i$ of the matrix $G^T G'$, where $G'$ is obtained from the considered generator matrix $G$ of $C$ by multiplying its columns by proper nonzero elements of the field so that each column of $G'$ is a column in $G_k$. This vector is called the composition of $v_i$ and is denoted by $\text{comp}(v_i)$ in [26].

**Theorem 1** The $i$-th coordinate $w_i$ of $\mathcal{N}(M_k)\chi^T$ is equal to $n - \omega_0(i)$, where $n$ is the length of the code, and $m_i^{[x]}$ is the $i$-th row of $M_k^{[x]}$.

**Proof** According to the definition of $m_i^{[x]}$ and Lemma 1, we have

$$n - \omega_0(i) = \sum_{j=1}^{\theta_k} \chi_j - \sum \{\chi_j : m_{ij} = 0, 1 \leq j \leq \theta_k\} = \sum \{\chi_j : m_{ij} \neq 0, 1 \leq j \leq \theta_k\} = \sum_{j=1}^{\theta_k} \chi_j N(m_{ij}) = u_i = w_i. \quad \square$$

According to Lemma 1, the coordinates of the vector $\mathcal{N}(M_k)\chi^T = (w_1, w_2, \ldots, w_\theta)^T$ are the weights of all codewords from a maximal subset of the code, where the maximal subset has the following properties: (1) no two codewords in the set are proportional, and (2) any codeword outside this set is proportional to a codeword belonging to the set. Hence if $N_j = \sharp\{i : w_i = j\}$, then the number of codewords of weight $j$ in the code is $A_j = (q - 1)N_j$. According to Theorem 1, $w_i = n - \omega_0(i)$ and so $N_j = \sharp\{i : \omega_0(i) = n - j\}$.

We are looking for a recurrence relation for the matrices $M_k^{[x]}$. Our aim is to use such a relation to obtain a transform matrix which can be represented as a product of sparse matrices. This could give a butterfly algorithm for fast computation.

Using the relation $\theta(p, k) = p\theta(p, k - 1) + 1$, we split the characteristic vector $\chi$ of the $[n, k]_p$ code $C$ into $p + 1$ parts as follows

$$\chi = \left(\chi^{(0)} | \chi^{(1)} | \cdots | \chi^{(p-1)} | \chi^{(p)}\right) \quad (6)$$

where $\chi^{(j)} \in \mathbb{Z}^{\theta(p, k-1)}$, $j = 0, \ldots, p - 1$, and $\chi^{(p)} \in \mathbb{Z}$. Splitting the $i$-th row of $M_k$ similarly to (6), we have $m_i = \left( m_i^{(0)} | m_i^{(1)} | \cdots | m_i^{(p-1)} | m_i^{(p)} \right)$, $m_i^{(j)} \in \mathbb{F}_p^{\theta(p, k-1)}$, $j = 0, \ldots, p - 1$, $m_i^{(p)} \in \mathbb{F}_p$. According to (5), we obtain $m_i^{(s)}[j] = a_{rj} + st$, where $i = t\theta_{k-1} + r$, $1 \leq r \leq \theta_{k-1}$, and $a_{rj}$ are the elements of the matrix $M_{k-1}$, $1 \leq r, j \leq \theta_{k-1}$.
\[ 0 \leq s \leq p - 1, \text{ and } m_{i}^{(p)} = t. \text{ Since } (m_{i}+1)^{[x]} = \sigma (m_{i}^{[x]}) \text{ and } (m_{i}+s)^{[x]} = \sigma^{s} (m_{i}^{[x]}) \text{ where } \sigma \text{ is the right circular shift, } s \geq 1, \text{ then} \]
\[
m_{i}^{[x]} = \left( \sum_{j} \chi_{j} : m_{ij} = u, 1 \leq j \leq \theta (p, k) \right)_{u=0}^{p-1} = \left( \sum_{s=0}^{p-1} \sum \chi_{j}^{(s)} = m_{ij}^{(s')} = u, 1 \leq j' \leq \theta_{k-1} \right)_{u=0}^{p-1} + t \chi(p)_{u=0}^{p-1} \]
\[
= \left( \sum_{s=0}^{p-1} \sum \chi_{j}^{(s)} : m_{ij}^{(s')} = u, 1 \leq j' \leq \theta_{k-1} \right)_{u=0}^{p-1} + t \chi(p)_{u=0}^{p-1} \]
\[
= \sum_{s=0}^{p-1} \left( (a_{r} + st) \chi^{(s)} + t \chi(p) \right)_{u=0}^{p-1} = \sum_{s=0}^{p-1} \left( \sigma^{s} (a_{r}^{[x]}(s')) + t \chi(p) \right)_{u=0}^{p-1}. \]

Hence the following recurrence relation holds
\[
M_{k}^{[x]} = \left( \begin{array}{cccc}
M_{k-1}^{[x^{(0)}]} & + & M_{k-1}^{[x^{(1)}]} & + & \cdots & + & M_{k-1}^{[x^{(p-1)}]} & + & 0^{[x^{(0)}]} \\
M_{k-1}^{[x^{(0)}]} & + & \sigma (M_{k-1}^{[x^{(1)}]}) & + & \cdots & + & \sigma^{p-1} (M_{k-1}^{[x^{(p-1)}]}) & + & 1^{[x^{(0)}]} \\
M_{k-1}^{[x^{(0)}]} & + & \sigma^{2} (M_{k-1}^{[x^{(1)}]}) & + & \cdots & + & \sigma^{2(p-1)} (M_{k-1}^{[x^{(p-1)}]}) & + & 2^{[x^{(0)}]} \\
\vdots & & & & & & & & \\
0^{[x^{(0)}]} & + & 1^{[x^{(1)}]} & + & \cdots & + & (p-1)^{[x^{(p-1)}]} & + & 1^{[x^{(0)}]} \\
\end{array} \right) \tag{7} \]

So we can use permutations and additions to compute \(M_{k}^{[x]} \) from \(M_{k-1}^{[x^{(0)}]} \), \(M_{k-1}^{[x^{(1)}]} \), \ldots, \(M_{k-1}^{[x^{(p-1)}]} \) and \(\chi^{(p)} \).Moreover, \(s^{[x]} \) can be obtained from \(0^{[x]} \) by right circular shift operation. Note that all coordinates of \(0^{[x]} \) are 0’s except the first column whose elements are equal to the sum of all coordinates of \(\chi \).

Example 1 If \( p = 3 \) the recurrence relation (7) is
\[
M_{k}^{[x]} = \left( \begin{array}{cccc}
M_{k-1}^{[x^{(0)}]} & + & M_{k-1}^{[x^{(1)}]} & + & M_{k-1}^{[x^{(2)}]} & + & 0^{[x^{(3)}]} \\
M_{k-1}^{[x^{(0)}]} & + & \sigma (M_{k-1}^{[x^{(1)}]}) & + & \sigma^{2} (M_{k-1}^{[x^{(2)}]}) & + & 1^{[x^{(3)}]} \\
M_{k-1}^{[x^{(0)}]} & + & \sigma^{2} (M_{k-1}^{[x^{(1)}]}) & + & \sigma (M_{k-1}^{[x^{(2)}]}) & + & 2^{[x^{(3)}]} \\
0^{[x^{(0)}]} & + & 1^{[x^{(1)}]} & + & \cdots & + & (p-1)^{[x^{(p-1)}]} & + & 1^{[x^{(3)}]} \\
\end{array} \right) \tag{8} \]

Let \( k = 3 \) and \( \chi = (0, 4, 3, 2, 0, 8, 5, 1, 1, 4, 3, 2, 3) \). We split \( \chi \) into 4 parts
\[
\chi^{(0)} = (0, 4, 3, 2), \quad \chi^{(1)} = (0, 8, 5, 1), \quad \chi^{(2)} = (1, 4, 3, 2), \quad \chi^{(3)} = 3. \]

Since \( M_{2} = \begin{pmatrix}
1 & 1 & 1 & 0 \\
1 & 2 & 0 & 1 \\
1 & 0 & 2 & 2 \\
0 & 1 & 2 & 1 \\
\end{pmatrix} \), we have
\[
M_{2}^{[x^{(0)}]} = \begin{pmatrix}
2 & 7 & 0 \\
3 & 2 & 4 \\
4 & 0 & 5 \\
0 & 6 & 3 \\
\end{pmatrix}, \quad M_{2}^{[x^{(1)}]} = \begin{pmatrix}
1 & 1 & 3 & 0 \\
5 & 1 & 8 \\
8 & 0 & 6 \\
0 & 9 & 5 \\
\end{pmatrix}, \quad M_{2}^{[x^{(2)}]} = \begin{pmatrix}
2 & 8 & 0 \\
3 & 3 & 4 \\
4 & 1 & 5 \\
1 & 6 & 3 \\
\end{pmatrix}. \]
We calculate $M_3[\chi]$ from $M_2^{[\chi^{(0)}]}$, $M_2^{[\chi^{(1)}]}$, $M_2^{[\chi^{(2)}]}$ and $\chi^{(3)}$ by (8):

\[
M_3^{[\chi]} = \frac{1}{271}
\begin{pmatrix}
2 \ 7 \ 0 \\
3 \ 2 \ 4 \\
4 \ 0 \ 5 \\
0 \ 6 \ 3 \\
2 \ 7 \ 0 \\
3 \ 2 \ 4 \\
4 \ 0 \ 5 \\
0 \ 6 \ 3 \\
9 \ 0 \ 0
\end{pmatrix}
\begin{pmatrix}
1 \ 1 \ 3 \ 0 \\
5 \ 1 \ 8 \\
8 \ 0 \ 6 \\
0 \ 9 \ 5 \\
0 \ 1 \ 1 \ 3 \\
8 \ 5 \ 1 \\
6 \ 8 \ 0 \\
5 \ 0 \ 9 \\
0 \ 1 \ 4 \ 0
\end{pmatrix}
\begin{pmatrix}
2 \ 8 \ 0 \\
4 \ 1 \ 5 \\
1 \ 6 \ 3 \\
8 \ 0 \ 2 \\
3 \ 4 \ 3 \\
1 \ 5 \ 4 \\
6 \ 3 \ 1 \\
0 \ 2 \ 8 \\
0 \ 0 \ 1 \ 0
\end{pmatrix}
\begin{pmatrix}
3 \ 0 \ 0 \\
3 \ 0 \ 0 \\
3 \ 0 \ 0 \\
0 \ 3 \ 0 \\
0 \ 3 \ 0 \\
0 \ 3 \ 0 \\
0 \ 3 \ 0 \\
0 \ 0 \ 3 \\
0 \ 0 \ 3
\end{pmatrix}
\begin{pmatrix}
8 \ 2 \ 8 \ 0 \\
1 \ 4 \ 6 \ 1 \ 6 \\
1 \ 9 \ 1 \ 1 \ 6 \\
4 \ 2 \ 1 \ 1 \\
1 \ 0 \ 1 \ 1 \ 5 \\
1 \ 4 \ 1 \ 4 \ 8 \\
1 \ 1 \ 6 \ 1 \ 9 \\
1 \ 1 \ 2 \ 1 \ 3 \\
1 \ 5 \ 9 \ 1 \ 2
\end{pmatrix}
\]

Next we define one more sequence of matrices connected to $M_k^{[\chi]}$ which we use in the algorithms.

**Definition 2** Let $k \in \mathbb{N}$ and $\chi = (\chi_1, \ldots, \chi_{\theta(p,k)}) \in \mathbb{Z}^{\theta(p,k)}$. The matrices $M_k^{[\chi]}(l)$, $l = 1, \ldots, k$, are defined recursively as follows

1. $M_k^{[\chi]}(k) = M_k^{[\chi]}$.
2. For $1 \leq l < k$, the vector $\chi$ is split into $p + 1$ parts as in (6) and

\[
M_k^{[\chi]}(l) = \begin{pmatrix}
M_k^{[\chi^{(0)}]}(l) \\
M_k^{[\chi^{(1)}]}(l) \\
\vdots \\
M_k^{[\chi^{(p-1)}]}(l) \\
M_1^{[\chi^{(p)}]}
\end{pmatrix}.
\]

The matrix $M_k^{[\chi]}(1)$ is a $\theta_k \times p$ matrix with rows $M_1^{[\chi_i]}$, $i = 1, \ldots, \theta_k$, where $\chi = (\chi_1, \ldots, \chi_{\theta_k})$. Since $M_1 = (1)$, the columns of the matrix $M_k^{[\chi]}(1)$ are zero vectors except the second one which is equal to $\chi$.

Note that the last row of the matrices $M_k^{[\chi]}(l)$ for $l = 1, \ldots, k - 1$ is the same, namely $M_1^{[\chi^{(p)}]} = (0, \chi^{(p)}, 0, \ldots, 0)$. Furthermore, the row before the last one in $M_k^{[\chi]}(l)$ is the same for $l = 1, \ldots, k - 2$. Actually, for all $l < k$ there are rows equal to $M_1^{[\chi^{(p)}]}$ in the matrix $M_k^{[\chi]}(l)$ that are the same as in the previous matrices $M_k^{[\chi]}(l')$, $l' < l$. We call them *inactive rows*. There are $\theta_{k-l}$ inactive rows in $M_k^{[\chi]}(l)$, $l = 2, \ldots, k - 1$. 
Example 2 Let $p = 3$, $k = 3$ and $\chi = (0, 4, 3, 2, 0, 8, 5, 1, 1, 4, 3, 2, 3)$. Then

$$M_3^{[\chi]}(1) = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 3 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \\ 0 & 8 & 0 \\ 0 & 5 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 4 & 0 \\ 0 & 3 & 0 \\ 0 & 2 & 0 \\ 0 & 3 & 0 \end{pmatrix}, \quad M_3^{[\chi]}(2) = \begin{pmatrix} 2 & 7 & 0 \\ 3 & 2 & 4 \\ 4 & 0 & 5 \\ 0 & 6 & 3 \\ 1 & 3 & 0 \\ 5 & 1 & 8 \\ 8 & 0 & 6 \\ 0 & 9 & 5 \\ 2 & 8 & 0 \\ 3 & 3 & 4 \\ 4 & 1 & 5 \\ 1 & 6 & 3 \\ 0 & 3 & 0 \end{pmatrix}, \quad M_3^{[\chi]}(3) = \begin{pmatrix} 8 & 28 & 0 \\ 14 & 6 & 16 \\ 19 & 1 & 16 \\ 4 & 21 & 11 \\ 10 & 11 & 15 \\ 14 & 14 & 8 \\ 11 & 16 & 9 \\ 11 & 12 & 13 \\ 15 & 9 & 12 \\ 8 & 13 & 15 \\ 9 & 10 & 17 \\ 12 & 12 & 12 \\ 9 & 17 & 10 \end{pmatrix}$$

Till the end of this section, we present an algorithm for calculating $M_k^{[\chi]}$ computing successively $M_k^{[\chi]}(1)$, $M_k^{[\chi]}(2)$, ..., $M_k^{[\chi]}(k)$, $M_k^{[\chi]}(k)$. The pseudo code of the main procedure is given in Algorithm 2.

Algorithm 2 Main procedure.

**Input:** a prime $p$, an integer $k$, and a vector $\chi$ of length $\theta(p, k)$ with integer coordinates 

$k$ is the dimension of the considered $p$-ary code given by its characteristic vector $\chi$

**Output:** the array $H \{ H = M_k^{[\chi]} \}$

1: $H := M_k^{[\chi]}(1)$
2: $\theta_1 := 1$;
3: for $l := 2$ to $k$ do
4: $\text{Initialize an array } a \text{ of length } k, a := 0 \{ \text{a help array for monitoring the inactive rows} \}$
5: $\theta_0 := \theta_1$;
6: $\theta_1 := \theta(p, l) = \frac{p^l-1}{p-1} = p\theta_0 + 1$;
7: $r := 0$;
8: while $r < \theta$ do
9: $r_0 := r \{ r_0 + 1 \text{ is the index of the first row of the considered submatrix} \}$
10: $r := r + \theta_1 \{ \text{the index for the last row of the considered submatrix} \}$
11: NewH($H$, $r_0$, $r$, $\theta_0$) \{Computes $M_k^{[\chi^{(n)}]}$ for the current part of $\chi$ \}
12: $s := l$
13: $a[s] := a[s] + 1$
14: while $a[s] = q$ do
15: $r := r + 1 \{ \text{skipping an inactive row} \}$
16: $a[s] := 0$
17: $s := s + 1$
18: $a[s] := a[s] + 1$
19: end while
20: end while
21: end for
Algorithm 3 shows how to obtain $M_k^{[x]}(l)$ from $M_k^{[x]}(l-1)$. It consists of three main transformations which we call ADD0, LAST ROW and ALLROWS. Let explain them in the case $l = k$. We start with the array

$$ M_k^{[x]}(k-1) = \begin{pmatrix} M_k^{[x]}(0) \\ M_{k-1}^{[x]}(1) \\ \vdots \\ M_{k-1}^{[x]}(p) \\ 0, \chi(p), 0, \ldots, 0 \end{pmatrix} $$

1. **ADD0**: First we apply the left circular shift operation on the last row of the matrix $M_k^{[x]}(k-1)$. Then we add the obtained vector $\text{lcs}(M_k^{[x]}(p)) = (\chi(p), 0, \ldots, 0)$ to all rows of $M_k^{[x]}(k-1)$.

$$ M_k^{[x]}(k-1) = \begin{pmatrix} M_k^{[x]}(0) \\ M_{k-1}^{[x]}(1) \\ \vdots \\ M_{k-1}^{[x]}(p-1) \\ 0, \chi(p), 0, \ldots, 0 \end{pmatrix} \rightarrow \begin{pmatrix} M_k^{[x]}(0) \\ M_{k-1}^{[x]}(1) \\ \vdots \\ M_{k-1}^{[x]}(p) \\ 0, \chi(p), 0, \ldots, 0 \end{pmatrix} $$

2. **LAST ROW**: In this step we calculate the last row of $M_k^{[x]}(k)$ equal to

$$ \sum_{s=0}^{p-1} s^{[x]} + 1^{[x]}(p) = \sum_{i=1}^{\theta_k-1} \chi_i, \sum_{i=\theta_k-1+1}^{2\theta_k-1} \chi_i + \chi(p), \ldots, \sum_{i=\theta_k-\theta_{k-1}}^{\theta_k-1} \chi_i $$

$$ = \left( \sum_{i=0}^{p-1} \omega_{0,i}, \sum_{i=0}^{p-1} \omega_{1,i}, \ldots, \sum_{i=0}^{p-1} \omega_{p-1,i} \right). $$
where \((\omega_{j,0}, \omega_{j,1}, \ldots, \omega_{j,p-1})\) is the first row of the matrix \(M_{k-1}^{[x(j)]}\), 
\(j = 0, 2, \ldots, p - 1\), and \((\omega_{1,0}, \ldots, \omega_{1,p-1})\) is the first row of the submatrix \(M_{k-1}^{[x(1)]}\) transformed in ADD0.

\begin{algorithm}
\textbf{Input}: The array \(H\) and the integers \(r_0, r, \theta\) \{parameters that fix a considered submatrix\}
\textbf{Output}: an updated array \(H\) \{in range of the considered submatrix\}
1: Initialize the auxiliary array \(T\) of size \(p \times p\)
2: \textbf{for} \(i := 1\) \textbf{to} \(\theta\) \textbf{do}
3: \quad \(H[r_0 + \theta + i] := H[r_0 + \theta + i] + \lcs(H[r])\) \{The transformation ADD0\}
4: \textbf{end for}
5: \(H[r] := (\sum_{i=0}^{p-1} H[r_0 + 1, i], \sum_{i=0}^{p-1} H[r_0 + \theta + 1, i], \ldots, \sum_{i=0}^{p-1} H[r_0 + (p - 1)\theta + 1, i])\);
\textbf{LASTROW}\n6: \textbf{for} \(i := 1\) \textbf{to} \(\theta\) \textbf{do}
7: \quad \textbf{for} \(j := 0\) \textbf{to} \(p - 1\) \textbf{do}
8: \quad \quad \(T[j] := H[r_0 + j\theta + i]\)
9: \quad \textbf{end for}
10: \(H[r_0 + i] := T[0] + T[1] + \cdots + T[p - 1]\)
11: \textbf{end for}
12: \(H[r_0 + j\theta + i] := T[0] + \sigma^{j}(T[1]) + \cdots + \sigma^{j(p-1)}(T[p - 1])\) \{ALLRows\}
13: \textbf{end for}
14: \textbf{end for}
\end{algorithm}

3. ALLROWS: This transformation consists of \(p\) similar steps ALLROWS\([J]\), \(j = 0, 1, \ldots, p - 1\), repeated \(\theta_{k-1}\) times. To realize this transformation, we use an auxiliary \(p \times p\) array \(T\). ALLROWS\([J]\) acts on \(T\) as follows: ALLROWS\([0](T) = T[0] + T[1] + \cdots + T[p - 1]\), ALLROWS\([J](T) = T[0] + \sigma^{j}(T[1]) + \cdots + \sigma^{j(p-1)}(T[p - 1])\) for \(j > 0\).

In the beginning \(T\) consists of the first rows of all submatrices \(M_{k-1}^{[x(j)]}\), and in the \(i\)-th step \(T\) consists of the \(i\)-th rows of these submatrices. Hence the transformation ALLROWS gives us

\[
\begin{pmatrix}
M_{k-1}^{[x(0)]} + M_{k-1}^{[x(1)]} + \cdots + M_{k-1}^{[x(p-1)]} & 0^{[x(p)]} \\
M_{k-1}^{[x(0)]} + \sigma(M_{k-1}^{[x(1)]}) + \cdots + \sigma^{p-1}(M_{k-1}^{[x(p-1)]}) & 1^{[x(p)]} \\
M_{k-1}^{[x(0)]} + \sigma^{2}(M_{k-1}^{[x(1)]}) + \cdots + \sigma^{2(p-1)}(M_{k-1}^{[x(p-1)]}) & 2^{[x(p)]} \\
\vdots & \vdots \\
M_{k-1}^{[x(0)]} + \sigma^{p-2}(M_{k-1}^{[x(1)]}) + \cdots + \sigma^{p-2}(M_{k-1}^{[x(p-1)]}) & (p - 1)^{[x(p)]} \\
0^{[x(0)]} + 1^{[x(1)]} + \cdots + (p - 1)^{[x(p)]} & 1^{[x(p)]}
\end{pmatrix} = M_{k}^{[x]}. 
\]

We keep the inactive rows unchanged in the computation of \(M_{k}^{[x]}(l)\) from \(M_{k}^{[x]}(l - 1)\), and apply the transformations described above to obtain \(M_{k}^{[x']}(l)\) from \(M_{k}^{[x']}(l - 1)\) where \(x'\) is a suitable part of \(x\).
Example 3 Let \( q = 3 \), \( k = 3 \), and \( \chi = (0, 4, 3, 2, 0, 8, 5, 1, 1, 4, 3, 2, 3) \). Applying Algorithms 2–3 we have

\[
M_3^{[\chi]}(1)  \begin{array}{cccc}
0, & 0, & 0, & 0, 0, 0, 2, 7, 0 \\
0, & 4, & 0, & 0, 2, 4, 0, 3, 2, 4 \\
0, & 3, & 0, & 0, 3, 0, 3, 2, 4, 4, 0, 5 \\
0, & 2, & 0, & 0, 2, 0, 0, 6, 3, 0, 6, 3 \\
0, & 0, & 0, & 0, 0, 0, 0, 1, 1, 3, 0, 0 \\
0, & 8, & 0, & 0, 1, 8, 0, 5, 1, 8, 0, 6 \\
0, & 5, & 0, & 0, 5, 0, 0, 5, 0, 5, 0, 6 \\
0, & 1, & 0, & 0, 1, 0, 0, 9, 5, 0, 9, 5 \\
0, & 1, & 0, & 0, 1, 0, 0, 1, 0, 2, 8, 0 \\
0, & 4, & 0, & 0, 2, 4, 0, 2, 4, 0, 3, 4 \\
0, & 3, & 0, & 0, 3, 0, 0, 3, 0, 4, 1, 5 \\
0, & 2, & 0, & 0, 2, 0, 1, 6, 3, 1, 6, 3 \\
0, & 3, & 0, & 0, 3, 0, 0, 3, 0, 0, 3, 0 \\
\end{array}
\]

\[
M_3^{[\chi]}(2)  \begin{array}{cccc}
2, 7, 0 & 2, 7, 0 & 2, 7, 0 & 8, 28, 0 \\
3, 2, 4 & 3, 2, 4 & 3, 2, 4 & 14, 6, 16 \\
4, 0, 5 & 4, 0, 5 & 4, 0, 5 & 14, 6, 16 \\
0, 6, 3 & 0, 6, 3 & 0, 6, 3 & 0, 6, 3 \\
1, 13, 0 & 4, 13, 0 & 4, 13, 0 & 14, 14, 8 \\
5, 1, 8 & 8, 1, 8 & 8, 1, 8 & 14, 14, 8 \\
8, 0, 6 & 11, 0, 6 & 11, 0, 6 & 11, 16, 9 \\
0, 9, 5 & 3, 9, 5 & 3, 9, 5 & 11, 16, 9 \\
2, 8, 0 & 2, 8, 0 & 15, 9, 12 & 15, 9, 12 \\
3, 3, 4 & 3, 3, 4 & 3, 3, 4 & 15, 9, 12 \\
4, 1, 5 & 4, 1, 5 & 4, 1, 5 & 8, 13, 15 \\
1, 6, 3 & 1, 6, 3 & 1, 6, 3 & 9, 10, 17 \\
0, 3, 0 & 0, 3, 0 & 9, 17, 10 & 9, 17, 10 \\
\end{array}
\]

To explain more formally the main algorithm we introduce a matrix representation of the transform steps. We put all rows of \( M_3^{[\chi]}(l) \) in one row vector of length \( p \theta(p, k) \) denoted by \( \hat{M}_3^{[\chi]}(l) \). We denote \( \hat{M}_3^{[\chi]}(k) = \hat{M}_3^{[\chi]}(k) \) and \( \hat{\chi} = \hat{M}_3^{[\chi]}(1) \) for short.

In the following theorem, we use matrices of three types, namely:

- the \( p \times p \) permutation matrix \( P = \begin{pmatrix} 0 & 1 \\ I_{p-1} & 0^T \end{pmatrix} \) which realizes the circular right shift operation. Then \( P^0 = I_p \), and \( P^j \) realizes the circular shift right operation by \( j \) positions;
- the \( p \times p \) matrices \( E_j, j = 0, 1, \ldots, p-1 \), where the \( j+1 \)-th row of \( E_j \) is the all-ones vector, and the other rows of the matrix are zero vectors;
- the matrices \( T_{k,l} \) for \( k, l \in \mathbb{Z}, 2 \leq l \leq k \). We define these matrices in the following way:

1) If \( k = l = 2 \), then

\[
T_{2,2} = \begin{pmatrix}
I_p & I_p & \ldots & I_p & P^{-1} \\
I_p & P & P^2 & \ldots & P^{p-1} & I_p \\
I_p & P^2 & P^4 & \ldots & P^{2(p-1)} & P \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
I_p & P^{p-1} & P^{2(p-1)} & \ldots & P^{(p-1)^2} & P^{p-2} \\
E_0 & E_1 & E_2 & \ldots & E_{p-1} & E_1 \\
\end{pmatrix}
\]
2) If \( k > l \), then
\[
T_{k,l} = \left( I_p \otimes T_{k-1,l} 0 \right) \quad (10)
\]
3) If \( k = l > 2 \) then
\[
T_{k,k} = \begin{pmatrix}
I_\theta \otimes I_p & I_\theta \otimes I_p & \ldots & I_\theta \otimes I_p & I \otimes P^{-1} \\
I_\theta \otimes I_p & I_\theta \otimes P & \ldots & I_\theta \otimes P^{p-1} & I \otimes P \\
& \ddots & \ddots & \ddots & \ddots \\
& & I_\theta \otimes P^{p-1} & \ldots & I_\theta \otimes P^{(p-1)} & I \otimes P^{p-2} \\
E_0 & E_1 & \ldots & E_{p-1} & 0 & I_p
\end{pmatrix} \quad (11)
\]
Here \( \otimes \) means Kronecker product and \( \theta = \theta(p, k - 1) \).

**Theorem 2** Let \( \chi \) be a characteristic vector of an \([n, k; q]\)-code. Then
\[
\left( \tilde{M}_k^{[\chi]}(l) \right)^T = T_{k,l} \left( \tilde{M}_k^{[\chi]}(l-1) \right)^T, \quad l = 2, \ldots, k,
\]
and
\[
\left( \tilde{M}_k^{[\chi]} \right)^T = T_{k,k} T_{k,k-1} \cdots T_{k,2} \tilde{\chi}^T \quad (13)
\]

**Proof** Let \( k = 2 \). Then \( \theta(p, 2) = p + 1 \), \( M_2 \) is a \((p + 1) \times (p + 1)\) matrix, and the characteristic vector \( \chi \) has length \( p + 1 \), let \( \chi = (\chi_0, \chi_1, \ldots, \chi_p) \). To obtain \( M_2^{[\chi]}(2) \), we have to apply the transformations ADD0, LASTROW and ALLROWS to \( M_2^{[\chi]}(1) = \begin{pmatrix}
M_1^{[\chi_0]} \\
\vdots \\
M_1^{[\chi_{p-1}]} \\
M_1^{[\chi_p]}
\end{pmatrix} \) (see Definition 2). These three transformations have matrix representations. The transform matrices in this case are square matrices of size \( q(q + 1) \). The three transformation matrices corresponding to ADD0, LASTROW and ALLROWS, respectively, are
\[
T_0 = \begin{pmatrix}
I_p & 0 & \cdots & 0 & 0 \\
0 & I_p & \cdots & 0 & p^{-1} \\
& \ddots & \ddots & \ddots & \ddots \\
0 & 0 & \cdots & I_p & 0 \\
0 & 0 & \cdots & 0 & I_p
\end{pmatrix}, \quad T_{\text{last}} = \begin{pmatrix}
I_p & 0 & \cdots & 0 & 0 \\
0 & I_p & \cdots & 0 & 0 \\
& \ddots & \ddots & \ddots & \ddots \\
0 & 0 & \cdots & I_p & 0 \\
E_0 & E_1 & \cdots & E_{q-1} & 0
\end{pmatrix},
\]
\[
T_{\text{all}} = \begin{pmatrix}
I_p & I_p & I_p & \cdots & I_p & 0 \\
I_p & P & P^2 & \cdots & P^{p-1} & 0 \\
I_p & P^2 & P^4 & \cdots & P^{2(p-1)} & 0 \\
& \ddots & \ddots & \ddots & \ddots & \ddots \\
I_p & P^{p-1} & P^{2(p-1)} & \cdots & P^{(p-1)^2} & 0 \\
0 & 0 & 0 & \cdots & 0 & I_p
\end{pmatrix}
\]
The matrix $T_{2,2}$ is the product of the above matrices:

$$T_{2,2} = T_{all}T_{last}T_0 = \begin{pmatrix} I_p & I_p & I_p & \cdots & I_p & p^{-1} \\ I_p & P & P^2 & \cdots & p^{p-1} & I_p \\ I_p & P^2 & P^4 & \cdots & p^{2(p-1)} & P \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ I_p & p^{p-1} & p^{2(p-1)} & \cdots & p^{(p-1)^2} & p^{p-2} \\ E_0 & E_1 & E_2 & \cdots & E_{q-1} & E_1 \end{pmatrix}. $$

Thus $(\hat{M}_2^{[x]}_1)^T = T_{2,2}(\hat{M}_2^{[x]}_1(1))T$.

Let $k > 2$. We assume that the theorem holds for every $k' \in \mathbb{Z}$ where $2 \leq k' < k$. We split the characteristic vector $\chi \in \mathbb{Z}^{\theta(p,k)}$ into $p + 1$ parts according (6).

If $k > l$ then

$$M^{[x]}_k(l) = \begin{pmatrix} M^{[x]^{(0)}}_{k-1}(l) \\ M^{[x]^{(1)}}_{k-1}(l) \\ \vdots \\ M^{[x]^{(p-1)}}_{k-1}(l) \\ M^{[x]^{(p)}}_1(l) \end{pmatrix} \quad \text{and} \quad M^{[x]}_k(l-1) = \begin{pmatrix} M^{[x]^{(0)}}_{k-1}(l-1) \\ M^{[x]^{(1)}}_{k-1}(l-1) \\ \vdots \\ M^{[x]^{(p-1)}}_{k-1}(l-1) \\ M^{[x]^{(p)}}_1 \end{pmatrix}. $$

Following the induction hypothesis we have

$$(\hat{M}^{[x]}_{k-1}(l))T = T_{k-1,l}(\hat{M}^{[x]}_{k-1}(l-1))T, \quad s = 0, 1, \ldots, p - 1.$$ 

So the assertion follows directly.

If $k = l$ we have $M^{[x]}_k(k) = M^{[x]}_k$ and $M^{[x]}_k(k-1) = \cdots$ and we have to apply (7). It turns out that

$$\hat{M}^{[x]}_k = T_{k,k} \left( \hat{M}^{[x]^{(0)}}_{k-1} \left| \hat{M}^{[x]^{(1)}}_{k-1} \right| \cdots \left| \hat{M}^{[x]^{(p-1)}}_{k-1} \right| \hat{M}^{[x]^{(p)}}_1 \right)^T = T_{k,k} \hat{M}^{[x]}_k(k-1).$$

The main assertion follows directly.  

\[\square\]

5 Codes over composite fields

Let $\mathbb{F}_q = \{0, \alpha_1 = 1, \alpha_2, \ldots, \alpha_{q-1}\}$ be a finite field with $q$ elements, where $q = p^m$, $p$ is prime and $m > 1$. We need a basis $\beta_1 = 1, \beta_2, \ldots, \beta_m$ of $\mathbb{F}_q$ over the prime field $\mathbb{F}_p$.

Let $\text{Tr} : \mathbb{F}_q \to \mathbb{F}_p$ denote the trace map, and $\text{Tr}(a) = (\text{Tr}(a_1), \ldots, \text{Tr}(a_m)) \in \mathbb{F}_p$ for $a \in \mathbb{F}_q^n$. Let $C$ be a $[n, k, d]_q$ linear code with a generator matrix $G$, and $G' = (G|\alpha_2 G| \cdots |\alpha_{q-1} G)$. The code $\text{Tr}(C) = \{\text{Tr}(c) | c \in C\}$ is the trace code of the linear $q$-ary code $C$. $\text{Tr}(c)$ is a linear code over the prime field $\mathbb{F}_p$ with the same length as $C$ but its dimension is less than or equal to $mk$ [32]. Therefore instead of $\text{Tr}(C)$, we consider the trace code of $C'$, where $C'$ is the code generated by the matrix $G'$ with parameters $[(q - 1)n, k, (q - 1)d]_q$.  

\[\odot\text{Springer}\]
**Lemma 2** The dimension of the code $Tr(C')$ is equal to $mk$.

**Proof** If $u_1, \ldots, u_k$ and $v_1, \ldots, v_k$ are the rows of $G$ and $G'$, respectively, then $v_i = (u_i | a_2 u_i | \cdots | a_q u_i)$. Let $\beta_1 = 1, \beta_2, \ldots, \beta_m$ be a basis of $\mathbb{F}_q$ over $\mathbb{F}_p$. We prove that $Tr(\beta_i v_j), i = 1, \ldots, m, j = 1, \ldots, k$, is a basis of the code $Tr(C')$.

Suppose that $\sum_{i=1}^{m} \sum_{j=1}^{k} \lambda_{ij} Tr(\beta_i v_j) = 0$, $\lambda_{ij} \in \mathbb{F}_p$. It turns out that

$$\sum_{i=1}^{m} \sum_{j=1}^{k} \lambda_{ij} Tr(\alpha_s \beta_i u_j) = \sum_{i=1}^{m} \sum_{j=1}^{k} \lambda_{ij} \alpha_s \beta_i u_j = 0, \forall s \in \{1, 2, \ldots, q - 1\}.$$

Hence

$$Tr(\alpha_s \sum_{i=1}^{m} \sum_{j=1}^{k} \lambda_{ij} \beta_i u_j) = 0, \forall s \in \{1, 2, \ldots, q - 1\}.$$ 

If $\sum_{i=1}^{m} \sum_{j=1}^{k} \lambda_{ij} \beta_i u_j \neq 0$ then $\alpha_s (\sum_{i=1}^{m} \sum_{j=1}^{k} \lambda_{ij} \beta_i u_j) = 0, s = 1, 2, \ldots, q - 1$, are all nonzero elements of the field and therefore some of their traces must be nonzero elements of $\mathbb{F}_p$ - a contradiction. This proves that $\sum_{i=1}^{m} \sum_{j=1}^{k} \lambda_{ij} \beta_i u_j = 0$. Since $u_1, \ldots, u_k$ is a basis of the code $C$ then

$$\sum_{i=1}^{m} \lambda_{ij} \beta_i = 0, \forall j = 1, 2, \ldots, k.$$ 

But $\beta_1, \beta_2, \ldots, \beta_m$ is a basis of $\mathbb{F}_q$ over $\mathbb{F}_p$, so $\lambda_{ij} = 0$ for all $i = 1, \ldots, m, j = 1, \ldots, k$. Hence the vectors $Tr(\beta_i v_j)$ are linearly independent and the dimension of $Tr(C')$ is $mk$. \qed

**Corollary 1** The codes $C$ and $Tr(C')$ have the same number of codewords, namely $q^k = p^{mk}$.

Let $c = (c_1, \ldots, c_n) \in C$ and $c_T = Tr(c | a_2 c | \cdots | a_q c)$. If $c_i \neq 0$ then $\{c_i, a_2 c_i, \ldots, a_q c_i\} = \mathbb{F}^*_q$. Hence $p^m - p^{m-1}$ of the elements in the set $\{Tr(c_i), Tr(a_2 c_i), \ldots, Tr(a_q c_i)\}$ are nonzero. Hence

$$wt(c_T) = (p^m - p^{m-1})wt(c).$$ 

It turns out that the minimum weight of $Tr(C')$ is

$$d_T = (p^m - p^{m-1})d = \frac{q(p-1)}{p}d.$$ 

So we obtain the following proposition

**Proposition 1** If $C$ is an $[n, k, d]$ linear code over $\mathbb{F}_q$, $q = p^m$, $p$ - prime, $m > 1$, then $Tr(C')$ is a $[(q-1)n, mk, \frac{q(p-1)d}{p}]_p$ code. Moreover, if $W(y) = \sum_{i=1}^{n} A_i y^d$ is the weight enumerator of the code $C$, then the weight enumerator of $Tr(C')$ is

$$W_T(y) = \sum_{i=0}^{n} A_i y^{q(p-1)i/p}.$$
Proposition 1 shows that we can use the weight distribution of the code $\text{Tr}(C')$ over the prime field $\mathbb{F}_p$ to obtain the weight distribution of the $q$-ary linear code $C$. That is why our algorithm is implemented only for codes over a prime field.

6 Complexity of the algorithms and experimental results

We consider codes over a prime field $\mathbb{F}_p$ with length $n < 2^{32}$ and number of codewords $p^k < 2^{64}$, so we need 32-bit integers for the weights of codewords and 64-bit integers for the number of codewords with a given weight. Therefore we use only basic integer types and operations with them. To calculate the weight distribution of a linear code, we use two arrays with 32-bit integers, namely $H$ of size $\theta(p, k) \times p$ and $T$ of size $p \times p$. The total memory we need (without counting the generator matrix) is $p \theta(p, k) + p^2 + 2n + C$ 32-bit units, where we add $2n$, because the weight distribution is a vector of length $n$ consisting of 64-bit integers, and a constant $C$ for the other variables in the algorithms. If we use the reduced weighted distribution, we will have one column less in the array $H$, so we have to subtract $\theta(p, k)$ from the above expression.

The main procedure computes the array $H$ in $k - 1$ steps. In the $l$-th step of the procedure, there are $a_l$ active and $b_l$ inactive rows, where $a_l + b_l = \theta(p, k)$, $a_l = p^{k-l}\theta(p, l)$, $b_l = \theta(p, k-l)$, $l = 2, 3, \ldots, k$. The inactive rows remain unchanged. Any element in an active row is calculated in Algorithm 3 as a sum with $p$ summands. There are $a_l$ active rows.

| $q$ | $k$ | $n = 30$ | $n = 300$ | $n = 3000$ | $n = 30000$ |
|-----|-----|----------|-----------|-------------|-------------|
|     |     | NEW | OLD | NEW | OLD | NEW | OLD | NEW | OLD |
| 2   | 23  | 0.190 | 0.084 | 0.132 | 0.139 | 0.131 | 0.700 | 0.133 | 6.064 |
| 2   | 24  | 0.214 | 0.168 | 0.268 | 0.278 | 0.266 | 1.412 | 0.271 | 11.665 |
| 2   | 25  | 0.552 | 0.337 | 0.552 | 0.549 | 0.552 | 2.816 | 0.552 | 23.649 |
| 2   | 26  | 1.146 | 0.637 | 1.144 | 1.107 | 1.148 | 5.595 | 1.150 | 47.001 |
| 3   | 13  | 0.039 | 0.015 | 0.101 | 0.032 | 0.101 | 0.190 | 0.103 | 1.690 |
| 3   | 14  | 0.292 | 0.048 | 0.295 | 0.099 | 0.294 | 0.575 | 0.295 | 5.070 |
| 3   | 15  | 0.968 | 0.145 | 0.949 | 0.291 | 0.955 | 1.716 | 0.957 | 15.122 |
| 3   | 16  | 3.012 | 0.439 | 3.035 | 0.881 | 3.100 | 5.249 | 3.111 | 46.695 |
| 4   | 10  | 0.016 | 0.007 | 0.016 | 0.013 | 0.016 | 0.089 | 0.016 | 0.747 |
| 4   | 11  | 0.064 | 0.025 | 0.064 | 0.052 | 0.065 | 0.344 | 0.066 | 2.997 |
| 4   | 12  | 0.261 | 0.109 | 0.263 | 0.208 | 0.263 | 1.325 | 0.263 | 11.637 |
| 4   | 13  | 1.444 | 0.422 | 1.444 | 0.857 | 1.445 | 5.359 | 1.446 | 46.976 |
| 5   | 8   | 0.130 | 0.013 | 0.140 | 0.133 | 0.140 | 1.225 | 0.150 | 12.228 |
| 5   | 9   | 0.063 | 0.068 | 0.063 | 0.614 | 0.065 | 6.135 | 0.067 | 60.834 |
| 5   | 10  | 0.335 | 0.309 | 0.335 | 3.062 | 0.337 | 30.318 | 0.343 | 295.691 |
| 5   | 11  | 1.847 | 1.517 | 1.842 | 15.197 | 1.841 | 151.716 | 1.843 | 1514.924 |
| 7   | 6   | 0.004 | 0.002 | 0.004 | 0.020 | 0.004 | 0.202 | 0.008 | 2.049 |
| 7   | 7   | 0.027 | 0.013 | 0.022 | 0.166 | 0.021 | 1.469 | 0.026 | 14.554 |
| 7   | 8   | 0.170 | 0.107 | 0.174 | 1.037 | 0.172 | 10.084 | 0.181 | 101.280 |
| 7   | 9   | 1.351 | 0.743 | 1.363 | 7.105 | 1.379 | 70.795 | 1.397 | 705.218 |
Table 2 Experimental results using Magma V2.25-2 by online Magma Calculator

| q | k | n= 30 | n= 300 | n= 3000 | n= 30000 |
|---|---|-------|--------|---------|----------|
| 2 | 23 | 0.000 | 0.130  | 1.140   | 11.340   |
| 2 | 24 | 0.000 | 0.260  | 2.300   | 22.680   |
| 2 | 25 | 0.000 | 0.520  | 4.560   | 45.380   |
| 2 | 26 | 0.000 | 1.030  | 9.150   | 90.680   |
| 3 | 13 | 0.010 | 0.040  | 0.180   | 1.730    |
| 3 | 14 | 0.010 | 0.080  | 0.540   | 5.180    |
| 3 | 15 | 0.060 | 0.240  | 1.580   | 15.530   |
| 3 | 16 | 0.020 | 0.630  | 4.810   | 46.580   |
| 4 | 10 | 0.010 | 0.040  | 0.090   | 0.730    |
| 4 | 11 | 0.010 | 0.070  | 0.320   | 2.940    |
| 4 | 12 | 0.060 | 0.180  | 1.250   | 11.760   |
| 4 | 13 | 0.230 | 0.680  | 5.000   | 47.030   |
| 5 | 8  | 0.00  | 0.070  | 0.040   | 0.330    |
| 5 | 9  | 0.00  | 0.090  | 0.170   | 1.670    |
| 5 | 10 | 0.03  | 0.170  | 0.860   | 8.350    |
| 5 | 11 | 0.160 | 0.600  | 4.270   | 41.770   |
| 7 | 6  | 0.00  | 0.00   | 0.010   | 0.080    |
| 7 | 7  | 0.00  | 0.010  | 0.060   | 0.530    |
| 7 | 8  | 0.010 | 0.140  | 0.370   | 3.600    |
| 7 | 9  | 0.100 | 0.410  | 2.630   | 25.300   |

Remark 1 We compare our algorithm with Algorithm 9.8 (Walsh transform over a prime finite field $F_p$) in [22]. According to Joux, the complexity of his algorithm when $p$ varies is $O(kp^{k+1})$. We implement the presented approach, based on Algorithms 1–3, in a C/C++ program. To compare the efficiency, we use C implementation of an algorithm, presented in [9], with
the same efficiency as the Gray code algorithms. As a development environment for both algorithms we use MS VISUAL STUDIO 2012. All examples are executed on INTEL CORE i7-3770K 3.50 GHZ PROCESSOR in Active solution configuration — Release, and Active solution platform — X64.

Input data are randomly generated linear codes with lengths 30, 300, 3000, 30000 and different dimensions over finite fields with 2, 3, 4, 5, and 7 elements. All the results with the obtained execution times are given in seconds (Table 1). Any column consists of two subcolumns. The first subcolumn (named 'NEW') contains the results obtained by the new algorithm (described in this paper), and the second one gives the execution time for the same code but using the algorithm from [9], implemented in the package Q-EXTENSION. The runtime shown in Table 1 is the full execution time to compute the weight distribution starting with a generator matrix of a code with the given parameters.

In Table 2 we present results for the same parameters as in Table 1 but obtained using Magma V2.25-2 by online Magma Calculator run in a virtual machine on an Intel Xeon Processor E3-1220, 3.10 GHz.

The results given in the tables show that the presented approach is faster for codes with large length. The execution time for computing the characteristic vector is negligible.

Acknowledgements

This research was supported by Grants DN 02/2/13.12.2016 and KP-06-N32/2-2019 of the Bulgarian National Science Fund.

We thank Geoff Bailey, Computational Algebra Group, University of Sydney, for the provided information about the processor used by Magma Calculator. We are greatly indebted to the unknown referees for their useful suggestions.

References

1. Baart, R., Boothby, T., Cramwinckel, J., Fields, J., Joyner, D., Miller, R., Minkes, E., Roijackers, E., Ruscio, L., Tjhai, C.: GAP package GUAVA. https://www.gap-system.org/Packages/guava.html
2. Bellini, E., Sala, M.: A deterministic algorithm for the distance and weight distribution of binary nonlinear codes. Int. J. Inform. Coding Theory (IICOT) 5, 18–35 (2018)
3. Berlekamp, E.R., McEliece, R.J., van Tilborg, H.C.: On the inherent intractability of certain coding problems. IEEE Trans. Inform. Theory 24, 384–386 (1978)
4. Bikov, D., Bouyukliev, I.: Parallel fast walsh transform algorithm and its implementation with CUDA on GPUs. Cybern. Inform. Technol. 18, 21–43 (2018)
5. Borges-Quintana, M., Borges-Trenard, M.A., Fitzpatrick, P., Martínez-Moro, E.: Groebner bases and combinatorics for binary codes. Appl. Algebra Eng. Comm. Comput. 19, 393–411 (2008)
6. Borges-Quintana, M., Borges-Trenard, M.A., Márquez-Corbella, I., Martínez-Moro, E.: Computing coset leaders and leader codewords of binary codes. J. Algebra Appl. 14(8), 1550128 (2015)
7. Bosma, W., Cannon, J., Playoust, C.: The Magma algebra system i: The user language. J. Symbolic Comput. 24, 235–265 (1997). https://doi.org/10.1006/jsco.1996.0125
8. Bouyukliev, I.: What is Q-EXTENSION? Serdica. J. Comput. 1, 115–130 (2007)
9. Bouyukliev, I., Bakoev, V.: A method for efficiently computing the number of codewords of fixed weights in linear codes. Discret. Appl. Math. 156, 2986–3004 (2008)
10. Carlet, C.: Boolean functions for cryptography and error correcting codes. In: Crama, Y., Hammer, P.L. (eds.) Boolean Models and Methods in Mathematics, Computer Science, and Engineering, in: Encyclopedia Math. Appl., vol. 134. Cambridge University Press (2010)
11. Carlet, C., Charnin, P., Zinoviev, V.: Codes, bent functions and permutations suitable for DES-like cryptosystems. Des. Codes Crypt. 15(2), 125–156 (1998)
12. Doduneakov, S., Simonis, J.: Codes and projective multisets. Electron. J. Comb. 5(1), 37 (1998)
13. Dodunekova, R., Dodunekov, S.M.: Sufficient conditions for good and proper error-detecting codes. IEEE Trans. Inform. Theory 43(6), 2023–2026 (1997)
14. Edel, Y., Pott, A.: On the equivalence of nonlinear functions. In: Enhancing Cryptographic Primitives with Techniques from Error Correcting Codes, NATO Science for Peace and Security Series - D: Information and Communication Security, vol. 23, pp. 87–103. IOS Press (2009)
15. Elliott, D.F., Rao, K.R.: Fast Transforms: Algorithms, Analises, Applications. Academic Press, Orlando (1982)
16. Giorgetti, M., Sala, M.: A commutative algebra approach to linear codes. J. Algebra 321(8), 2259–2286 (2009)
17. Good, I.J.: The interaction algorithm and practical fourier analysis. J. Royal Stat. Soc. 20(2), 361–372 (1958)
18. Gulliver, T., Bhargava, V., Stein, J.: Q-ary Gray codes and weight distributions. Appl. Math Comput. 103, 97–109 (1999)
19. Han, S., Seo, H.S., Ju, S.: Efficient calculation of the weight distributions for linear codes over large finite fields. Contemporary Engineering Sciences 9, 609–617 (2016)
20. Huffman, W.C., Pless, V.: Fundamentals of error-correcting codes. Cambridge Univ. Press (2003)
21. Jan, B., Montrucchio, B., Ragusa, C., Khan, F.G., Khan, O.: Parallel butterfly sorting algorithm on GPU. In: Proceedings of Artificial Intelligence and Applications, pp. 795–026, Innsbruck, Austria (2013)
22. Joux, A.: Algorithmic Cryptanalysis. Chapman & Hall/CRC, Boca Raton (2009)
23. Jurrius, R., Pellikaan, R.: Codes, arrangements and matroids. In: Algebraic geometry modeling in information theory, in: Series on Coding Theory and Cryptology, vol. 8. World Scientific Publishing (2013)
24. Karpovsky, M.G.: On the weight distribution of binary linear codes. IEEE Trans. Inform. Theory 25(1), 105–109 (1979)
25. Katsman, G.L., Tsfasman, M.A.: Spectra of algebraic-geometric codes. Probl. Peredachi Inf. 23(4), 19–34 (1987)
26. MacWilliams, F., Mallows, C., Sloane, N.: Generalizations of Gleason’s theorem on weight enumerators of self-dual codes. IEEE Trans. Inform. Theory 18(6), 794–805 (1972)
27. MacWilliams, F.J., Sloane, N.J.A.: The Theory of Error-Correcting Codes. Elsevier, North-Holland (1977)
28. Márquez-Corbella, I., Martínez-Moro, E., Suárez-Canedo, E.: On the ideal associated to a linear code. Advances in Mathematics of Communications 10(2), 229–254 (2016)
29. Parhami, B.: Introduction to Parallel Processing: Algorithms and Architectures, Series in Computer Science. Springer, Berlin (2002)
30. Sala, M.: Gröbner basis techniques to compute weight distributions of shortened cyclic codes. J. Algebra Appl. 6, 403–414 (2007)
31. Sendrier, N.: Finding the permutation between equivalent linear codes: The support splitting algorithm. IEEE Trans. Inform. Theory 46, 1193–1203 (2000)
32. Stichtenoth, H.: Subfield subcodes and trace codes. In: Algebraic function fields and codes, graduate texts in mathematics, vol. 254. Springer, Berlin (2009)
33. Vardy, A.: The intractability of computing the minimum distance of a code. IEEE Trans. Inform. Theory 43(6), 1757–1766 (1997)

Publisher’s note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.