




The Program GENERATION in the Software Package QEXTNEWEDITION

Iliya Bouyukliev (✉) 

Institute of Mathematics and Informatics, Bulgarian Academy of Sciences,
P.O. Box 323, Veliko Tarnovo, Bulgaria
`iliyab@math.bas.bg`

Abstract. This paper is devoted to the program GENERATION which is a self-containing console application for classification of linear codes. It can be used for codes over fields with $q < 8$ elements and with wide-range parameters. The base of the implemented algorithm is the concept of canonical augmentation.

Keywords: Linear code · Classification · Software

1 Introduction

The classification problem of linear codes is important and difficult. Computer algorithms have been used to find the best linear codes for given length and dimension. There are many computational results for classification of linear codes over finite fields (see for example [3, 12, 13]), but there is not much related software available (for example MAGMA [4], GUAVA [1], ORBITER [2], Q-EXTENSION [5]). Our paper is a contribution to this research.

The system QEXTNEWEDITION is a software package consisting of several user interface programs for classification of linear codes over finite fields, along with the necessary supporting functions. Here we describe the program GENERATION for classification of linear codes over fields with $q < 8$ elements and with wide-range parameters. Despite its simple interface, it allows a lot of restrictions on the considered codes. It gives the possibility to classify not only codes with fixed parameters but also all codes with a given length n and dimensions from k_0 to k for given integers $1 \leq k_0 \leq k$. To use the program, a knowledge of a programming language is not needed. This program is supported by many different basic functions which implement complicated (in some cases) algorithms and has specific data organizations. The most important of these functions give the minimum distance, a list of codewords with weights smaller than a given integer (for large dimensions Brouwer-Zimmerman algorithm is applied), canonical form, automorphism group.

Supported by Grant DN 02/2/13.12.2016 of the Bulgarian National Science Fund.

This paper tries to give answers to the following questions:

- What type of algorithms are implemented and why they allow parallel implementation?
- What is the data organization?
- What is the difference with the previous version?
- How can the interface be used to enter parameters and what type of restrictions are possible and suitable for different cases?
- What can be expected from the program?

The remaining part of the paper is organized as follows. Section 2 contains the needed definitions. In Sects. 3 and 4 we describe the main algorithms in the program GENERATION and its basic functions and data organization, respectively. Sections 5 and 6 answer the questions how we can use GENERATION for code classification and what can be expected from the program. Finally, we draw a brief conclusion in Sect. 7.

2 Basic Definitions

In this section we present some definitions following [11]. Let \mathbb{F}_q^n denote the vector space of n -tuples over the q -element field \mathbb{F}_q . A q -ary linear code C of length n and dimension k , or an $[n, k]_q$ code, is a k -dimensional subspace of \mathbb{F}_q^n . A $k \times n$ matrix G whose rows form a basis of C is called a generator matrix of C . The number of nonzero coordinates of a vector $\mathbf{x} \in \mathbb{F}_q^n$ is called its Hamming weight $\text{wt}(\mathbf{x})$. The Hamming distance $d(\mathbf{x}, \mathbf{y})$ between two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{F}_q^n$ is defined by $d(\mathbf{x}, \mathbf{y}) = \text{wt}(\mathbf{x} - \mathbf{y})$. The minimum distance of a linear code C is

$$d(C) = \min\{d(\mathbf{x}, \mathbf{y}) \mid \mathbf{x}, \mathbf{y} \in C, \mathbf{x} \neq \mathbf{y}\} = \min\{\text{wt}(\mathbf{c}) \mid \mathbf{c} \in C, \mathbf{c} \neq \mathbf{0}\}.$$

A q -ary linear code of length n , dimension k and minimum distance d is said to be an $[n, k, d]_q$ code. Let A_i denote the number of codewords in C of weight i . Then the $n + 1$ -tuple (A_0, \dots, A_n) is called the *weight spectrum* of the code C .

An inner product (\mathbf{x}, \mathbf{y}) of vectors $\mathbf{x}, \mathbf{y} \in \mathbb{F}_q^n$ defines orthogonality: Two vectors are said to be orthogonal if their inner product is 0. The set of all vectors of \mathbb{F}_q^n orthogonal to all codewords in C is called the orthogonal code C^\perp to C :

$$C^\perp = \{\mathbf{x} \in \mathbb{F}_q^n \mid (\mathbf{x}, \mathbf{y}) = 0 \text{ for any } \mathbf{y} \in C\}.$$

It is well-known that the code C^\perp is a linear $[n, n - k]_q$ code. If $C \subseteq C^\perp$, the code C is called *self-orthogonal*. Self-orthogonal codes with $n = 2k$ are of particular interest, then $C = C^\perp$ and these codes are called *self-dual*.

The program GENERATION has an option for classification of self-orthogonal codes over fields with 2, 3 and 4 elements. In the binary and ternary cases, we consider *Euclidean inner product* defined by $u \cdot v = u_1v_1 + u_2v_2 + \dots + u_nv_n \in \mathbb{F}_q$ for $\mathbf{u} = (u_1, u_2, \dots, u_n)$, and $\mathbf{v} = (v_1, v_2, \dots, v_n)$. For $q = 4$ the considered inner product is the Hermitian inner product defined by $u \cdot v = u_1v_1^2 + u_2v_2^2 + \dots + u_nv_n^2 \in \mathbb{F}_4$ where $\mathbf{u} = (u_1, u_2, \dots, u_n)$, $\mathbf{v} = (v_1, v_2, \dots, v_n) \in \mathbb{F}_4^n$.

Two linear q -ary codes C_1 and C_2 are said to be *equivalent* if the codewords of C_2 can be obtained from the codewords of C_1 via a sequence of transformations of the following types:

1. permutation of coordinates;
2. multiplication of the elements in a given coordinate by a nonzero element of \mathbb{F}_q ;
3. application of a field automorphism to the elements in all coordinates simultaneously.

This equivalence may not preserve self-orthogonality over fields with $q \geq 5$ elements, for that reason we exclude the classification of self-orthogonal codes over fields with 5 and 7 elements.

An *automorphism* of a linear code C is a sequence of such transformations that maps each codeword of C onto a codeword of C . The automorphisms of a code C form a group, called the automorphism group of the code and denoted by $\text{Aut}(C)$.

Practically, we will identify a linear code with its generator matrix. We consider the code classification problem as follows. Given a set of parameters q, n, k, d find generator matrices of all inequivalent $[n, k, d]_q$ codes. In geometrical aspect, we can define an $[n, k, d]_q$ code C as a multiset of n points in $PG(k - 1, q)$ such that (a) each hyperplane of $PG(k - 1, q)$ meets C in at most $n - d$ points and (b) there is a hyperplane meeting C in exactly $n - d$ points. This definition is equivalent to the one given in [9].

Codes which are equivalent belong to the same equivalence class. Every code can serve as a representative for its equivalence class. We use the concept for a canonical representative, selected on the base of some specific conditions.

Let G be a group that acts on a set Ω . This action defines an equivalence relation in Ω as two elements $X, Y \in \Omega$ are equivalent, $X \cong Y$, if they belong to the same orbit. A canonical representative map for this action is a function $\rho : \Omega \rightarrow \Omega$ 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 $\rho(X) = \rho(Y)$. We take Ω to be the set of all linear $[n, k]_q$ codes. For a code $C \in \Omega$, the code $\rho(C)$ is the canonical form of C with respect to ρ . Analogously, C is in canonical form if $\rho(C) = C$. The code $\rho(C)$ is the canonical representative of its equivalence class with respect to ρ . Let $\gamma_C : C \rightarrow \rho(C)$ maps the code C to its canonical form, or $\gamma_C(C) = \rho(C)$. According to the definition given above, γ_C induces a permutation of the coordinates which we denote by π_C . The permutation π_C defines an ordering of the coordinates and the orbits of C with respect to the action of $\text{Aut}(C)$.

To find the canonical form and the automorphism group of C , we need a sufficiently large set $M(C)$ of codewords of the code C (we will call it *sufficient set*) with the following properties:

- $M(C)$ generates the code C ;
- $M(C)$ is stable with respect to $\text{Aut}(C)$;
- if $C' \cong C''$ and $\psi(C') = C''$ then $\psi(M(C')) \equiv M(C'')$, $\psi \in G$.

This set is not uniquely determined. Usually, we can accept as a sufficient set the set of all codewords with minimum weight. If the rank of this set is smaller than the dimension of the code, a larger set of codewords is used.

3 Main Algorithms in the Program GENERATION

In the program GENERATION any linear code is represented by its generator matrix. The program has two main parts. The first one implements a construction method for generator matrices. This method is based on row by row backtracking with $k \times k$ identity matrix as a fixed part. In the m -th step the considered matrices have the following form

$$G = (I_k \ A') = \left(\begin{array}{c|c|c} I_m & O & A_m \\ \hline O & I_{k-m} & X \end{array} \right)$$

where the columns of the matrix A_m are lexicographically ordered, and X is the unknown part of G . In that case any vector v_m of length $n - k$ which fits for the m -th row of A_m strictly depends on one of the vectors put on the previous rows. Consider, for example, the binary case. If $m = 1$ there are only two options for columns of matrix A_1 , namely 0 and 1, four options for $m = 2$, namely $(00)^T$, $(01)^T$, $(10)^T$ and $(11)^T$, and so on. Let the matrix A_{m-2} already be constructed. We define a set T_{m-1} of all suitable vectors for the last row in the next matrix A_{m-1} . Taking $v_{m-1} \in T_{m-1}$, we obtain the matrix A_{m-1} . The vector v_{m-1} defines an ordered partition $\Pi_{v_{m-1}}$ of the set $S = \{k + 1, k + 2, \dots, n\}$. The possibilities for the next m -th row correspond to the refinement partitions of $\Pi_{v_{m-1}}$ induced by the vectors in T_{m-1} .

Example 1. Let us try to construct all $[11, 3, 6]$ binary even codes taking their generator matrices in a systematic form: $G = (I_3|X)$. Any row in the unknown matrix X must have 5 or 7 nonzero coordinates. For the set T_1 we have $T_1 = \{(00011111), (01111111)\}$. The current possible matrices G are

$$\left(\begin{array}{c|c} 100 & 00011111 \\ \hline 010 & X \\ 001 & \end{array} \right), \text{ and } \left(\begin{array}{c|c} 100 & 01111111 \\ \hline 010 & X \\ 001 & \end{array} \right).$$

Take $v_1 = (00011111)$. The vector v_1 induces the partition $\Pi_{v_1} = \{\{1, 2, 3\}, \{4, 5, 6, 7, 8\}\}$ and the set $T_2 = \{t_1 = (01100111), t_2 = (11100011), t_3 = (11101111)\}$. Let fix $v_2 = t_1$,

$$A_2 = \left(\begin{array}{c|c} 0 \ 00 \ 11 \ 111 \\ \hline 0 \ 11 \ 00 \ 111 \end{array} \right) \text{ and } G = \left(\begin{array}{c|c} 100 & 00011111 \\ \hline 010 & 01100111 \\ 001 & X \end{array} \right).$$

Then $\Pi_{v_1, v_2} = \{\{1\}, \{2, 3\}, \{4, 5\}, \{6, 7, 8\}\}$. Now we have to find the solutions for the last row. The first vector $t_1 \in T_2$ and Π_{v_1} give the information that we have to put two 1's in the first three coordinate positions, and three 1's in the last 5 positions. We obtain the following possibilities (taking in mind also the lexicographical ordering and the partition Π_{v_1, v_2}) (01100111) , (01101011) , (01111001) , (10100111) , (10101011) , (10111001) . Only the last two vectors give $[11, 3, 6]$ even codes (the other four codes have minimum distance ≤ 4). In the

same way we consider the second and the third vectors in T_2 . By exhausting all possibilities in this way, we get all inequivalent codes we are looking for.

As a result, we obtain only one (up to equivalence) binary even $[11, 3, 6]$ code. This example explains the construction part given above.

There are several advantages of this approach:

- the number of equivalent candidates in the search tree becomes smaller,
- the construction of generator matrices is very effective,
- it allows us to consider codes with relatively large length - more than a hundred in the binary case.

Moreover, this construction is also appropriate to the other part of the program that determines inequivalent objects. In fact, this has been a key idea in the package Q-EXTENSION (more detail for this approach and the implementation can be found in [8]). To the rest of construction part we add functions for minimum and dual distances, orthogonality check and restrictions on weights.

The second part of the program is related to the identification of non-equivalent objects in the whole generation process. The general method which we apply is known as *canonical augmentation* [13, 14]. Description for this specific case is given in [6]. The basic idea is to accept only non-equivalent objects without an equivalence test (in some cases with a small number of tests) at every step of the generation process. Instead of an equivalence test, a canonical form of the objects and a canonical ordering of orbits are used. So for every vector v_m in the construction that fits as a m -th row (we call these vectors *possible solutions*), the algorithm decides acceptance (possible solution becomes *real*) or rejection. In this model, the different branches of the search tree are independent and therefore it is easy for parallel implementation.

The main algorithms are developed by the basic functions of the package. Some of them are presented in the next section.

4 Basic Functions and Data Organization

To present the basic functions used in the program GENERATION, we have to give some information for the whole package QEXTNEWEDITION. It contains several hierarchically ordered modules with functions written in C/C++. Each module depends on the previous one and makes it possible to realize the functions of the next one. The interface programs (like GENERATION) stays on the top of this hierarchy.

The first module deals with the safe allocation of dynamic memory for the whole package. The main structures of the package are matrices (two dimensional arrays) of different types. These structures are used to store generator matrices, check matrices, sets of generator matrices with non-intersecting information sets, sets of all or some of the codewords of considered linear codes, sufficient sets, their corresponding binary matrices, the canonical forms and so on. The concept of the package is to investigate linear codes one by one (in consecutive execution).

Therefore, it is convenient to use some global variables. The size of the dynamic variables for different types of data related to linear codes changes when the main function considers the next object. In the beginning, the first module allocates memory for the first object. If this memory is not enough for some of the following objects, it allocates more memory by default.

The second module consists of functions related to the rank of a system of codewords, information set, orthogonal code, construction of different generator matrices (with non-intersecting information sets), etc.

The following module is related to functions for generating some or all codewords. They give minimum distance, weight spectrum, sufficient set of codewords, coset leaders, etc. We use two general approaches for calculating the weight characteristics of linear codes. One of them is exhaustive search (for small dimensions only) and the other is based on Brouwer-Zimmerman algorithm. Many of the functions check if the minimum distance, weight spectrum and other distance parameters are suitable.

A very important part of the package is the module for canonical form and automorphism group. The central object here is the $(0,1)$ -matrix or bipartite graph. The main function in this module obtains canonical form, generators of the automorphism group and orbits of rows and columns (and their ordering) of a given binary matrix. For a linear code C , we use sufficient set $M(C)$ of codewords and invertible mapping of this set to a binary matrix $T(C)$ (see [7]). If two codes C_1 and C_2 are equivalent their corresponding binary matrices $T(C_1)$ and $T(C_2)$ are isomorphic. Moreover, the automorphism groups of C and $T(C)$ are isomorphic, too.

5 How Can We Use the Program for Code Classification?

In this section, we show how the program GENERATION can be used with examples. Let us consider the binary codes with parameters $[24, 7, 10]$. It is known that there are 6 inequivalent codes with these parameters [12].

After starting, GENERATION gives us the following by default:

```

Generating Linear Codes (Generation v1.1 QextNewEdition first module)
      Generate [24,12,8;2] Linear codes
      With weights:
wt1= 8, wt2= 12, wt3= 16, wt4= 20, wt5= 24,
      Proportional columns:
d2->800, d3->800, d4->800, d5->800, d6->800, d7->800, d8->800,
d9->800, d10->800, d11->800, d12->800,
      1. Start
      2. Change input parameters
      3. Restriction on weights
      4. Restriction on proportional coordinates
      5. Dual distance 1
      6. Brute generation
      7. About QextNewEdition
      8. Exit
Choose:

```

To obtain the generator matrices of all 6 inequivalent binary $[24,7,10]$ codes in the file with name `24_7_10.2` we just have to choose point 2, enter the parameters and start the calculations choosing 1. The generator matrices of all inequivalent codes obtained in the generation process (157 in our case) will be written in a file with name `24_7_10.2h`. They correspond to all real solutions for $k \neq 1, 7$. The table of optimal codes [10] indicates that binary linear codes with parameters $[22, 6, 10]$ do not exist. Therefore binary $[24, 7, 10]$ codes do not have two proportional coordinates and can be obtained from $[23,6,10]$ codes. That is why we can use restrictions for proportional coordinates (point 4) as follows: up to 4 proportional coordinates in dimension five, 2 in dimension six and no (enter 1) in dimension 7. In this case, the calculation time is 25% less.

If we are interested only in codes with dual distance 4, we can use point 5. The program looks for the codes with dual distance 2 in dimension five and 3 in dimension six. The number of inequivalent codes in the file `24_7_10.2h` becomes smaller - 146.

The program has two options for restrictions on possible weights of the codes under investigation. With point 2 we can set an integer w which divides all the weights. After that with point 3 we can choose only some of the weights between d and n , divisible by w . The restriction for self orthogonality works only for codes over fields with 2, 3 and 4 elements.

In the general case, when the program have to generate $[n, k, d]$ codes, the codes with parameters $[n - t, k - i, d]$ are in the search tree, where $1 \leq i \leq k - 1$, $i \leq t \leq n - k + i$.

For optimal search of all codes with fixed n and d , and dimensions from k_{min} to k_{max} , we use point 6. In that case the results will be written in files with extensions "2b" and "2bh". For example, the search trees for constructing $[25, 6, 10]$, $[25, 5, 10]$ and $[25, 4, 10]$ self-orthogonal binary codes have 226, 289 and 99 nodes, respectively (614 summary). If we look for the self-orthogonal codes with the same parameters simultaneously by point 6, the nodes of the corresponding search tree are only 430.

6 Computational Results

In this section we present some examples. To obtain the results, we use one thread of INTEL XEON E5-2620 V4 processor. For natural reasons, the calculational time in the case of relatively small parameters depends on the size of the search tree. For given n , k and q the search tree strictly depends on the restrictions for minimum distance, self-orthogonality, possible weights, dual distance and proportional columns.

In the case of codes with large length, the number of objects that need to be checked for acceptability increases exponentially. That is why, even with a small search tree, the computational time grows.

The following table contains classification results for linear codes with different parameters and restrictions. The first and second columns show the parameters and the used restrictions, respectively. Column 3 contains the execution times in seconds, and the number of equivalent codes in each case is given in the fourth column.

Parameters	Restrictions	Time	#
$[109, 5, 56]_2$	Weights: 56 64 72	1145.38 s	1
$[34, 12, 12]_2$	Weights:12 16 20 24 28 32	19404.67 s	11
$[18, 6, 4]_2$	Even	2337.91 s	434906
$[19, 7, 9]_3$		72.01 s	61
$[22, 6, 12]_3$		114.52 s	701
$[24, 12, 9]_3$	Self-orthogonal	148.73 s	2
$[28, 8, 15]_3$		47.17 s	1
$[24, 5, 16]_4$	Weights: 16 18 20 22 24	472.49 s	1

7 Conclusion

In this paper, we present the first interface program GENERATION of the software package QEXTNEWEDITION. There are freely available versions for WINDOWS and LINUX on the webpage

<http://www.moi.math.bas.bg/moiuser/~data/Software/QextNewEdition>

The package contains two more programs, namely LENGTHEXTENSION and DIMEXTENSION which will be available on the same webpage.

The package QEXTNEWEDITION is a successor of Q-EXTENSION [5]. The aim of both systems is classification of linear codes with different properties and restrictions. They share some ideas in the development of algorithms and have similar interface. The package Q-EXTENSION is written in PASCAL (DELPHI) with static variables depending on the size of the field. QEXTNEWEDITION is a new software system, written in C/C++, designed to be widely portable and suitable for parallelization. All basic functions are rewritten, looking for optimal implementation. The main concept and used methods for classification are different. The classification here is based on canonical augmentation as opposed to Q-EXTENSION where the used method is isomorph-free generation via recorded objects [13].

There are many differences between QEXTNEWEDITION and Q-EXTENSION. We list some new points:

- Programming language is C/C++ which make program portable and proper for MPI parallelization.
- Dynamically allocated variables are used. This means that the size of the input data depends only on the hardware and the range of the program can easily be extended to larger fields.

- The implementation of the generating part presented in Sect. 3 is different. In the beginning, it was by nested loops and now it is based on specific integer partitions [8].
- The algorithm for canonical form is optimized by additional invariants for the partitioning process.
- The representation of the sufficient set as a binary matrix now is much more flexible [7].

With these features, the program GENERATION is a powerful tool for classifying linear codes.

Acknowledgements. We are greatly indebted to the unknown referees for their useful suggestions.

References

1. Baart, R., et al.: GAP package GUAVA. <https://www.gap-system.org/Packages/guava.html>
2. Betten, A.: Classifying discrete objects with orbiter. *ACM Commun. Comput. Algebra* **47**(3/4), 183–186 (2014)
3. Betten, A., Braun, M., Friepertinger, H., Kerber, A., Kohnert, A., Wassermann, A.: *Error-Correcting Linear Codes - Classification by Isometry and Applications*. Springer, Heidelberg (2006). <https://doi.org/10.1007/3-540-31703-1>
4. Bosma, W., Cannon, J., Playoust, C.: The Magma algebra system I: The user language. *J. Symbolic Comput.* **24**, 235–265 (1997)
5. Bouyukliev, I.: What is Q-EXTENSION? *Serdica J. Comput.* **1**, 115–130 (2007)
6. Bouyukliev, I., Bouyuklieva, S., Kurz, S.: Computer classification of linear codes. [arXiv:2002.07826](https://arxiv.org/abs/2002.07826) [cs.IT] (2020)
7. Bouyukliev, I., Dzhumalieva-Stoeva, M.: Representing equivalence problems for combinatorial objects. *Serdica J. Comput.* **8**(4), 327–354 (2014)
8. Bouyukliev, I., Hristova, M.: About an approach for constructing combinatorial objects. *Cybernetics Inf. Technol.* **18**, 44–53 (2018)
9. Dodunekov, S., Simonis, J.: Codes and projective multisets. *Electron. J. Comb.* **5**, R37 (1998)
10. Grassl, M.: Bounds on the minimum distance of linear codes and quantum codes. <http://www.codetables.de>. Accessed 10 March 2020
11. Huffman, W.C., Pless, V.: *Fundamentals of Error-Correcting Codes*. Cambridge University Press, Cambridge (2003)
12. Jaffe, D.B.: Optimal binary linear codes of length ≤ 30 . *Discret. Math.* **223**, 135–155 (2000)
13. Kaski, P., Östergård, P.R.: *Classification Algorithms for Codes and Designs*. Springer, Heidelberg (2006). <https://doi.org/10.1007/3-540-28991-7>
14. McKay, B.: Isomorph-free exhaustive generation. *J. Algorithms* **26**, 306–324 (1998)