Serdica J. Computing **1** (2007), 115–130

Serdica
Journal of Computing

Bulgarian Academy of Sciences
Institute of Mathematics and Informatics

WHAT IS Q-EXTENSION?

Iliya G. Bouyukliev

Abstract. In this paper we present a developed software in the area of Coding Theory. Using it, codes with given properties can be classified. A part of this software can be used also for investigations (isomorphisms, automorphism groups) of other discrete structures—combinatorial designs, Hadamard matrices, bipartite graphs etc.

1. Introduction. We presented three programs ($q\_ext\#\_l.exe$, $q\_ext\#\_d.exe$ for $\# = 2, 3, 4$ and $q\_ext\_tools.exe$) in the area of Coding Theory which can be started and can work independently. Using them, codes with given properties can be classified. One of these programs ($q\_ext\_tools.exe$) can be used also for investigations (isomorphisms, automorphism groups) of other discrete structures—combinatorial designs, Hadamard matrices, bipartite graphs etc. To use these programs, knowledge of a programming language is not needed. This program is available in `http://www.moi.math.bas.bg/~iliya/Q\_ext.htm`.
Other software in this direction can be found in
– `http://cadigweb.ew.usna.edu/~wdj/gap/GUAVA/`
– `http://www.math.unl.edu/~djaffe/`
– `http://magma.maths.usyd.edu.au/magma/`
– `http://www.cs.sunysb.edu/algorith/implement/nauty/implement.shtml`.

ACM Computing Classification System (1998): D.0.
Key words: Classification of codes, Software, Algorithm, Isomorphism Test.

2. What can you do with Q-EXTENSION**?** You can classify all linear codes with given parameters over finite fields with 2, 3 and 4 elements. You can construct some or all codes with a given length and/or dual distance and/or orthogonality conditions.

If you have got generator matrices of codes with length n, dimension k, and minimum distance $\geq d$, you can find:

– all $[n+n_1, k, d+d']$ codes for $n_1 = 1, 2, \ldots$ (*q_ext_l.exe*);

– all $[n+n_1, k+1, d']$ codes for which the $[n, k, d]$ codes are residual (*q_ext_l.exe*);

– all $[n+n_1, k+1, d]$ codes for which the $[n, k, d]$ codes are subcodes (*q_ext_d.exe*).

With the additional program *q_ext_tools.exe* you can find:

– the nonequivalent between many linear codes over small finite fields;

– the orders of their automorphism groups and the orbits;

– the nonequivalent between many binary matrices, which cover the similar problems for combinatorial designs, Hadamard matrices, bipartite graphs;

– weight characteristics of linear codes in a large set and some subsets connected with these characteristics;

– systematic form, parity-check matrix and other trivial matters.

For examples look at Section 8.

3. Basic definitions. We consider two types of objects: linear codes and binary matrices. In this section we present some definitions following [8].

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 $\mathbb{F}_q^n$. 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 from 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.

A $k \times n$ matrix G_C whose rows form a basis of C is called a generator matrix of C. A generator matrix of the code $C^\perp$, orthogonal to C, is a parity check matrix for C, denoted by H_C.

The number of nonzero coordinates of a vector $\mathbf{x} \in \mathbb{F}_q^n$ is called its Hamming weight $\mathrm{wt}(\mathbf{x})$. The Hamming distance $d(\mathbf{x}, \mathbf{y})$ between two vectors

$\mathbf{x}, \mathbf{y} \in \mathbb{F}_q^n$ is defined by

$$d(\mathbf{x}, \mathbf{y}) = \mathrm{wt}(\mathbf{x} - \mathbf{y}).$$

The minimum distance of a linear code C is

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

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 $(A_0, \ldots, A_n)$ is called the *weight spectrum* of the code C.

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 *Euclidean inner product* of two vectors $\mathbf{u} = (u_1, u_2, \ldots, u_n)$ and $\mathbf{v} = (v_1, v_2, \ldots, v_n)$ from $\mathbb{F}_q^n$ is defined by

$$(\mathbf{u}, \mathbf{v})_E = u_1 v_1 + u_2 v_2 + \cdots + u_n v_n.$$

For codes over $\mathbb{F}_q$ where q is an even power of an arbitrary prime p, one can consider another type of inner product, the Hermitian inner product. The *Hermitian inner product* of two vectors $\mathbf{u} = (u_1, u_2, \ldots, u_n)$ and $\mathbf{v} = (v_1, v_2, \ldots, v_n)$ from $\mathbb{F}_q^n$ is defined by

$$(\mathbf{u}, \mathbf{v})_H = u_1 \bar{v}_1 + u_2 \bar{v}_2 + \cdots + u_n \bar{v}_n,$$

where $\bar{v}_i = v_i^{\sqrt{q}}$ for $v_i \in \mathbb{F}_q$. Consequently, for $q = 4$ the Hermitian inner product is defined by

$$(\mathbf{u}, \mathbf{v})_H = u_1 v_1^2 + u_2 v_2^2 + \cdots + u_n v_n^2.$$

In the ternary case we consider the Euclidean inner product and in the quaternary case (like in most other studies) the Hermitian inner product. Throughout the paper, these inner products are assumed in the discussion of self-dual and self-orthogonal codes.

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.

The field $\mathbb{F}_3$ does not have nontrivial automorphisms, and the only nontrivial automorphism of $\mathbb{F}_4$ is conjugation. 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 $\mathrm{Aut}(C)$.

Define the following basic problem in coding theory. For a given set of parameters n, k, d and q find generator matrices of all nonequivalent q-ary $[n, k, d]$ codes. Precise discussion in this topic can be found in [10]. This problem has two main subproblems. First of them is to construct all codes with such parameters and second one is the equivalence test. Practically, we reduce the code equivalence test to the problem for isomorphism of binary matrices.

Two binary matrices of the same size are equivalent if the rows of the second one can be obtained from the rows of the first one with a permutation of the columns. Any permutation of the columns of the matrix A which maps the rows of A into 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 S_n and we denote it by $Aut(A)$.

The equivalence test for binary matrices is connected with the graph isomorphism problem. There are several reasons to say that. First of all, any binary matrix can be considered as a bipartite graph. In the case of bipartite graph, the set of vertices is decomposed into two disjoint colored sets (columns and rows) such that no two graph vertices within the same set are adjacent. This is why solving the isomorphism problems for bipartite graphs and binary matrices is the same.

In other hand, any graph can be made bipartite by replacing each edge by two edges connected with a new vertex. And any two graphs are isomorphic if and only if the transformed bipartite graphs are.

4. How can you use these programs for code classification?

In this Section we will show how you can use these programs and especially one of them (*q_ext2_l.exe*) to extend a code up to length with one example. Let us consider the binary codes with parameters [47,7,22]. It is known that there exists a unique code C_{47} with these parameters. How to prove this by Q-Extension.

For these codes we know that they have codewords of weight 22. Without loss of generality, we can take the first row g_1 of the generator matrix G to be

such a codeword. Up to equivalence, $g_1 = (11\ldots1100\ldots00)$. The last 25 columns of G generate a linear code called a residual code $\mathrm{Res}(C, g_1)$ of C with respect to g_1. For its minimum distance we have the following Lemma [6].

Lemma 1. *Suppose C is an $[n,k,d]_q$ code and suppose $\mathbf{c} \in C$ has weight w, where $d > w(q-1)/q$. Then $Res(C,\mathbf{c})$ is an $[n-w, k-1, d']_q$ code with $d' \geq d - w + \lceil w/q \rceil$.*

We define a residual code in general in the following way:

Let G be a generator matrix of a linear $[n,k,d]_q$ code C. Then the *residual code* $\mathrm{Res}(C,\mathbf{c})$ of C with respect to a codeword $\mathbf{c}$ is the code generated by the restriction of G to the columns where $\mathbf{c}$ has a zero entry.

It is clear that the last 25 coordinates of G have to generate $[25, 6, \geq 11]$ code. We see in Table [5] that [25,6,12] codes do not exist. If we know all [25,6,11] codes given by their generator matrices, we know the first row and the last 25 columns of a generator matrix of any [47,7,22] code. The program *Q_ext_l* has to take care about the first 22 columns of G.

Similarly, we can consider a generator matrix of the [25,6,11] code partitioned in two parts such that the second one is a generator matrix of a [14,5,6] code according to Lemma 1. These codes are also optimal, since [14,5,7] codes do not exist.

To construct all codes with such relatively small parameters, we can use the property that every code is equivalent to a code with a generator matrix in the form $(A|I_k)$ where I_k is the identity matrix of order k. We call this form systematic. The last k columns generate the trivial [k,k,1] code. We can also fix the first row, as every linear code is equivalent to a code with a generator matrix in which the first row has weight d.

How to use the program to construct all [47,7,22] codes. After starting, *Q_ext_l* gives us the following possibilities:

```
       Q-Extension ver 0.1  {Length}

Linear [n,k,d] Codes over GF(2);  n<128  k<20
-------
Extension:
(1)  [3,3,1] to [6,3,2]
   1. Start
   2. Restrictions on weights
   3. Column multiplicity  restrictions
   4. Change inp --> outp
```

```
 2 5. Dual distance
 o 6. Form of the output matrices (c-convenient for extension,
                                   o-ordinary)
 n 7. Self-orthogonal
   8. Number of ones adding in first row (Now :1)
   9. Show input
   10. Show output
   11. Start with all even weights: 1  to 5
   12. Restrictions on even weights
 n 13. Self-complementary
   14. Help
   15. About Q-Extension
   16. Exit
Choose:_
```

The codes with parameters $[n, k = n, 1]$ are trivial and they are integrated in the program. As a default, the program works with the parameters $[3, 3, 1]$ for the input code and $[6, 3, 2]$ for the output codes. If we choose point 1 (type 1 and press enter) we start the program. It will find all nonequivalent [6,3,2] codes with generator matrices:

$$\begin{pmatrix} 100100 \\ 111010 \\ 010001 \end{pmatrix} \quad \begin{pmatrix} 100100 \\ 111010 \\ 100001 \end{pmatrix} \quad \begin{pmatrix} 100100 \\ 111010 \\ 011001 \end{pmatrix} \quad \begin{pmatrix} 100100 \\ 110010 \\ 001001 \end{pmatrix}$$

$$\begin{pmatrix} 100100 \\ 110010 \\ 101001 \end{pmatrix} \quad \begin{pmatrix} 100100 \\ 011010 \\ 100001 \end{pmatrix} \quad \begin{pmatrix} 100100 \\ 010010 \\ 001001 \end{pmatrix}$$

The program writes these matrices in a file with the name 6_3_2.2. This file could be used as input for another extension of the codes with parameters $[6, 3, 2]$. If the parameters of the input codes are not $[n, k \neq n, 1]$ the program takes them from a file with the name $n\_k\_d.q$.

We can ask for some restrictions on the code parameters. If you want to construct codes with given weights, you have to choose 2. After that you have to verify or exclude any of the weights. With 4 you can change the parameters of the input and output codes. For an input code you have to use the trivial code or already classified codes. The output code must have the same dimension as the input code, or the parameters of the input code have to be parameters of a residual of the output code. Point 5 is for the dual distance. We should mention

that if the code C has dual distance $d^{\perp}$ its residual code has the same or bigger dual distance (if its dimension is $k-1$). When you want to classify self-orthogonal codes, use 7, and for self-complementary codes use 13. If you want to find all $[n, k, \geq d]$ codes using input codes with the same dimension, use 8.

In our example, to classify all [14,5,6] codes we choose 4 (type 4 and press enter). The input code has parameters [5,5,1]. The program asks: **Input codes n,k,d:_** . Then type '5 5 1' and press enter. The output codes have parameters [14,5,6]. After entering the parameters of the input code the program asks in a similar way for the parameters of the output codes. While working, the program writes on the screen information for the tree of the back-track search. In the example it looks like that:

```
1- 1   0-s  1
1- 1   0-s  0  7
1- 1   0-s  0  6  9
1- 1   0-s  0  6  8  9
1- 1   5-s  0  6  8  8
1- 1   5-s  0  6  8  7
1- 1   5-s  0  6  8  6
1- 1   5-s  0  6  8  5
1- 1   5-s  0  6  8  4
1- 1   5-s  0  6  8  3
1- 1   5-s  0  6  8  2
1- 1   5-s  0  6  8  1
1- 1   5-s  0  6  8
1- 1   5-s  0  6  7  4
1- 1   5-s  0  6  7  3
1- 1   5-s  0  6  7  2
1- 1   5-s  0  6  7  1
1- 1   5-s  0  6  7
1- 1   5-s  0  6  6  3
1- 1   6-s  0  6  6  2
1- 1   6-s  0  6  6  1
1- 1   6-s  0  6  6
1- 1   6-s  0  6  5  7
1- 1   6-s  0  6  5  6
1- 1   6-s  0  6  5  5
......
```

In the first column we have the number of the codes for extension, in

the second which code is being extended at the moment, in the third how many nonequivalent codes are constructed until now. After that the number of nonequivalent codes in dimension 2, 3, etc. which we obtain in the extension, follows.

After a few seconds you can see the generator matrices of the constructed codes with parameters [14,5,6] in a separate window using 10. After closing that window, you can continue with the extension of the codes with parameters [14,5,6] to codes with parameters [22,6,11] using 4. Such an extension we will denote in the following way: $[14,5,6]-l->[22,6,11]$.

In the last step of the extension $[22,6,11]-l->[47,7,22]$ we construct the only up to equivalence [47,7,22] code. The generator matrix of this code has following form

$$\begin{pmatrix}
11111111111111111111110000000000000000000000000\\
11111111111000000000001111111111100000000000000\\
11111000000111111000001111100000011111000010000\\
10000100000111000110001100011100011111111101000\\
11000011100110100101101010010011011100110000100\\
10100010011100110011001011001110011010101000010\\
11010001010101011000101000111011000110101100001
\end{pmatrix}$$

Such kind of extension we call extension up to length.

Sometimes it is easier to use the codes with parameters $[n,k,d]$ to construct and classify codes with parameters $[n+1,k+1,d]$. We use the fact that any $[n+1,k+1,d]$ code contains $[n,k,d]$ and $[n+1,k,d]$ subcodes. If the code has dual distance 2, it contains $[n-1,k,d]$ subcode and also $[n-i,k,d]$ subcode for $i=1,2,\ldots,t$ where t is the maximum number of proportional columns in the code. We call this extension up to dimension. For it we use the program *q_ext_d*.

We denote this type of extension by $[45,6,22]-d->[47,7,22]$ (extension up to dimension). The results, obtained for the output codes in *q_ext_d*, can be used as input for *q_ext_l* and the opposite.

Now we will list some comments and options related to the programs for extension:

- First and second row of the main menu shows that the started program is about extension by length of binary codes with maximum length 128 and dimension 20.
- The number of the input codes and their parameters and the parameters of output codes are written in the 4-th row. After starting the program we

always have "(1) [3,3,1] to [6,3,2]". The parameters of input and output codes can be changed using point 4 from the main menu. The number of the input codes is 1 if the code parameters are [k,k,1] or it is the number of the matrices in the input file $n_{inp}\_k_{inp}\_d_{inp}.q$

- With points 2,3 and 7 additional restrictions on the codes which we try to classify can be done. Some of these restrictions are very important for the efficiency of the algorithms and the calculational time.

- In the case of binary codes with dual distance greater than 2 we know that there are no proportional (repeated) columns in a generator matrix. Moreover, any subcode with dimension $k-1$ has at most 2 proportional columns, etc. We can add this type of information using point 3.

- If the input and output codes have the same dimension we can use point 8 "to say" to program the number of ones which we add in the first row of the generator matrix.

- The program visualizes in a simple way what is done in the different points of the menu:
  – Default dual distance is written before '5' in point 5.
  – There are two types of output matrices: ordinary and convenient for extension (see [4]). That is why the program types 'o' or 'c' before point 6.
  – There are two options for self-orthogonally—y(yes) or n(no). The program types this before point 7.

5. About data organization. These programs use and create usual text files, in which generator matrices of the considered codes in a given format are written, and also some additional information for how the program has been started. The file's names are connected with the code parameters. For example generator matrices of the codes with parameters $[22, 6, 11]_2$ are in a file named "22_6_11.2". This file can be opened, explored and changed with any text editor. There is a row above every matrix with the code parameters – dimension, length, number of the elements of the field, name (identificator). This row begins with the character **'?'**. We have the opportunity to write some additional information in the file, like weight enumerators, orders of the automorphism groups, the parameters of the input codes etc. The dimension, the length, and the number of the elements of the fields are written after **'?'**. These data are obligatory. After

them in the same row a name-identificator can be written. When the program reads the special (for us) character **'?'** it expects a generator matrix from the next row. If **'?'** is missing the program consider the following matrix as a comment. This special row is followed by the rows of a generator matrix. Every vector is in a different row and its elements are not separated by any symbol. So we have the following type of data:

?3 6 2 $Name1$
100100
111010
010001

?4 7 2 $Name2$
1001001
1110101
0100010
0101100
...

If you use the programs $q\_ext\#\_l.exe$, $q\_ext\#\_d.exe$ for $\# = 2, 3, 4$, it is not necessary to know the data organization. Suppose that you want to find all inequivalent codes with parameters n_{out},k_{out} and d_{out} which contain some of the codes $C_1, C_2, \ldots, C_j$ with parameters n_{inp}, k_{inp} and d_{inp} as subcodes (or residual codes). In this case you have to put the generator matrices of the codes $C_1, C_2, \ldots, C_j$ in the given above form in a text file with name $n_{inp}\_k_{inp}\_d_{inp}.q$. For example, if you extend $[5, 5, 1] - l - > [14, 5, 6]$ the program will create a file with name 14_5_6.2 and will write in it the generator matrices of all $[14, 5, 6]_2$. If you want to extend only some of the codes with parameters $[14, 5, 6]_2$ the generator matrices of these codes have to be written in the file with the same name 14_5_6.2. Obviously, any input file for the program for extension has to consists only of matrices with the same parameters.

There are no restriction for the input file name for the program Q_ext_tools. This input file can contain matrices with different parameters. If you want to investigate binary matrices (not linear codes) the row with parameters has to contain '?', the number of rows, the number of the columns of the matrix, and '2' in the end (as a generator matrix of binary linear codes).

6. About the algorithms. In the considered programs we use many different algorithms. The most important of them are connected with two main

directions: constructing generator matrices of the output codes, and a test for equivalence in any step in the construction. The basic ideas are described in [4]. Our aim is to construct as small number of generator matrices as possible. We use such a form of the generator matrices in which there is a fixed part common for all codes with these parameters. For example, we can consider matrices in systematic form and the first row to be a codeword with minimum weight. So at least $n - k$ unknown columns remain. Our approach is a nontrivial back-track search close to the dynamic programming. To restrict the search tree we cut some parts using isomorphisms up to extension.

The isomorphism test in the current version is different from the previous versions of the program. It is based on the idea of canonical representation of an invariant set of codewords. For details see [2] and [9].

The main advantage of the canonical representation is that the equivalence (isomorphism) test is reduced to check of coincidence of the canonical representations of the structures. In the case of many nonequivalent codes the computational time for comparing is growing fast. A technic for surmounting this problem is worked out. We split the set of nonequivalent codes into a big amount of cells according to a proper invariant. To implement this algorithm, we use many other algorithms as: Counting the weight enumerator of a linear code, finding the set of codewords with a given weight, counting the rank of a matrix, etc. An estimate of the efficiency of some of these algorithms is presented in [1]. The algorithm in fact is an exhaustive search. The number of the solutions even in special cases grows exponentially. That's why we do not discuss the efficiency here.

The generating of an invariant set of codewords is connected with the computation of the weight spectrum or the codewords with a given weight. Unfortunately, computing the weight enumerator of a code becomes computationally intractable when its size grows, from [12] it is in fact NP-hard (see also [1]). Computational difficulty of the problem of code equivalence have been discussed by Petrank and Roth [11].

Algorithms for code equivalence and for computing the codewords with a given weight, implemented in the package are presented in [2] and [3].

7. About *q_ext_tools*. This is the third program in the package. It consists of different algorithms from the first two programs. To facilitate the interface and its use the procedures are separated in a few modules. The data organization (input and output) is universal for all procedures. The input data are taken from a text file, and the results are written in another text file. You

can point the names of the input and output files. The files with the results from the first two programs can be used as input files in this program, too. When you start the program, you see the following:

```
      Q-Extension TOOLS  ver 0.1

  1. Weight
  2. Utilities
  3. Isomorphism and automorphism group
  4. Covering radius
  5. Change the infilename : "Data_file"
  6. Change the outfilename : "Res_file"
  7. Show infile "Data_file"
  8. Show outfile "Res_file"
  9. About q_ext_tools
  10. Help
  11. Exit

           Choose:_
```

Let us choose 1 (Type 1 end press enter). Then you see the following possibilities:

```
              Q-Extension TOOLS -weights

    Data_file  --> Res_file

    1. Find spectrum of the linear codes
    2. Find the number of codewords with minimum distance
    3. Find the linear codes with minimum distance >= w
    4. Find the linear codes with  fixed number of codewords
       with weight w
    5. Find the linear codes with dual distance >= w
    6. Change the infilename : "Data_file"
    7. Change the outfilename : "Res_file"
    8. Show infile "Data_file"
    9. Show outfile "Res_file"
    10. Main menu

              Choose:_
```

In the first four points the program reads the next matrix from the input file, counts the corresponding parameter and if it satisfies the conditions writes it and the matrix in the output file. In the end it writes summarized information in the output file. For example for 1 it writes after any generator matrix the weight spectrum of the corresponding code. In the end, in the output file the program writes the number of all codes and a list of all different spectrums.

If the point 3 in main menu is chosen, it is written on the screen

```
            Q-Extension TOOLS -automorphisms, equivalence

   Data_file  --> Res_file

   1. Codes
   2. Binary Matrices
   3. Change the complexity local invariant
   4. Change the complexity global invariant
   5. Automorphisms with f. ord.
   6. Change the infilename : "Data_file"
   7. Change the outfilename : "Res_file"
No 8. Real_automorphism
   9. Show infile "Data_file"
  10. Show outfile"Res_file"
  11. Main menu

            Choose:_
```

When starting 1, the matrices from the input file are considered as generator matrices of linear codes, and when starting 2, these matrices are considered as sets of binary vectors (or incidence matrices of combinatorial designs). The procedure for automorphisms takes only nonproportional columns (as default) and it colors them depending on the number of the corresponding column in the matrix. In such a way some of the automorphisms are lost and therefore it is written **'No'** before point 8 in the menu. If you want to find the real automorphism groups, choose 8. Points 3 and 4 can be used for more difficult cases, for example combinatorial designs or binary matrices corresponding to Hadamard matrices.

The next information is for the readers, who are familiar with such type of algorithms. If you use a local invariant with complexity 3 in level 2, this means that when you fix one of the coordinates the program will compute the invariant

for more than $\binom{n}{3}m$ operations which can separate the coordinates into orbits with respect to the stabilizer of this coordinate.

8. About Efficiency. We give some execution times for a Pentium/2 GHz PC computer, under Windows XP.

program	extension	time
q_ext2_l.exe	$[5,5,1]-l->[14,5,6]$	15 sec
	$[14,5,6]-l->[22,6,11]$	2 sec
	$[22,6,11]-l->[47,7,22]$	1 sec
q_ext2_d.exe	$[45,6,22]-d->[47,7,22]$	111 sec
q_ext4_l.exe	$[3,3,1]_4-l->[5,3,\geq 1]_4$	1 sec
	$[5,3,1]_4-l->[13,4,8]_4$	66 sec
	$[3,3,1]_4-l->[13,4,8]_4$	22 min
q_ext4_d.exe	$[13,4,8]_4-d->[14,5,8]_4$	45 sec
	$[14,5,8]_4-d->[15,6,8]_4$	10 sec
	$[15,6,8]_4-d->[16,7,8]_4$	10 sec
	$[16,7,8]_4-d->[17,8,8]_4$	30 sec
q_ext_tools.exe	Randomly 1 000 000 linear codes with parameters $[35,7,d]_2$ Isomorphism test and authomorphism group	73 min
	Randomly 100 linear codes with parameters $[30,15,d]_3$ Find spectrum of the linear codes	9 min
	Randomly 100 linear codes with parameters $[30,15,d]_3$ Find the number of codewords with minimum distance	10 sec
	5000 self-orthogonal codes with parameters $[42,21,8]_2$ Isomorphism test and authomorphism group	35 min

9. Installing and starting Q-EXTENSION**.** You can run Q-EXTENSION in Pentium computer with minimum 256 MB RAM memory, under Windows $98, 2000, \ldots$, or some versions of Linux. You can make a new directory (for example c:extend) and collect all files from the package. For installing just unpack the zip file in a separate directory. You can run the different programs (exe files) in the usual way.

Acknowledgement. The author is very grateful to J. Simonis for his early and constant interest in this work. The first version of these programs contains a source code of S. Kapralov – procedure for equivalence of codes and some other procedures [7].

REFERENCES

[1] BARG A. Complexity Issues in Coding Theory. In: Handbook of Coding Theory (Eds V. S. Pless and W. C. Huffman), Elsevier, Amsterdam, 1998.

[2] BOUYUKLIEV I. About the code equivalence. In: Advances in Coding Theory and Cryptology (Eds T. Shaska, W. C. Huffman, D. Joyner, V. Ustimenko), Series on Coding Theory and Cryptology, 3. World Scientific Publishing Co. Pte. Ltd., Hackensack, NJ, 2007. (In Press)

[3] BOUYUKLIEV I., V. BAKOEV. Algorithms for computing the number of codewords of fixed weight in linear codes. Proc. of the Intern. Workshop on Coding theory and Applications, 14–21 June 2005, Pamporovo, Bulgaria.

[4] BOUYUKLIEV I., J. SIMONIS. Some new results for optimal ternary linear codes. *IEEE Trans. Inform. Theory* **48** (2002), 981–985.

[5] BROUWER A. E. Bounds on the size of linear codes. In: Handbook of Coding Theory (Eds V. S. Pless, W. C. Huffman), Elsevier, Amsterdam, 1998, 295–461.

[6] DODUNEKOV S. Minimal block length of a linear q-ary code with specified dimension and code distance. *Problems Inform. Transmission* **20** (1984), 239–249.

[7] Bogdanova G., P. Christov, S. Kapralov. The new version of QLC – a computer program for linear codes studying. Proc. Inter. Workshop OCRT′95, Sozopol, Bulgaria, 1995, 11–14.

[8] Huffman W. C., V. Pless. Fundamentals of Error-Correcting Codes. Cambridge University Press, Cambridge, 2003.

[9] McKay B. Practical graph isomorphism. *Congr. Numer.* **30** (1981), 45–87.

[10] Kaski P., P. Ostergard. Classification Algorithms for Codes and Designs. Springer, 2006.

[11] Petrank E., R. M. Roth. Is code equivalence easy to decide? *IEEE Trans. Inform. Theory* **43**, *5* (1997), 1602–1604.

[12] Berlekamp E., R. J. McEliece, H. C. van Tilborg. On the inherent intrac tability of certain coding problems. *IEEE Trans. Inform. Theory* **24**, *3* (1978), 384–386.

Institute of Mathematics and Informatics
Bulgarian Academy of Sciences
P.O.Box 323
5000 Veliko Tarnovo, Bulgaria
e-mail: ilia@moi.math.bas.bg

Received October 23, 2006
Final Accepted June 27, 2007