To appear in the *International Journal of Computer Mathematics*
Vol. 00, No. 00, Month 20XX, 1–10

What is GenSelfDual?

Iliya Bouyukliev[a], Stefka Bouyuklieva[a,b]*, Maria Dzhumalieva-Stoeva[a,b] and Venelin Monev[b]

[a] *Institute of Mathematics and Informatics, Bulgarian Academy of Sciences, Veliko Tarnovo, Bulgaria;* [b] *Faculty of Mathematics and Informatics, Veliko Tarnovo University, Bulgaria*

(*v1 released November 2014*)

In this paper we present a developed software in the area of Coding Theory. Using it, all binary self-dual codes with given properties can be classified. The programs have consequent and parallel realizations.

Keywords: self-dual codes; classification; software; isomorph-free generation

2010 AMS Subject Classification: 68N19; 68R05; 90C09

1. Introduction

In this note, we present the package GenSelfDual. It is a free math software that supports research in the area of Coding Theory. GenSelfDual consists of two programs for the construction and classification of binary self-dual codes, GenSelfDualAllD and GenSelfDualD4. They have both consequent and parallel realizations. The first program (GenSelfDualAllD) gives a recursive construction - starting from a given set of inequivalent binary self-dual codes of smaller length (the default is the only $[4, 2, 2]$ self-dual code) one can obtain self-dual codes of a given length n ($n \leq 42$ in the current version). The second program helps us to obtain all binary self-dual codes of length n and minimum distance 4 starting from the set of all self-dual codes of length $n - 4$. To use these programs, a knowledge of a programming language is not needed. The software is available in http://www.moi.math.bas.bg/ $\sim$ iliya. These two programs could be successfully combined with the programs from the package Q_Extension, especially with q_ext_tools. Using q_ext_tools, one can calculate the spectrums, the covering radii and the automorphism groups of the codes, to obtain only the codes with fixed number of codewords with a given weight, etc. (see [10]).

The largest software systems for computing with error-correcting codes are Magma Computational Algebra System [9] and GAP package GUAVA [3]. In [17] Harada and Munemasa present their Magma programs for checking equivalences and mass formulae of binary self-dual codes (for lengths up to 34, and for length 40, minimum weight 8). The database [17] contains classification results for binary self-dual codes of length $n \leq 40$ (for lengths 38 only the optimal codes and for length 40 all doubly even and all optimal codes). The package GenSelfDual is especially devoted to the construction and classification of binary self-dual codes and therefore it is faster and more efficient in the study of the

*Corresponding author. Email: stefka@uni-vt.bg

codes of this type than the cited software systems. Using its programs, the authors have classified all self-dual codes of lengths 38 and 40 [11–13].

The paper is organized as follows. In Section 2 we present the needed definitions and a summary on the construction and classification results for binary self-dual codes. Section 3 is devoted to the implemented algorithms. In Section 4 we describe the input and output of the programs. Section 5 gives some details how one can use these programs. The parallel realizations are presented in Section 6. In the last section we show an example for a parallel implementation of GenSelfDualAllD.

2. Preliminaries

Throughout this paper all codes are assumed to be binary. A binary linear code of length n, dimension k and minimum distance d is said to be an $[n, k, d]$ code. A $k \times n$ matrix G_C whose rows form a basis of C is called a generator matrix of C.

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. The weight enumerator (or the weight function) is the polynomial

$$W(x, y) = \sum_{i=0}^{n} A_i x^{n-i} y^i.$$

Sometimes it is more convenient to consider $W(y) = W(1, y) = \sum_{i=0}^{n} A_i y^i$ as a weight function.

A code is called *doubly even* if the weights of all codewords are divisible by 4. The *dual code* of a code C is defined as $C^\perp = \{x \in \mathbb{F}_2^n \mid x \cdot y = 0 \text{ for all } y \in C\}$, where $x \cdot y$ is the standard inner product. A code C is called *self-dual* if $C = C^\perp$. A self-dual code which is not doubly even is called *singly even*. A doubly even self-dual code of length n exists if and only if $n \equiv 0 \pmod 8$, while singly even self-dual codes exist for all even lengths.

Two binary codes are called *equivalent* if one can be obtained from the other by a permutation of coordinates. The permutation $\sigma \in S_n$ is an *automorphism* of the code C of length n, if $C = \sigma(C)$ and the set of all automorphisms of C forms a group called the *automorphism group* of C, denoted by $\mathrm{Aut}(C)$. Then the number of codes equivalent to C is $n!/|\mathrm{Aut}(C)|$.

If we consider the action of the symmetric group S_n on the set Ω_n of all self-dual codes of length n, then two codes from this set are equivalent if they belong to the same orbit. This action induces an equivalence relation in Ω_n and the equivalence classes are the orbits with respect to the action of S_n. To classify the self-dual codes of length n means to find exactly one representative of each equivalence class.

The classification of self-dual codes is an important problem. The self-dual codes are connected with other combinatorial structures, like combinatorial designs and lattices, therefore a database with all inequivalent self-dual codes with given parameters is needed for the study of these structures. Such a database could be used in the construction and classification of codes and connected structures of larger lengths and dimensions. Moreover, the classification results give us a look on the rate of increase for the number of codes and how far is the real number from the known upper and lower bounds.

The classification of self-dual codes began in the seventies in the work of Vera Pless [22], where she classified the binary self-dual codes of length $n \leq 20$. The method used in the beginning remained essentially the same throughout the succeeding classifications.

This is a recursive classification which proceeds from smaller to larger length and codes are classified up to equivalence. There is a formula for the number of all self-dual codes of length n called a mass formula. To classify self-dual codes of length n, it is necessary to find inequivalent self-dual codes $C_1, \ldots, C_r$ so that the following mass formula holds [25]:

$$\sum_{i=1}^{r} \frac{n!}{|\mathrm{Aut}(C_i)|} = \prod_{i=1}^{n/2-1} (2^i + 1). \tag{1}$$

For the verification of the classification results we use also another mass formula [18]. The family $U(n, d)$ of inequivalent self-dual codes of length n and minimum weight at most d is a complete set of representatives for the equivalence classes of self-dual codes of length n and minimum weight at most d if and only if

$$\sum_{C \in U} \frac{n!}{|\mathrm{Aut}(C)|} |\{x \in C | \mathrm{wt}(x) = d\}| = \binom{n}{d} \prod_{i=1}^{n/2-2} (2^i + 1). \tag{2}$$

In the survey [19] Huffman summarized the classification of all binary self-dual codes of length $n \leq 36$. Harada and Munemasa in [18] completed the classification of the self-dual codes of length 36 and created a database of self-dual codes [17]. The doubly-even self-dual codes of length 40 were also classified by Betsumiya, Harada and Munemasa [6]. Moreover, using these codes, they classified the optimal self-dual $[38, 19, 8]$ codes. A classification of extremal self-dual codes of length 38 was done independently in [2] by different techniques.

In the paper [11] the first two authors presented the theoretical foundations of a new classification algorithm. It gives as output exactly one representative of every equivalence class without using tests for equivalence at all. To develop this algorithm we have used an approach presented by Brendan McKay known as isomorph-free exhaustive generation [21]. The algorithm changed extremely the speed of the generation of the inequivalent codes of a given length. Using this algorithm, the first two authors succeeded to complete the classification of all binary self-dual codes of length 38. The result was verified with the help of the mass-formulae (1) and (2). On the base of the same technique and the constructed self-dual $[38, 19, 6$ and $8]$ codes, the first two authors classified the optimal self-dual $[40, 20, 8]$ codes in a joint work with M. Harada [12]. The calculations took about 95 days on a PC Intel i5 4 core processor. This result was approved, and the classification of all self-dual codes of length 40 was completed by Bouyukliev, Dzhumalieva-Stoeva and Monev in [13].

We summarize the classification results on binary self-dual codes of length $n \leq 40$ in Table 1. The number of all inequivalent singly-even (Type I) and doubly-even (Type II) codes is denoted by $\sharp_I$ and $\sharp_{II}$, respectively. In the table $d_{max,I}$ ($d_{max,II}$) is the biggest possible minimum distance for which Type I (Type II) codes with the given length exist, and $\sharp_{max,I}$ ($\sharp_{max,II}$) is their number. This is a new version of Table 1 from [19].

3. About the algorithms

The construction methods, which we use, are based on the following theorems.

Table 1. Binary self-dual codes of length $n \leq 40$ [19]

n	$\sharp_I$	$\sharp_{II}$	$d_{max,I}$	$\sharp_{max,I}$	$d_{max,II}$	$\sharp_{max,II}$	Reference
2	1		2	1			[22]
4	1		2	1			[22]
6	1		2	1			[22]
8	1	1	2	1	4	1	[22]
10	2		2	2			[22]
12	3		4	1			[22]
14	4		4	1			[22]
16	5	2	4	1	4	2	[22]
18	9		4	2			[22]
20	16		4	7			[22]
22	25		6	1			[24]
24	46	9	6	1	8	1	[24]
26	103		6	1			[14, 15, 23]
28	261		6	3			[14, 15, 23]
30	731		6	13			[14, 15, 23]
32	3210	85	8	3	8	5	[8, 14, 16]
34	24147		6	938			[7]
36	519492		8	41			[1, 18]
38	38682183		8	2744			[2, 6, 11]
40	8249963738	94343	8	10200655	8	16470	[6, 12, 13]

THEOREM 3.1 *Let C be a binary self-dual $[n, k = n/2, d \geq 4]$, and $C_0 = \{(x_1, \ldots, x_n) \in C,\ x_{n-1} = x_n\}$. If C_1 is the punctured code of C_0 on the coordinate set $T = \{n-1, n\}$ then C_1 is a self-dual $[n-2, k-1, d_1 \geq d-2]$ code.*

THEOREM 3.2 *Let C be a binary self-dual $[n, k = n/2, 4]$ code and $T = \{i_1, i_2, i_3, i_4\}$ be the support of a codeword of weight 4. If C_0 is the shortened code of C on the set $T1 = \{i_1, i_2\}$ then the punctured code $C_1 = C_0^{T2}$ of C_0 on the set $T2 = \{i_3, i_4\}$ is a self-dual $[n-4, n/2-2, \geq 2]$ code.*

Theorem 3.1 is the main point in the construction algorithm implemented in the program GENSELFDUALALLD. It's detailed description is given in [11]. The program GENSELFDUALD4 is devoted to the construction and classification of binary self-dual codes with minimum distance 4. It's construction algorithm is based on Theorem 3.2 and is explained in [13].

Actually, we do not use test for equivalence. Instead of that we develop a McKey type algorithm for isomorph-free generation (see [20] and [21]). This means that in the generation process we take only one representative from each equivalence class. The algorithm uses a canonical representation of the objects (matrices). The main advantage of this technic is that the equivalence (isomorphism) test is reduced to check of coincidence of the canonical representations of the structures. In the case of many inequivalent codes the computational time for comparing is growing fast. We split the set of inequivalent 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 [4]. Very im-

portant for our package is the algorithm for generating of an invariant set of codewords. This generation is connected with the computation of the weight spectrum and/or the codewords with a given weight. Unfortunately, computing the weight enumerator of a code becomes computationally intractable when its size grows and according [5] it is in fact NP-hard (see also [4]).

4. Input and output of the programs

There are no restrictions for the input file name for both programs. It is a text file which could contain any information but only generator matrices in a specific form are visible for the programs. Above any such matrix, there must be a row with parameters which has to contain the character '?' in the beginning, the number of the rows, the number of the columns of the matrix, and '2' in the end (as a generator matrix of a binary linear code). So we have the following type of data:

?3 6 2 $Name1$
100100
010010
001001

The generator matrices of the input codes have to be in systematic form. Moreover, all input codes have to be inequivalent otherwise the program will output some equivalent codes. 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 considers 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.

These programs create usual text files, in which generator matrices of the considered codes in the given format are written, as well as some additional information for how the program has been started. The file's names are x`M.txt`, x`S.txt`, x`R.txt`, x`Mas.txt`, where x is the number of the considered interval (for the intervals see the next section). These files can be opened, explored and changed with any text editor. The first file contains generator matrices of the constructed codes in the same format (with a row above every matrix containing the character **'?'** followed by the dimension, length, number of the elements of the field, name (identificator)). 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.

5. How these programs can be used for code classification?

In this section we show how one can use these programs and especially the first one to classify all binary self-dual codes of a given dimension. As an example we classify the self-dual codes of dimension 16 starting from the self-dual $[8, 4]$ codes. There are two inequivalent self-dual codes of length 8, namely A_8 and C_2^4 [22], with minimum distances 4 and 2, respectively. After starting, GenSelfDualAllD gives us the following possibilities:

```
     <Program for a classification of binary self-dual codes>

  1. Start
  2. Change the input file
     Now the input file is: "4_2"
     The codes have parameters [4,2] <Total count = 1>
    <Output files are: 1M.txt, 1S.txt, 1R.txt, 1Mas.txt>

  3. The dimension of the constructed codes is k=20
  4. The size of the intervals is 1000
     <There is 1 interval>

  5. The program runs for interval 1
  6. Printing the generator matrices: YES
  7. About
  8. Exit

          Choose:
```

To change the input file, we press 2 and then print the name of the file, in our case `8_4_2.2`. Then we press 3 to change the dimension to $k = 16$. Now the menu is:

```
     <Program for a classification of binary self-dual codes>

  1. Start
  2. Change the input file
     Now the input file is: "8_4_2.2"
     The codes have parameters [8,4] <Total count = 2>
    <Output files are: 1M.txt, 1S.txt, 1R.txt, 1Mas.txt>

  3. The dimension of the constructed codes is k=16
  4. The size of the intervals is 1000
     <There is 1 interval>

  5. The program runs for interval 1
  6. Printing the generator matrices: YES
  7. About
  8. Exit

          Choose:
```

Then we choose 1 (type 1 and press enter) to start the program.

The program writes generator matrices of the constructed codes in a file with the name `1M.txt`. The file `1S.txt` contains some information about the codes. For example the first row is

`? 16 120 560 1820 1371195958099968000`,

which means that the first code contains 16 codewords of weight 2, 120 of weight 4, 560 of weight 6 and 1820 of weight 8. The last number is the order of the automorphism group of this code. The same information is presented for each code.

The files `1R.txt` and `1Mas.txt` contain some information about the mass formula (1).

Five variables are written in the file `1R.txt`:

(1) *totallarge* - a check for the number of the constructed codes. It must be equal to *all_mat*, which for the length 32 is 3295;
(2) $NUMGS$ - it increases with 1 in the parent test function, so this is a counter for the codes which are tested by the parent test. For length 32 $NUMGS = 204310$;
(3) *numcan* - it increases with 1 in the function CANON(), so this is a counter for the codes for which a canonical form is made. For our example $numcan = 10409$;
(4) *all_mat* - the number of the constructed codes;
(5) $COD8$ - 159500 a check for the considered codes with minimum weight 8.

The last file `1Mas.txt` contains some partial calculations for the mass-formula (2). For the example the content is:

```
mass product right 2 = 47968158914345809928647166729850000
mass product right 4 = 34776915212900712198269195879141250000
mass product right 6 = 87637826336509794739638373615435950000
mass product right 8 = 10172247699773458317993739794648815625000
mass without binomial: 31690899182745924305480625132468843750
```

While working, the program writes on the screen some useful information for the tree of the back-track search.

We list also some additional comments and options related to the programs:

- If the codes for extension are too many, we divide them into intervals. Point 4 shows the size of the considered intervals. The default is 1000 which means that each interval (instead of the last one) consists of 1000 generator matrices for extension.
- Point 5 shows which of the interval is considered.
- `YES` in points `6. Printing the generator matrices`, means that generator matrices of the constructed inequivalent codes will be written in the file `1M.txt`. If we put `NO` there, the file `1M.txt` will contain only one row with information about the current code as this is explained above. We need this possibility because for larger lengths (for example 40), the codes are too many and the file becomes too huge if we write the matrices there.

6. Parallel programs

Parallel programs are implemented by the message passing interface MPI with the programming language C. Since the input data are identical, namely matrices with given parameters, for which the same actions are applied, SPMD (Single Program Multiple Data) model is used. For our implementation this scheme is based on the master worker paradigm. In this strategy, one process is determined as a master process, which has a function of distribute the data (management) among worker processes and its coordination. In this model, one process is defined as a master and it takes the function to allocate and coordinate the work between other processes called workers. For the software implementation MPICH2 for Windows operating system is used. MPICH2 can also be used to run the programs with required minimum number of processes 2. The user can set the input parameters by the menu file placed on disc C or disc D. In the starting step, master process reads from the menu file the full path name and input file name, placed in the same directory. Then, the master process broadcasts the initializing values to the worker processes. After the input file is opened, master process reads the next matrix and sends it to an idle worker process for augmentation. After reading the all matrices

from the input file, the master process sends a quit signal and requires the number of the constructed new codes and the partial sums in the mass formulas from each worker process. Then the master summarizes these results.

As in the consequent realization, we use two parallel programs. The first one is for generation of all self-dual codes with given parameters and the menu file is named `menuMPI6`. The second version is for self-dual codes with minimum distance 4 and in this case the menu file is `menuMPI4`. In both files there are no comments.

Example for menuMPI6:

```
D:/WorkingDirectory/SelfDualCodes/  //working directory
16_8.txt                            //name of the input file
10                                  //dimension of the constructed codes
10000                               //size of the intervals
1                                   //current working interval
1                                   //print the matrices: 1-yes, 0-no
////
```

Example for menuMPI4:

```
D:/WorkingDirectory/SelfDualCodes4/  //working directory
16_8.txt                             //name of the input file
10000                                //size of the intervals
1                                    //current working interval
1                                    //print the matrices: 1-yes, 0-no
////
```

Note that the separator for working directory is the symbol / and it is placed at the end, too. If either working directory or input file name is not correct an error message will appear in the console. The parallel versions of the programs are not fully validated for all parameters and the user must be careful. As in the sequential case any process in each interval generates four files with information as this is described in the previous section, with one more file with information about the current state. For example, the first program in process 3 for interval 2 generates the files `cpu3_i2M.txt`, `cpu3_i2Mas.txt`, `cpu3_i2R.txt`, `cpu3_i2S.txt`, and the console file with the current state of process 3 named `cpu3_my_console.txt`. The corresponding files for the second program are `cpu3_i2M_d4.txt`, `cpu3_i2Mas_d4.txt`, `cpu3_i2R_d4.txt`, `cpu3_i2S_d4.txt`, `cpu3_my_console.txt`. Note that, the process 0 is the master and it does not generate codes. In the end, the sums of the mass formulae are printed in a file named `cpu0_i2Mas.txt` (respectively `cpu0_i2Mas.txt_d4` for the secong program) and the number of all generated codes from workers are in the file `cpu0_i2R.txt` (`cpu0_i2R_d4.txt`). More information about the execution of the programs is written in the file `console.txt`.

7. Partial results for the self-dual codes of length 42

Using the program GenSelfDualAllD, we construct inequivalent self-dual $[42, 21, d \geq 6]$ codes such that some of their punctured codes contain a self-dual $[40, 20, 8]$ code with weight enumerator

$$W_{40,C,\beta} = 1 + (125 + 16\beta)y^8 + (1664 - 64\beta)y^{10} + (10720 + 32\beta)y^{12} + \cdots .$$

for $\beta = 4$. The input file contain generator matrices of all 64 692 inequivalent optimal codes of length 40 with the given weight enumerator. We run the parallel implementation of GenSelfDualAllD with six working processes. The computations took about three days on INTEL i7 processor. As a result we obtain 28 376 939 inequivalent self-dual codes of length 42. The optimal codes among them are 2 439 874 and they have 13 different weight enumerators, namely

$$W_1(y) = 1 + (84 + 8\beta)y^8 + (1449 - 24\beta)y^{10} + (10640 - 16\beta)y^{12} + \cdots,$$

for $\beta = 4, 5, \ldots, 16$. The number of the constructed codes with different weight enumerators are given in Table 2. These codes have automorphism groups of order 1 (2 414 550, or 98.96 percent of the codes), 2^m for $m = 1, 2, \ldots, 10, 12$; $3 \cdot 2^m$ for $m = 0, 1, \ldots, 7$; 36 (one code) and 144 (one code). So many optimal self-dual codes of length 42 have not been constructed until now but we expect much larger number for all inequivalent $[42, 21, 8]$ SD codes.

Table 2. Number of the constructed inequivalent codes of length 42

β	4	5	6	7	8	9	10
# codes	683953	886496	526666	256915	65442	16523	3185
β	11	12	13	14	15	16	
# codes	539	129	10	13	1	2	

References

[1] C. Aguilar-Melchor and P. Gaborit, *On the classification of extremal* $[36, 18, 8]$ *binary self-dual codes*, IEEE Trans. Inform. Theory 54 (2008), pp. 4743–4750.

[2] C. Aguilar-Melchor, P. Gaborit, J.-L. Kim, L. Sok and P. Solé, *Classification of extremal and s-extremal binary self-dual codes of length 38*, IEEE Trans. Inform. Theory 58 (2012), pp. 2253–2262.

[3] R. Baart, T. Boothby, J. Cramwinckel, J. Fields, D. Joyner, R. Miller, E. Minkes, E. Roijackers, L. Ruscio, C. Tjhai, *GUAVA - a GAP package, Version 3.12, 21/05/2012*, http://www.gap-system.org/Packages/guava.html

[4] A Barg, Complexity Issues in Coding Theory, *Handbook of Coding Theory*, V. S. Pless and W. C. Huffman, Eds., Elsevier, Amsterdam, 1998.

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

[6] K. Betsumiya, M. Harada and A. Munemasa, *A complete classification of doubly even self-dual codes of length* 40, Electronic J. Combin. 19 (2012), #P18 (12 pp.).

[7] R. T. Bilous, *Enumeration of the binary self-dual codes of length* 34, J. Combin. Math. Combin. Comput. 59 (2006), pp. 173–211.

[8] R.T. Bilous, G.H.J. Van Rees, *An enumeration of binary self-dual codes of length* 32, Designs, Codes and Cryptography 26 (2002), pp. 61–86.

[9] Wieb Bosma, John Cannon, and Catherine Playoust, *The Magma algebra system. I. The user language*, J. Symbolic Comput. 24 (1997), pp. 235265.

[10] I. Bouyukliev, *What is* Q-Extension*?*, Serdica Journal of Computing 1 (2007), pp. 115–130.

[11] S. Bouyuklieva and I. Bouyukliev, *An algorithm for classification of binary self-dual codes*, IEEE Trans. Inform. Theory 58 (2012), pp. 3933–3940.

[12] S. Bouyuklieva, I. Bouyukliev and M. Harada, *Some extremal self-dual codes and unimodular lattices in dimension* 40, Finite Fields Appl. 21 (2013), pp. 67-83.

[13] I. Bouyukliev, M. Dzhumalieva-Stoeva and V. Monev, *Classification of Binary Self-dual Codes of Length* 40, preprint.

[14] J.H.Conway and V.Pless, *On the enumeration of self-dual codes*, Journ. Combin. Theory, ser. A, 28 (1980), pp. 26-53.
[15] J.H.Conway, V.Pless and N.J.A.Sloane, *The binary self-dual codes of length up to 32: a revised enumeration*, Journ. Combin. Theory, ser. A, 60 (1992), pp. 183-195.
[16] J.H.Conway and N.J.A.Sloane, *A new upper bound on the minimal distance of self-dual codes*, IEEE Trans. Inform. Theory 36 (1991), pp. 1319–1333.
[17] M. Harada and A. Munemasa, *Database of Self-Dual Codes*, Online available at http://www.math.is.tohoku.ac.jp/ munemasa/selfdualcodes.htm.
[18] M. Harada and A. Munemasa, *Classification of self-dual codes of length* 36, Adv. Math. Commun. 6 (2012), pp. 229–235.
[19] W.C. Huffman, *On the classification and enumeration of self-dual codes*, Finite Fields Appl. 11 (2005), pp. 451–490.
[20] P. Kaski and P. R. Östergård, *Classification Algorithms for Codes and Designs*, Springer, 2006.
[21] B. D. McKay, *Isomorph-free exhaustive generation*, J. Algorithms 26 (1998), pp. 306–324.
[22] V. Pless, *A classification of self-orthogonal codes over* $GF(2)$, Discrete Math. 3 (1972), pp. 209-246.
[23] V. Pless, The children of the (32,16) doubly even codes, *IEEE Trans. Inform. Theory*, **24** (1978), 738-746.
[24] V. Pless and N.J.A. Sloane, On the classification and enumeration of self-dual codes, *Journ. Combin. Theory*, ser. A, **18** (1975), 313-335.
[25] E.M. Rains and N. J. A. Sloane, Self-dual codes, in *Handbook of Coding Theory*, V. S. Pless and W. C. Huffman, Eds., Elsevier, Amsterdam, 1998, 177–294.