Mathematical and Software Engineering, vol. 4, no. 2, 18–23, 2018.
Varεpsilon Ltd, http://varepsilon.com/index.php/mse

A note on reducing the computation time for minimum distance and equivalence check of binary linear codes

Nikolay Yankov[1] **and Krassimir Enev**[2]

[1]Faculty of Mathematics and Informatics Konstantin Preslavski University of Shumen, 9712 Shumen, Bulgaria, e-mail: jankov_niki@yahoo.com, ORCID 0000-0003-3703-5867
[2]College Dobrich, Konstantin Preslavski University of Shumen, 9712 Shumen, Bulgaria, e-mail: kr.enev@shu.bg

Abstract

In this paper we show the usability of the Gray code with constant weight words for computing linear combinations of codewords. This can lead to a big improvement of the computation time for finding the minimum distance of a code.

We have also considered the usefulness of combinatorial 2-$(t, k, 1)$ designs when there are memory limitations to the number of objects (linear codes in particular) that can be tested for equivalence.

Keywords: Classification, Combinatorial design, Linear code

1 Introduction

Binary linear codes and self-dual codes in particular are extensively studied for the plethora of connections to communication, cryptography, combinatorial designs, among many. When computing self-dual codes one should be aware that with the increase of the code length the number of codes also rises exponentially.

The classification of binary self-dual codes begun in 1972 with [11] wherein all codes of lengths $n \leq 20$ are classified. Later Pless, Conway and Sloane classify all codes for $n \leq 30$ [7]. Next lengths: 32 is due to Bilous and Van Rees [2], 34 by Bilous [1], 36 by Harada and Munemasa in [9]. Latest development in this area are for length 38 in [5] and for $n = 40$ due to Bouyukliev, Dzumalieva-Stoeva and Monev in [4].

As length of the code gets bigger the number of codewords rises exponentially and one need efficient algorithms for computing the minimum distance of a linear code, and also efficient ways to check codes for equivalence when there are memory limitations.

This paper is organized as follows: In Section 2 we outline an introduction to linear codes, self-dual codes, combinatorial designs and Gray codes. Next, in Section 3, we discuss how a reduction in computation time for minimum distance of linear code with constant-weight Gray code can be achieved. In Section 4 we explain a method for reducing the computation time for code equivalence by the use of combinatorial 2-designs. We conclude in Section 5 with a few final notes.

2 Definitions and preliminaries

Let $\mathbb{F}_q$ be the finite field of q elements, for a prime power q. A linear $[n,k]_q$ *code* C is a k-dimensional subspace of $\mathbb{F}_q^n$. The elements of C are called *codewords*, and the *(Hamming) weight* of a codeword $v \in C$ is the number of the non-zero coordinates of v. We use $\mathrm{wt}(v)$ to denote the weight of a codeword. The *minimum weight* d of C is the minimum nonzero weight of any codeword in C and the code is called an $[n,k,d]_q$ code. A matrix whose rows form a basis of C is called a *generator matrix* of this code.

Let $(u,v) \in \mathbb{F}_q$ for $u,v \in \mathbb{F}_q^n$ be an inner product in $\mathbb{F}_q^n$. The *dual code* of an $[n,k]_q$ code C is $C^\perp = \{u \in \mathbb{F}_q^n \mid (u,v) = 0 \text{ for all } v \in C\}$ and $C^\perp$ is a linear $[n,n-k]_q$ code. In the binary case the inner product is the standard one, namely, $(u,v) = \sum_{i=1}^n u_i v_i$. If $C \subseteq C^\perp$, C is termed *self-orthogonal*, and if $C = C^\perp$, C is *self-dual*. We say that two binary linear codes C and C' are *equivalent* if there is a permutation of coordinates which sends C to C'. In the above definition the code equivalence is an equivalence relation is a binary relation that is reflexive, symmetric and transitive. Denote by $\mathrm{Eq}(a,b)$ some function that checks for equivalence all pairs of elements in both sets of linear codes a and b. For more information on codes we encourage the reader to [10].

When working with linear codes it is often needed for certain algorithm to pass trough all (or part) of binary vectors of given length. One way to make the generation efficient is to ensure that successive elements are generated such that they differ in a small, pre-specified way. One of the earliest examples of such a process is the Gray code generation. Introduced in a pulse code communication system in 1953 [8], Gray codes now have applications in diverse areas: analogue-to-digital conversion, coding theory, switching networks, and more. For the past 70 years Gray codes have been extensively studied and currently there are many different types of Gray code.

A binary Gray code of order n is a list of all 2^n vectors of length n such that exactly one bit changes from one string to the next.

A t-(v,k,λ) design D is a set X of v points together with a collection of k-subsets of X (named blocks) such that every t-subset of X is contained exactly in λ blocks. The block intersection numbers of D are the cardinalities of the intersections of any two distinct blocks.

3 Reducing computation time for minimum distance of linear code with constant-weight Gray code

Assume we have a linear binary $[n,k]$ code $\mathcal{C}$ and we need to find its minimum distance d. Denote by $\mathcal{G}$ the generator matrix of the code $\mathcal{C}$ with rows $r_1,\dots,r_k$. The obvious and direct approach is to compute all codewords of $\mathcal{C}$ and find their weight. This means that all 2^k linear combinations of t ($1 \le t \le k$) of the rows of $\mathcal{G}$ must be computed using Algorithm 1.

Algorithm 1: The direct approach

```
for (i1 = 1; i1 <= k-t+1; i1++) {
    for (i2 = i1+1; i2 <= k-t+2; i2++) {
      for (i3 = i2+1; i3 <= k-t+3; i3++) {
        ...
          for (it = itm1+1; it <= k; it++) {body}... }}
```

Then for each of the $\binom{k}{t}$ combination we need to compute t cycles and essentially t operations. Furthermore, in the body of this algorithm we need to find the codeword $c \in \mathcal{C}$ which is a linear combination of those rows of the generator matrix $\mathcal{G}$ that are chosen for the current combination, i.e. $c = \sum_{s=1}^{t} r_{i_s}$, which will be represented by t "exclusive or" (`xor`) operations $c = r_{i_1} \oplus r_{i_2} \oplus \ldots \oplus r_{i_t}$.

Our approach is to use Gray code for generating combinations in such a way that each successive combination is generated by the previous one with only two `xor` operations. Two `xor` operations are the absolute minimum since, if we have to switch from one combination of t elements to another, one `xor` will add or remove a position making a $t+1$ or a $t-1$ combination. In [12] it was proved that the set of $\binom{k}{t}$-vectors of weight t, when chained according to the ordering on the Gray code $\mathcal{G}_k$, has a Hamming distance of exactly two between every pair of adjacent code vectors. Also in [12] an algorithm for generating the constant-weight code vectors on a Gray code was given. Later in [3] a more efficient recursive algorithm was introduced (Algorithm 2).

What we want to do is to find in Gray code $\mathcal{G}_k$ those k-tuples that have the same weight t, for example when $k = 4$ for $t = 1$ we have: 0000→**0001**→0011→**0010**→0110→0111 →0101→**0100**→1100→1101→1111→1110→1010→1011→1001→**1000** and similarly, for $t = 2$ we have: 0000→0001→0011→0010→**0110**→0111→**0101**→0100→**1100**→1101→ 1111→1110→**1010** →1011→ **1001**→1000. Note that Algorithm 2 starts with the word 1^t0^{k-t} and finishes with $0^{k-t}1^t$.

Example 1: If we need to find all triples in $\mathcal{G}_6$ we have a total of 20 triples. We start with 000111 and from Gray code we have the following sequence of positions to change

$$[2,4],[1,2],[1,3],[2,5],[1,2],[2,3],[1,4],[1,2],[1,3],[2,6],[1,2],[2,3],$$
$$[3,4],[1,5],[1,2],[2,3],[1,4],[1,2],[1,3].$$

So the sequence of triples is as follows

$$\{1,2,3\},\{1,3,4\},\{2,3,4\},\{1,2,4\},\{1,4,5\},\{2,4,5\},\{3,4,5\},\{1,3,5\},\{2,3,5\},\{1,2,5\},$$
$$\{1,5,6\},\{2,5,6\},\{3,5,6\},\{4,5,6\},\{1,4,6\},\{2,4,6\},\{3,4,6\},\{1,3,6\},\{2,3,6\},\{1,2,6\}.$$

Algorithm 2: Constant t-weight $(0 < t \leq k)$ Gray code $\mathcal{G}_t$ [3]

$$\textbf{for } j = 1 \textbf{ to } t \textbf{ do } \begin{cases} g_j = 1 \\ \tau_j = j + 1 \end{cases}$$

$$\textbf{for } j = t + 1 \textbf{ to } k + 1 \textbf{ do } \begin{cases} g_j = 0 \\ \tau_j = j + 1 \end{cases}$$

$$s = k$$

$$\tau_1 = k + 1$$

$$i = 0$$

$$\textbf{while } i{<}k{+}1 \textbf{ do } \begin{cases} \textbf{output } (g_k, g_{k-1} \ldots, g_1) \\ i = \tau_1 \\ \tau_1 = \tau_i \\ \tau_i = i + 1 \\ \textbf{if } g_i = 1 \textbf{ then } \begin{cases} \textbf{if } s \neq 0 \textbf{ then } g_s = \overline{g}_s \\ \qquad\quad \textbf{else } g_{i-1} = \overline{g}_{i-1} \\ s = s + 1 \end{cases} \\ \qquad\qquad \textbf{else } \begin{cases} \textbf{if } s \neq 1 \textbf{ then } g_{s-1} = \overline{g}_{s-1} \\ \qquad\quad \textbf{else } g_{i-1} = \overline{g}_{i-1} \\ s = s - 1 \end{cases} \\ g_i = \overline{g}_i \\ \textbf{if } s = i - 1 \text{ or } s = 0 \\ \qquad \textbf{then } s = s + 1 \\ \qquad \textbf{else } \begin{cases} s = s - g_{i-1} \\ \tau_{i-1} = \tau_i \\ \textbf{if } s = 0 \textbf{ then } \tau_1 = i - 1 \\ \qquad\quad \textbf{else } \tau_1 = i + 1 \end{cases} \end{cases}$$

Usually, when we need to compute the minimum weight of a binary code C, we start with the initializing $r_1 \oplus \cdots \oplus r_t$, then we need the pair of position that should be changed to obtain the next t-tiple and so on. Since for given $i(1 < i < 2^t)$ it is easy to find the i-th t-weight vector and begin with the linear combination generated by it, the algorithm can be parallelized to accommodate its use on multiple CPU cores.

4 Reducing computation time for code equivalence with combinatorial 2-designs

What can be done when there are more linear codes that the equivalence algorithm can accommodate in the allowed memory. We consider the case when all codes have the same weight enumerator and also the same order of their automorphism group. This means that all other options for reducing the number of codes we are considering are exhausted.

The question then is: How can we efficiently ensure that the algorithm will check every pair of codes. If we have $s \in \mathbb{N}$ times more codes that that algorithm can check, we can split this into $2s$ halves of sets of codes and then check all $\binom{2s}{2} = s(2s - 1)$ pairs for equivalence. This is not very efficient since this has the quadratic $O(s^2)$ efficiency. The more efficient way is to use 2-$(v, k, 1)$ combinatorial design, which ensures that every pair of points (sets of codes in our case) appear exactly in one block and is checked for equivalence only once. Such designs exists, for example, when $\lambda = 1$ and $v = k^2$, we have a projective plane: X is the point set of the plane and the blocks are the lines [13].

For example, consider the case of 7 sets $i_1, \ldots, i_7$ of binary self-dual codes. If we use the standard approach we should do the tests $\mathrm{Eq}(i_j, i_s)$, $1 \leq i_k < i_j \leq 7$ for all $\binom{7}{2} = 21$ pairs of sets. Now, consider using the combinatorial design approach, viz. the Fano plane (see [6]) illustrated in Fig. 1. It is well known that the Fano plane is a combinatorial 2-$(7, 3, 1)$-design [6]. This means that every pair of sets (i_j, i_s), $1 \leq i_j < i_s \leq 7$ appear in exactly one of the 7 blocks (the blocks of Fano plane are the 6 lines and the circle), so if a code is present in different sets it is reduced to only one copy.

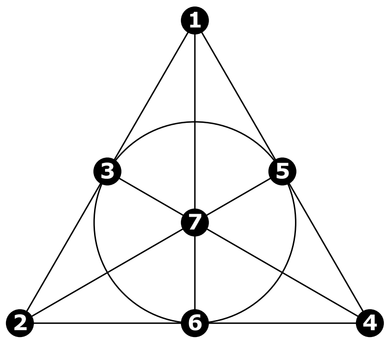

Figure 1: Fano plane

Using the ordering of the sets $i_j \prec i_s$ iff $j < s$, we can use the following sequence for automorphism testing:

$$\mathrm{Eq}(i_1,i_2,i_3), \mathrm{Eq}(i'_1,i_4,i_5), \mathrm{Eq}(i'_1, i_6, i_7), \mathrm{Eq}(i'_2, i'_4, i'_6), \mathrm{Eq}(i'_2, i'_5, i'_7), \mathrm{Eq}(i'_3, i''_4, i''_7), \mathrm{Eq}(i'_3, i''_5, i''_6),$$

where i'_j means that the interval i_j is purged of the codes that are equivalent to codes from preceding sets, i''_j means that the interval i'_j is purged of the codes that are equivalent to codes from preceding sets, and so on. As a result the reduced inequivalent set of codes will be the union $i'_1 \cup i'_2 \cup i'_3 \cup i'''_4 \cup i'''_5 \cup i'''_6 \cup i'''_7$.

5 Conclusions

In the present research we have considered the usability of the Gray code with constant weight words for computing linear combinations of codewords. We have shown that, in this way, a big improvement of the computation time for finding the minimum distance of a code can be achieved.

We have also considered the usefulness of combinatorial 2-$(t, k, 1)$ designs when there are memory limitations to the number of objects (linear codes in particular) that can be tested for equivalence. In our example we have shown that using the Fano plane one can achieve complete classification with as much as half of the computation time needed otherwise. It remains to find efficient designs for different number of sets to be checked for equivalence.

Acknowledgement

The authors express their gratitude to prof. Borislav Stoyanov for the invitation to publish in this journal. This work was supported by European Regional Development Fund and the Operational Program "Science and Education for Smart Growth" under contract UNITe No BG05M2OP001-1.001-0004-C01(2018-2023).

References

[1] R.T. Bilous (2006) Enumeration of the binary self-dual codes of length 34, *Journal of Combinatorial Mathematics and Combinatorial Computing*, 59, 173–211.

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

[3] J.R. Bitner, G. Ehrlich, E.M. Reingold (1975) Efficient Generation of the Binary Reflected Gray Code and Its Applications, *Commun. ACM*, 19(9), 517–521.

[4] I. Bouyukliev, M. Dzhumalieva-Stoeva, V. Monev (2015) Classification of Binary Self-Dual Codes of Length 40, *IEEE Transactions on Information Theory*, 61(8), 4253–4258.

[5] S. Bouyuklieva, I. Bouyukliev (2012) An Algorithm for Classification of Binary Self-Dual Codes, *IEEE Transactions on Information Theory*, 58(6), 3933–3940.

[6] C.J. Colbourn, J.H. Dinitz, *Handbook of Combinatorial Designs*, 2nd ed., CRC Press, 2010, ISBN 978-1-5848-8-5061.

[7] J.H Conway, V. Pless, N.J.A. Sloane (1992) The binary self-dual codes of length up to 32: A revised enumeration, *Journal of Combinatorial Theory, Series A*, 60(2), 183–195.

[8] F. Gray (1953) Pulse code communication, U.S. Patent 2,632,058, March 17, 1953

[9] M. Harada, A. Munemasa (2010) Classification of self-dual codes of length 36, *Advances in Mathematics of Communications*, 2, 229–235.

[10] W.C. Huffman, V.S. Pless (2003) *Fundamentals of Error-Correcting Codes*, Cambridge University Press, ISBN 978-0-5211-3-1704.

[11] V. Pless (1972) A classification of self-orthogonal codes over $GF(2)$, *Discrete Mathematics*, 3(1-3), 209–246.

[12] D.T. Tang, C.N. Liu (1973) Distance-2 cyclic chaining of constant-weight codes, *IEEE Transactions on Computers*, 2, 176–180.

[13] V. Tonchev (2017) On resolvable Steiner 2-designs and maximal arcs in projective planes, *Designs, Codes and Cryptography*, 84(1-2), pp 165–172.