A deterministic algorithm for the distance and weight distribution of binary nonlinear codes

E. Bellini*

DarkMatter LLC,
Abu Dhabi, 27655, UAE
Email: eemanuele.bellini@gmail.com
*Corresponding author

M. Sala

Department of Mathematics,
University of Trento,
Trento, TN, 38123, Italy
Email: maxsalacodes@gmail.com

Abstract: Given a binary nonlinear code, we provide a deterministic algorithm to compute its weight and distance distribution, and in particular, its minimum weight and its minimum distance, which takes advantage of fast Fourier techniques. This algorithm's performance is similar to that of best-known algorithms for the average case, while it is especially efficient for codes with low information rate. We provide complexity estimates for several cases of interest.

Keywords: distance distribution; minimum distance; weight distribution; minimum weight; nonlinear code.

Reference to this paper should be made as follows: Bellini, E. and Sala, M. (2018) 'A deterministic algorithm for the distance and weight distribution of binary nonlinear codes', *Int. J. Information and Coding Theory*, Vol. 5, No. 1, pp.18–35.

Biographical notes: E. Bellini received his BSc in 2006 and the MSc in 2009, both from the University of Turin, Italy, while he obtained his PhD in Mathematics in 2014, at the University of Trento. After working at Telsy Elettronica S.p.A., Turin, until January 2017, he moved to DarkMatter LLC in UAE. His research interests are in coding theory, especially nonlinear codes, and cryptography, especially elliptic curves and block cypher design.

M. Sala received the MSc in 1995 from the University of Pisa, Pisa, Italy, and the PhD degree in 2001 from the University of Milan, Milan, Italy. From 2002 to 2006 he has been a Senior Research Fellow at the Boole Research Centre in Informatics, University College Cork, Cork Ireland. He is currently a Full Professor in the Department of Mathematics, University of Trento, where he founded the Laboratory of Cryptography (CryptoLabTN), which he is still leading. His research interests are in computational algebra, algebraic coding theory, algebraic cryptography and their mutual interactions. Among his publications, we select the book 'Groebner bases, Coding and Cryptography', edited together with several

colleagues, containing an extensive survey on the current ongoing research. His recent industrial collaborations focus on security aspects for the banking system.

1 Introduction

Let C be a nonlinear code, that is, a code which is not necessarily linear. There are some related computational problems which are of interest, that we list as the computation of: the distance distribution (A), the minimum distance (A1), a minimum-distance codeword-pair (A2), the weight distribution (B), the minimum weight (B1), a minimum-weight codeword (B2). The decoding performance of C can be established by solving Problem A.

Remark 1: Solving Problem A2 (respectively, B2) implies solving Problem A1 (B1), but the converse does not hold. However, it is noteworthy that no known algorithm is able to solve A1 (B1) without solving A2 (B2).

If C is linear, Problem A (respectively, A1, A2) and B (B1, B2) are equivalents. This holds also for some nonlinear codes, called distance-invariant codes (Mitchell, 1989), and many of these are optimal codes (e.g., the Preparata and Kerdock codes (Preparata, 1968)). When C is linear, we consider also the decoding problem, which is implied by solving Problem B2 in the suitable code coset (which is a nonlinear code). Observe that the considerations in Remark 1 remain valid also if we restrict to linear codes.

To compute the minimum weight of a binary linear code, both deterministic and probabilistic algorithms are known. Example of the former is the Brouwer-Zimmermann algorithm (Zimmermann, 1996), or some recent improvements (Lisoněk and Trummer, 2015) and generalisations (Bouyukliev and Bakoev, 2008) of it. Example of the latter is given in Canteaut and Chabaud (1998), Leon (1988), or Stern (1989). Although several theoretical bounds for the distance are known (see e.g., Singleton (1964), Guerrini et al. (2016) and Bellini et al. (2014)), we remark that in this paper we are interested only in computational methods. We note that these algorithms must actually retrieve (at least) one minimum-weight codeword in order to obtain the minimum-weight value. Also recall that the following problems have been proved to be NP-hard in the case of binary nonlinear codes: computing the weight distribution (Berlekamp et al., 1978) and computing the minimum distance (Vardy, 1997). In the nonlinear case, the minimum weight and the minimum distance may be different. For some classes of nonlinear codes there are algorithms which perform much better than brute force, e.g., codes with a large kernel (Villanueva et al., 2014) or additive codes (White and Grassl, 2006). However, in the general nonlinear case, it is not possible to improve significantly on the brute force approach, as shown in Guerrini et al. (2010). Indeed, we are not aware of any non-exponential probabilistic or deterministic algorithm to solve any of the problems A, A1, A2, B, B1, B2.

In particular, to compute the weight distribution of a generic binary $(n, 2^k)$-nonlinear code given as a list of binary vectors, we need to perform $O(n2^k)$ bit operations, while finding the distance distribution requires $O(n2^{2k})$ bit operations. These complexity estimates depend heavily on the way the code is presented as input to the relevant algorithms.

The main result of this paper is a *deterministic* algorithm to compute the distance and weight distribution, and thus the minimum distance and the minimum weight, of any random binary code *represented* as a set of Boolean functions in numerical normal form (NNF).

Our method performs better than brute force for those codes with low information rate and sparse NNF representation, while in the general case, it achieves the same asymptotic computational complexity as brute force methods.
In Section 2, after some preliminaries on Boolean functions, we argue that representing a code as a set of Boolean functions in NNF does not have any particular drawback with respect to the classical representation of a code as a set of binary vectors. In Section 3, to each binary code we associate a polynomial whose evaluations are the weights of the code. Similarly, in Section 4, to each binary code we associate a polynomial whose evaluations are the distances of all possible pairs of codewords. Given these two polynomials we are able to compute the weight and the distance distribution of any binary nonlinear code. Finally, in Section 5 we provide some complexity considerations regarding our algorithms. In particular, we show that, to compute the weight distribution starting from the NNF representation of a binary nonlinear code has a complexity of $O((n/h + k)2^k)$, where n/h is the average number of nonzero monomials of the Boolean functions representing the code. Moreover, there are cases where our approach is provably faster than brute force (e.g., in the nonlinear case when the NNF representation of the code is sparse), and cases where it is experimentally faster than the Brouwer-Zimmermann method.

2 Preliminaries

We denote by $\mathbb{F}$ the field $\mathbb{F}_2$. The set $\mathbb{F}^r$ is the set of all binary vectors of length r, viewed as an $\mathbb{F}$-vector space.
Let $n \geq k \geq 1$ be integers. Let $C \subseteq \mathbb{F}^n$, $C \neq \emptyset$. We say that C is a binary $(n, |C|)$-*code* (since we only deal with binary codes, we often omit the word binary). Any $c \in C$ is a *codeword*. If C is a vector subspace of dimension k of $\mathbb{F}^n$, then C is a binary $[n, k]$-*linear code*.
Let $v \in \mathbb{F}^n$. The *Hamming weight* $\mathrm{w}(v)$ of the vector v is the number of its nonzero coordinates. For any two vectors $v_1, v_2 \in \mathbb{F}^n$, the *Hamming distance* between v_1 and v_2, denoted by $\mathrm{d}(v_1, v_2)$, is the number of coordinates in which the two vectors differ.
We call *minimum weight* of a code C the integer $w = \min\{\mathrm{w}(c) \mid c \in C\}$, and *(minimum) distance* of a code C the integer $d = \min\{\mathrm{d}(c, c') \mid c, c' \in C, c \neq c'\}$. If C is an $(n, |C|)$-code with distance d then we can write that C is an $(n, |C|, d)$-code.
Finally, we call *weight distribution* of the code C the sequence of numbers $A_t = |\{c \in C \mid \mathrm{w}(c) = t\}|$ and *distance distribution* of an $(n, |C|)$-code C the sequence of numbers $A_t = \frac{1}{|C|}|\{(c, c') \in C \times C \mid \mathrm{d}(c, c') = t\}|$

2.1 Representations of Boolean functions

In this section we briefly summarise some definitions and known results from Carlet (2010) and MacWilliams and Sloane (1977), concerning representations of Boolean functions.

A *Boolean function* (B.f.) is a function $f : \mathbb{F}^k \to \mathbb{F}$. The set of all Boolean functions from $\mathbb{F}^k$ to $\mathbb{F}$ will be denoted by $\mathcal{B}_k$. There are several ways one can uniquely represent a B.f. . We briefly outline those we need.

2.1.1 Evaluation vector

We assume to have ordered $\mathbb{F}^k$, so that $\mathbb{F}^k = \{\mathsf{p}_1, \ldots, \mathsf{p}_{2^k}\}$. A Boolean function f can be specified by a *truth table*, which gives the evaluation of f at all p_i's. We consider the evaluation map:

$$\mathcal{B}_k \longrightarrow \mathbb{F}^{2^k} \qquad f \longmapsto \underline{f} = (f(\mathsf{p}_1), \ldots, f(\mathsf{p}_{2^k})).$$

The vector $\underline{f}$ is called the *evaluation vector* of f. Once the order on $\mathbb{F}^k$ is chosen, i.e., the p_i's are fixed, it is clear that the evaluation vector of f identifies f.

2.1.2 Algebraic normal form

A Boolean function $f \in \mathcal{B}_k$ can be expressed in a unique way as a square-free polynomial in $\mathbb{F}[X] = \mathbb{F}[x_1, \ldots, x_k]$, i.e.,

$$f(x_1, \ldots, x_k) = \sum_{v \in \mathbb{F}^k} b_v X^v,$$

where $b_u \in \mathbb{F}$ and $v = (v_1, \ldots, v_k)$, and $X^v = x_1^{v_1} \cdots x_k^{v_k}$.
This representation is called the *Algebraic Normal Form* (ANF).
There exists a simple divide-and-conquer butterfly algorithm (Carlet, 2010), p.10) to compute the ANF from the truth table (or vice-versa) of a Boolean function, which requires $k2^{k-1}$ bit sums (i.e., complexity $O(k2^k)$), while $O(2^k)$ bits must be stored. This algorithm is known as the *fast Möbius transform*.

2.1.3 Numerical normal form

In Carlet and Guillot (1999) (see also Carlet and Guillot (2001), Carlet (2002)) the following representation of Boolean functions has been introduced.
Let f be a function on $\mathbb{F}^k$ taking values in a field $\mathbb{K}$. We call the *numerical normal form (NNF)* of f the following expression of f as a polynomial:

$$f(x_1, \ldots, x_k) = \sum_{u \in \mathbb{F}^k} \lambda_u (\prod_{i=1}^{k} x_i^{u_i}) = \sum_{u \in \mathbb{F}^k} \lambda_u X^u,$$

with $\lambda_u \in \mathbb{K}$ and $u = (u_1, \ldots, u_k)$. This representation of f considers f as a mapping from the set $\{0,1\}^k$ to the set $\{0,1\}$, where 0,1 are the in $\mathbb{K}$. Clearly, such a mapping can be represented by a polynomial via multivariate interpolation.

It can be proved (Carlet and Guillot, 1999), Proposition 1) that any Boolean function f admits a unique NNF. As for the ANF, it is possible to compute the NNF of a Boolean function from its truth table by mean of an algorithm similar to a fast Fourier transform, thus requiring $O(k2^k)$ additions over $\mathbb{K}$ and storing $O(2^k)$ elements of $\mathbb{K}$.

From now on let $\mathbb{K} = \mathbb{Q}$.
The truth table of f can be recovered from its NNF by the formula

$$f(u) = \sum_{a \in \mathbb{F}^k | a \preceq u} \lambda_a, \forall u \in \mathbb{F}^k,$$

where $a \preceq u \iff \forall i \in \{1, \ldots, k\}\ a_i \leq u_i$. Conversely, as shown in Carlet and Guillot (1999) (Section 3.1), it is possible to derive an explicit formula for the coefficients of the NNF by means of the truth table of f.

Proposition 1: *Let f be any integer-valued function on $\mathbb{F}^k$. For every $u \in \mathbb{F}^k$, the coefficient λ_u of the monomial X^u in the NNF of f is*

$$\lambda_u = (-1)^{\mathrm{w}(u)} \sum_{a \in \mathbb{F}^k | a \preceq u} (-1)^{\mathrm{w}(a)} f(a). \tag{1}$$

It is possible to convert a Boolean function from NNF to ANF simply by reducing its coefficients modulo 2. The inverse process is less trivial. One can either apply Proposition 1 to the evaluation vector of f or apply recursively the fact that

$$a +_{\mathbb{F}} b = a +_{\mathbb{Z}} b +_{\mathbb{Z}} (-2ab), \tag{2}$$

and the fact that each variable has to be square-free (we are working in the affine algebra $\mathbb{Q}[x_1, \cdots, x_k]/\langle x_1^2 - x_1, \cdots, x_k^2 - x_k \rangle$).
From Equation 2 we can derive recursively

$$a_1 +_{\mathbb{F}} \ldots +_{\mathbb{F}} a_m = \sum_{i=1}^{m} (-1)^{m-1} 2^{m-1} \sum_{\substack{(e_1,\ldots,e_m) \in \mathbb{F}^m, \\ \mathrm{w}((e_1,\ldots,e_m))=i}} a_1^{e_1} \cdot \ldots \cdot a_m^{e_m} \tag{3}$$

where the summations on the left side are over the integers.

2.2 Representing a code as a set of Boolean functions

Recall that we only consider binary codes, i.e., codes over the finite field $\mathbb{F}$ of length n, with M codewords.

Now we show that any binary $(n, 2^k)$-code C can be represented in a unique way as a set of n Boolean functions $f_1, \ldots, f_n : \mathbb{F}^k \to \mathbb{F}$. We indicate with $f^{(\mathbb{F})}$ a Boolean function represented in algebraic normal form, and with $f^{(\mathbb{Z})}$ a Boolean function represented in NNF.

Definition 1: Given a binary $(n, 2^k)$-code C, consider a fixed order of the codewords of C and of the vectors of $\mathbb{F}^k$. Then consider the matrix M whose rows are the codewords of C. We call the *defining polynomials* of the code C the set $\mathcal{F}_C = \{f_1, \ldots, f_n\}$ of the uniquely determined Boolean functions whose truth table are the columns of M. We also indicate with $F = (f_1^{(\mathbb{F})}, \ldots, f_n^{(\mathbb{F})}) \in \mathbb{F}[X]^n$, where $X = x_1, \ldots, x_k$, the polynomial vector whose components are the defining polynomials of C in ANF, and with $\bar{F} = (f_1^{(\mathbb{Z})}, \ldots, f_n^{(\mathbb{Z})}) \in \mathbb{K}[X]^n$ the polynomial vector whose components are the defining polynomials of C in NNF. With abuse of notation, we sometimes write

$$\mathcal{F}_C = \{f_1^{(\mathbb{F})}, \ldots, f_n^{(\mathbb{F})}\} \text{ or } \mathcal{F}_C = \{f_1^{(\mathbb{Z})}, \ldots, f_n^{(\mathbb{Z})}\}.$$

Notice that F can be seen as an encoding function, since $F : \mathbb{F}^k \to \mathbb{F}^n$.

Therefore, there is a bijection between the points in $\mathbb{F}^k$ and the codewords of C, given by F. Moreover, this bijection extends to a bijection between the points in $\{0, 1\}^k$ and the codewords of C, given by $\bar{F}$.

2.2.1 Memory cost of representing a code

Let us call *vectorial* the representation of a code as a list of vectors over $\mathbb{F}$, and *Boolean* the representation of the same code as a list of Boolean functions.
For a random code, in terms of memory cost, the two representations are equivalent. In the vectorial representation, we need to store all the components of each codeword, which are n times 2^k codewords. In the Boolean representation, we need to store the 2^k coefficients of the n defining polynomials. In both cases, we need a memory space of order $O(n2^k)$.
If the code C is linear it can be represented with a binary generator matrix of size $k \times n$. In this case, the defining polynomials are linear Boolean functions, i.e., any is of the form $\sum_{i=1}^{k} \lambda_i x_i, \lambda_i \in \mathbb{F}$, which means that to represent them it is sufficient to store kn elements of $\mathbb{F}$, yielding again an equivalent representation.

Recall the *kernel* of a binary code C is defined as $\ker(C) = \{x \in \mathbb{F}^n \mid x + C = C\}$. If the all-zero vector is in C, as we can always suppose without loss of generality, then $K = \ker(C)$ is a binary linear sub-code of C. Let us denote with k_K the dimension of K. Then the code C can be written as union of cosets of K, i.e., $C = \bigcup_{i=0}^{t} (K + c_i)$, where $c_0 = (0, \ldots, 0), t + 1 = |C|/2^k$ and $c_1, \ldots, c_t$ are representatives of the cosets of K. Note that the representatives need not to be the ones having minimum weight in their respective cosets.

As shown in Pujol et al. (2012); Villanueva et al. (2014), since the kernel needs a memory space of order $O(nk_K)$, then the kernel plus the t coset representatives takes up a memory space of order $O(n(k_K + t))$. When C is linear then $C = \ker(C)$, so the generator matrix is used to represent C. On the other hand, when $t + 1 = |C|$, then representing the code as the kernel plus the coset representatives requires a memory of $O(n|C|) = O(n2^k)$ (since we are supposing the code has 2^k codewords). In the latter case, a Boolean representation can be equivalent or even more convenient, especially if the Boolean polynomials representing the code are sparse. Notice that the Boolean representation depends on how the codewords are ordered, and so, it is possible to choose the most convenient ordering to obtain the sparser representation.

It is worth noticing that a linear structure of a nonlinear binary code can be found over a different ring. For example, there are binary codes which have a $\mathbb{Z}_4$-linear or $\mathbb{Z}_2\mathbb{Z}_4$-linear structure and, therefore, they can also be compactly represented using quaternary generator matrix, as shown in Hammons et al. (1994) and Borges et al. (2010).

2.2.2 Number of coefficients of the NNF

In this section, we show that, when k is small, i.e., for codes which can be used in practice, the Boolean representation of a code using ANF polynomials is as convenient as the one using or NNF polynomials.

In particular, we study the distribution of the number of nonzero coefficients of a B.f. represented in NNF, i.e., once the number of variables k is fixed we want to know how many B.f. 's have only one nonzero coefficient, how many have two, and so on.

We are also interested in finding a relation between this distribution and the distribution of the number of nonzero coefficients of a B.f. represented in ANF.

In Table 1, we report the distribution of the nonzero coefficients of B.f. 's represented in ANF and NNF with $k = 1, 2, 3, 4$ variables. As one may expect, the ANF follows a binomial distribution. This means that choosing a random B.f. its ANF is likely to have half of the

coefficients equal to 0 and a half equal to 1. This does not happen for the NNF, although for k small the two distributions are close. This means that, when k is small, a random binary $(n, 2^k)$ code can be represented with a set of B.f. 's in NNF with half of the coefficients equal to 0 with high probability, while sparse NNF representations are rarer as k grows.

Proposition 2: *Let f be a B.f. in k variables. Let $f^{(\mathbb{F})}$ and $f^{(\mathbb{Z})}$ be respectively the ANF and the NNF of f. Then if $f^{(\mathbb{F})}$ is a polynomial with $r \leq 2^k$ nonzero coefficients, then $f^{(\mathbb{Z})}$ is a polynomial with no more than $\min\{2^k, 2^r - 1\}$ nonzero coefficients.*

Proof: When computing the NNF from the ANF we have again the r initial terms of the ANF, plus $\binom{r}{2}$ terms which are all possible double product of the r initial terms, plus, in general, $\binom{r}{i}$ terms which are all possible i-product of the r initial terms, for each $i \in \{1, \ldots, r\}$. Thus we will have

$$\sum_{i=1}^{r} \binom{r}{i} = 2^r - 1 \tag{4}$$

terms to be summed together. If no sum of similar monomials becomes zero then we have $2^r - 1$ nonzero terms. □

By Proposition 2, if we want an NNF with no more than s terms, then we have to choose the ANF with no more than $r = \log_2(s + 1)$ terms.

Proposition 3: *Let f be a linear B.f. in k variables with r nonzero coefficients. Let $f^{(\mathbb{F})}$ and $f^{(\mathbb{Z})}$ be respectively the ANF and the NNF of f. Then, for $r \leq k$ and for some $i_1 < i_2 < \ldots < i_r$,*

$$f^{(\mathbb{F})} = x_{i_1} + \ldots + x_{i_r}.$$

Moreover, $f^{(\mathbb{Z})}$ is a polynomial with exactly $2^r - 1$ nonzero coefficients:

$$f^{(\mathbb{Z})} = \sum_{i=1}^{r} (-1)^{r-1} 2^{r-1} \sum_{\substack{(v_1,\ldots,v_r)\in\mathbb{F}^r, \\ \mathrm{w}((v_1,\ldots,v_r))=i}} v_1^{v_1} \cdot \ldots \cdot x_r^{v_r} \tag{5}$$

Proof: Equation 5 is the same as Equation 3.
For each i there are $\binom{r}{i}$ nonzero terms in the inner summation, so the total of nonzero terms is $\sum_{i=1}^{r} \binom{r}{i} = 2^r - 1$. □

Proposition 3 shows that for a linear B.f. , its NNF representation is exponentially denser than its ANF representation.

Table 1 Distribution of the nonzero coefficients in the ANF and NNF

k	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*
A: 1	1	2	1	–	–	–	–	–	–	–	–	–	–	–	–	–	–
N: 1	1	2	1	–	–	–	–	–	–	–	–	–	–	–	–	–	–
A: 2	1	4	*6*	4	1	–	–	–	–	–	–	–	–	–	–	–	–
N: 2	1	4	*5*	4	2	–	–	–	–	–	–	–	–	–	–	–	–
A: 3	1	8	28	56	*70*	56	28	8	1	–	–	–	–	–	–	–	–
N: 3	1	8	19	42	*59*	50	34	28	15	–	–	–	–	–	–	–	–
A: 4	1	16	120	560	1820	4368	8008	11440	*12870*	11440	8008	4368	1820	560	120	16	1
N: 4	1	16	65	304	840	1768	3250	5458	8077	*9986*	9819	7948	5954	4458	3193	2830	1569

3 Finding the codewords with weight exactly t

It is possible to construct a polynomial with integer coefficients whose evaluations in $\{0,1\}^k \subseteq \mathbb{Z}^k$ are the weights of the codewords of the code C.

Definition 2: Let $X = \{x_1, \ldots, x_k\}$, and $X^2 - X = \{x_1^2 - x_1, \ldots, x_k^2 - x_k\}$. We call the **weight polynomial** of the code C the polynomial

$$\mathfrak{w}_C(X) = \sum_{i=1}^{n} f_i^{(\mathbb{Z})}(X) \in \mathbb{Z}[X]/\langle X^2 - X \rangle,$$

where the $f_i^{(\mathbb{Z})}$'s are the defining polynomials of the code C in NNF.

Theorem 1: *Let $v \in \{0,1\}^k \subseteq \mathbb{Z}^k$. The two sets $\{\mathrm{w}(c) \mid c \in C\}$ and $\{\mathfrak{w}_C(P) \mid P \in \{0,1\}^k\}$ are equal.*

Proof: From the discussion following Definition (1), it is clear that for any $c \in C$ there is one and only one $P \in \{0,1\}^k$ such that $c = (f_1^{(\mathbb{Z})}(P), \ldots, f_n^{(\mathbb{Z})}(P))$. Our claim follows from observing that the sum of all $f_i^{(\mathbb{Z})}$ is over the integers and $f_i^{(\mathbb{Z})}(P) \geq 0$, for $i = 1, \ldots, n$. □

Once we have the weight polynomial $\mathfrak{w}_C$ of the code C, not only we can find the minimum weight of C, but we also find which are the codewords having certain weights by looking at its evaluation vector over the set $\{0,1\}^k$. As we will see in Section 5.4, computing this evaluation has a cost of $O(k2^k)$.

We summarise in Algorithm 1 the steps to obtain the weight distribution of a binary $(n, 2^k)$-code C given as a list of 2^k codewords (and thus also the minimum weight of C), by finding the evaluation vector of the weight polynomial $\mathfrak{w}_C$. We indicate with $C_{i,j}$ the jth component of the ith word of C, with $1 \leq j \leq n$ and $1 \leq i \leq 2^k$.

Algorithm 1 To find the weight distribution $\underline{\mathfrak{w}}_C$ of a binary nonlinear code C.

Require: $c_1, \ldots, c_{2^k} \in C$
Ensure: the evaluation vector $\underline{\mathfrak{w}}_C$ of $\mathfrak{w}_C$
1: $f_j^{(\mathbb{Z})} \leftarrow$ NNF of the binary vector $(C_{1,j}, \ldots, C_{2^k,j})$ for $1 \leq j \leq n$
2: $\mathfrak{w}_C \leftarrow f_1^{(\mathbb{Z})} + \ldots + f_n^{(\mathbb{Z})}$
3: $\underline{\mathfrak{w}}_C \leftarrow$ Evaluation of $\mathfrak{w}_C$ over $\{0,1\}^k$
4: **return** $\underline{\mathfrak{w}}_C$

4 Finding pairs of codewords with distance exactly t

It is straightforward to adapt the techniques in Section 3 to the computation of the distance distribution of a code C.

First, we show how to construct a polynomial with integer coefficients whose evaluations in $\{0,1\}^{2k} \subseteq \mathbb{Z}^{2k}$ are the distances of all possible pairs of codewords of the code C.

Definition 3: Let $X = x_1, \ldots, x_k$, $\tilde{X} = \tilde{x_1}, \ldots, \tilde{x_k}$, and $X^2 - X = x_1^2 - x_1, \ldots, x_k^2 - x_k$, $\tilde{X}^2 - \tilde{X} = \tilde{x_1}^2 - \tilde{x_1}, \ldots, \tilde{x_k}^2 - \tilde{x_k}$.
We call the **distance polynomial** of the code C the polynomial

$$\begin{aligned}\mathfrak{d}_C(X, \tilde{X}) &= \sum_{i=1}^{n} (f_i^{(\mathbb{Z})}(X) - f_i^{(\mathbb{Z})}(\tilde{X}))^2 \\ &\in \mathbb{Z}[X, \tilde{X}]/\langle X^2 - X, \tilde{X}^2 - \tilde{X}\rangle,\end{aligned}$$

where the $f_i^{(\mathbb{Z})}$'s are the defining polynomials of the code C in NNF.

Notice that the squaring operation does not introduce squared variables in the expression of $\mathfrak{d}_C$, because we are working in the quotient ring $\mathbb{Z}[X, \tilde{X}]/\langle X^2 - X, \tilde{X}^2 - \tilde{X}\rangle$.
Notice also that, for $v = (v_1, \ldots, v_k, v_{k+i}, \ldots, v_{2k}) \in \{0,1\}^{2k}$, we have that $\mathfrak{d}_C((v_1, \ldots, v_k, v_{k+1}, \ldots, v_{2k})) = 0$ if and only if $v_i = v_{k+1}$ for $i = 1, \ldots, k$, and that $\mathfrak{d}_C((v_1, \ldots, v_k, v_{k+1}, \ldots, v_{2k})) = \mathfrak{d}_C((v_{k+1}, \ldots, v_{2k}, v_1, \ldots, v_k))$.

Theorem 2: *The two sets* $\{\mathrm{d}(c_1, c_2) \mid c_1, c_2 \in C, c_1 \neq c_2\}$ *and* $\{\mathfrak{d}_C(P, Q) \mid P, Q \in \{0,1\}^k, P \neq Q\}$ *are equal.*

Proof: Note that $\forall c_1, c_2 \in C, c_1 \neq c_2$ we have that $c_1 - c_2 = ((f_1^{(\mathbb{Z})}(P) - f_1^{(\mathbb{Z})}(Q))^2, \ldots, (f_n^{(\mathbb{Z})}(P) - f_n^{(\mathbb{Z})}(Q))^2) \in \{0,1\}^n$, for some $P, Q \in \{0,1\}^k, P \neq Q$. The squaring operation is needed in order to correct those components which have become a -1 after the subtraction operation. Finally, the sum of all $(f_i^{(\mathbb{Z})}(X) - f_i^{(\mathbb{Z})}(\tilde{X}))^2$ is over the integers. □

We summarise in Algorithm 2 the steps to obtain the distance distribution of a binary $(n, 2^k)$-code C given as a list of 2^k codewords (and thus also the minimum distance of C), by finding the evaluation vector of the distance polynomial $\mathfrak{d}_C$. We indicate with $C_{i,j}$ the jth component of the ith word of C, with $1 \leq j \leq n$ and $1 \leq i \leq 2^k$.

Algorithm 2 To find the distance distribution $\underline{\mathfrak{d}}_C$ of a binary nonlinear code C.

Require: $c_1, \ldots, c_{2^k} \in C$
Ensure: the evaluation vector $\underline{\mathfrak{d}}_C$ of $\mathfrak{d}_C$
1: $f_j^{(\mathbb{Z})} \leftarrow$ NNF of the binary vector $(C_{1,j}, \ldots, C_{2^k,j})$ for $1 \leq j \leq n$
2: $\mathfrak{d}_C \leftarrow (f_1^{(\mathbb{Z})}(X) - f_1^{(\mathbb{Z})}(\tilde{X}))^2 + \ldots + (f_n^{(\mathbb{Z})}(X) - f_n^{(\mathbb{Z})}(\tilde{X}))^2$
3: $\underline{\mathfrak{d}}_C \leftarrow$ Evaluation of $\mathfrak{d}_C$ over $\{0,1\}^{2k}$
4: **return** $\underline{\mathfrak{d}}_C$

5 Complexity considerations

First of all let us notice that given a binary $(n, 2^k)$-code as a list of 2^k codewords, to find the weight distribution of a binary nonlinear code C using brute force requires $n2^k$ bit

operations, since we have to check each component of each codeword of C. Similarly, to find the distance distribution, $n2^{2k}$ operations are needed.

We note that the operations involved in our following complexity estimates are over the integers, but the size of the integers involved in our operations, i.e., the nonzero coefficients of the NNF of a Boolean function, is limited by 2^k, as shown by equation (3). Moreover, in the cases of our interest, i.e., for sparse Boolean functions, these coefficients have a sparse binary representation, and in particular, when the Boolean function is linear, by Proposition 3 we have that the coefficients are precisely powers of 2.

We now analyse the complexity of Steps 1–3 of Algorithms 1 and 2. Then, due to the similarities of the two algorithms, we only concentrate on the first one. We compare our method to compute the minimum weight of a binary code with brute force and, in the linear case, with the Brouwer-Zimmermann method (Zimmermann, 1996). We provide more emphasis on the comparison in the linear case since no other methods than brute force are known in the nonlinear case, with the only exception of Pujol et al. (2012); Villanueva et al. (2014). We do not provide a computational comparison with these works since their methods are known to perform better than brute force only for codes with a large kernel.

5.1 From the list of codewords to defining polynomials in NNF

Proposition 4: *The overall worst-case complexity of determining the coefficients of the n defining polynomials in NNF of the code C given as a list of vectors is $O(nk2^k)$.*

Proof: We want to find the NNF of the Boolean function whose truth table is given by a column of the binary matrix whose rows are the codewords of the code C. In (Carlet and Guillot, 1999, Proposition 2) it is shown that to compute the NNF of a Boolean function in k variables given its truth table requires $k2^{k-1}$ integer subtractions. Since we have to compute the NNF for n columns the overall complexity is $O(nk2^k)$. □

5.2 From defining polynomials to weight polynomial

Proposition 5: *The overall worst-case complexity of summing together all the defining polynomials in NNF is $O(n2^k)$.*

Proof: Each monomial m in a defining polynomial is square-free, and since $m \in \mathbb{Z}[x_1, \ldots, x_k]$, then a defining polynomial can have no more than 2^k monomials. Since the defining polynomials are n, the proposition follows. □

Remark 2: Clearly, the computational complexity of this steps decreases if the defining polynomials are sparse when considering their NNF.

5.3 From defining polynomials to distance polynomial

Proposition 6: *The overall worst-case complexity of Step 2 of Algorithm 2 is $O(n2^{2k})$.*

Proof: The sum $\hat{f}_i = f_i^{(\mathbb{Z})}(X) - f_i^{(\mathbb{Z})}(\tilde{X})$ for $i = 1, \ldots, n$ is just a concatenation of coefficients, where the coefficients of $f_i^{(\mathbb{Z})}(\tilde{X})$ need to have their sign switched.
The polynomial obtained has 2^{k+1} terms in the worst-case, and squaring it requires $2^{2(k+1)}$ integer multiplications and the same number of integer sums, for a total of 2^{2k+3} integer

operations. Since we have n such polynomials $\hat{f}_i$, to compute their square requires $n2^{2k+3}$ integer operations. Each $\hat{f}_i^2$ has at most 2^{2k} terms, since $\hat{f}_i \in \mathbb{Z}[X, \tilde{X}]/\langle X^2 - X, \tilde{X}^2 - \tilde{X}\rangle$. Summing all $\hat{f}_i^2$ together thus requires at most $n2^{2k}$ integer sums. The overall worst-case complexity of Step 2 of Algorithm 2 is then

$$n2^{2k+3} + n2^{2k} = n2^{2k}(2^3 + 1).$$

□

Remark 3: Again, the complexity of this step is lower if the defining polynomials are sparse in their NNF. If, for example, the nonzero coefficients of $f_i^{(\mathbb{Z})}(X)$ are $\sim k$, so are the coefficients of $f_i^{(\mathbb{Z})}(\tilde{X})$, and the squaring of $\hat{f}_i$ requires $\sim (2k)^2$ integer operations.

5.4 Evaluation of the weight and the distance polynomial

Algorithm 3 describes the fast Möbius transform to compute the evaluation vector of a Boolean function f in NNF in k variables. We use the following notation: the coefficient c_{2^k} is the coefficient of the greatest monomial, i.e., of $x_1 \cdots x_k$, c_{2^k-1} the coefficient of the second greatest monomial, and so on until c_1, which is the constant term. We provide Example 1 to clarify our notation.

Notice that the sum in Step 6 is over our integers. If it was a sum in $\mathbb{F}$ then we would obtain the truth table of f.

Algorithm 3 Fast Möbius transform for fast integer polynomial evaluation.

Require: vector of coefficients $c = (c_1, \ldots, c_{2^k})$
Ensure: evaluation vector $e = (e_1, \ldots, e_{2^k})$
1: $e \leftarrow c$
2: **for** $i = 0, \ldots, k$ **do**
3: $\quad b \leftarrow 0$
4: $\quad$ **repeat**
5: $\quad\quad$ **for** $x = b, \ldots, b + 2^i - 1$ **do**
6: $\quad\quad\quad e_{x+1+2^i} \leftarrow e_{x+1} + e_{x+1+2^i}$
7: $\quad\quad b \leftarrow b + 2^{i+1}$
8: $\quad$ **until** $b = 2^k$
9: **return** e

Example 1: Consider $k = 3$ and lexicographical ordering with $x_1 \succ x_2 \succ x_3$. Let $f = 8x_1x_2x_3 + 3x_1 + 2$. Then $c = (c_1, \ldots, c_8) = (2, 0, 0, 0, 3, 0, 0, 8)$ and $e = (e_1, \ldots, e_8) = (2, 2, 2, 2, 5, 5, 5, 13)$.

Proposition 7: *Evaluating the weight polynomial over the set* $\{0, 1\}^k$ *has a computational cost of* $O(k2^k)$.

Proof: This is the cost of Algorithm 3, i.e., $k2^{k-1}$ integer sums. □

Similarly, we can derive the following estimate.

Proposition 8: *Evaluating the distance polynomial over the set* $\{0,1\}^{2k}$ *has a computational cost of* $O(k2^{2k})$.

5.5 Comparison with brute force method

Because of the similarities of Algorithms 1 and 2, we now concentrate our analysis only on Algorithm 1. All considerations we expose can be easily extended for Algorithm 2.

Theorem 3: *Let* h *be a positive integer. If the code* C *is given as a set of B.f. 's whose NNF have on average* $2^k/h$ *coefficients different from* 0*, then computing the minimum weight of* C *requires at most*

$$\left(\frac{n}{h}+\frac{k}{2}\right)2^k.$$

integer sums. Thus the complexity is

$$O\left(\left(\frac{n}{h}+k\right)2^k\right).$$

Proof: By Proposition 8 computing the evaluation vector of the weight polynomial $\mathfrak{w}_C$ requires $k2^{k-1}$ integer sums using the fast Möbius transform. To compute the weight polynomial we need to sum the n defining polynomials $f_i^{(\mathbb{Z})}, i=1,\ldots,n$, in NNF. If each of these polynomials has on average $2^k/h$ coefficients then the complexity of computing $\mathfrak{w}_C$ requires $O(n\frac{2^k}{h})$ integer sums. So the final complexity is at most $(n/h)2^k + k2^{k-1}$. □

Remark 4: Our method is more efficient than brute force when $n/h + k < n$. This is very likely to happen for a random code of low information rate where $k \ll n$. If $k \sim n$ and the NNF is dense, then it is convenient to use brute force rather than our method.

Notice also that if the sets of nonzero monomials of two polynomials in NNF are disjoint, then the sum of the two polynomials is simply their concatenation. So, if the defining polynomials of a code are "disjoint", then the cost of computing the weight polynomial is $O(1)$, and the final cost of finding the minimum weight becomes the cost of computing the evaluation of $\mathfrak{w}_C$, i.e., $O(k2^k)$.

Fact 1 shows that, for $n \gg k$, when the code is linear our method to compute the minimum nonzero weight (i.e., the distance of the code) given the set of the defining polynomials in NNF is more efficient than the classical method which uses brute force, given the list of the codewords of the code.

Fact 1 (Comparison with brute force, linear case, $n \sim 2^k$**):**

Consider a random binary $[n,k]$-linear code C such that $n \sim 2^k$. Then computing the weight distribution of C

- given the list of its codewords and using brute force requires $O(2^{2k})$.
- given the list of the defining polynomials in NNF and finding the minimum of $\mathfrak{w}_C$ requires $O(2^{\frac{3}{2}k})$.

Proof: The complexity of finding the weight distribution of C in case 1 is $O(n2^k) = O(2^{2k})$, since $n \sim 2^k$.
The complexity of finding the weight distribution of C in case 2 is $O((n/h + k)2^k)$ (by Theorem 3), where n/h is the average number of nonzero coefficients of the NNF. If the linear code C is random, then so are the random linear defining polynomials. A random linear function in k variables has on average $k/2$ nonzero coefficient in ANF and thus $2^{k/2} - 1$ nonzero coefficients in NNF, i.e., $n/h \sim 2^{k/2}$, and

$$O((n/h + k)2^k) = O((2^{k/2} + k)2^k) = O(2^{\frac{3}{2}k}).$$

□

Fact 2 (Comparison with brute force, nonlinear case, $n \sim 2^k$**):** Consider a random binary $(n, 2^k)$-nonlinear code C such that $n \sim 2^k$, and whose defining polynomials have on average $k/2$ nonzero coefficients in the ANF. Then computing the weight distribution of C given the list of the defining polynomials in NNF and finding the minimum of $\mathfrak{w}_C$ requires at most $O(2^{\frac{3}{2}k})$.

Proof: The arguments are the same as in the proof of Fact 1, except that this time the nonzero coefficients of the NNF are less than $2^{k/2} - 1$. This implies that in practice the overall complexity, in this case, is even lower, as shown in Table 2. □

Given two codes C_1, C_2 we can consider the time t_1, t_2 to compute the minimum weight of each code. We call *coefficient of growth* (of C_2 with respect to C_1) of an algorithm the number $\log_2(t_2/t_1)$.

In Table 2, we show the coefficient of growth of the complexity of our method in three different cases. The first line shows the coefficient of growth of the brute force method applied to a generic code, while the second line refers to linear codes. The third line shows the coefficient of growth of our method applied to a linear code. In the fourth line our method is applied to a nonlinear code whose ANF representation is sparse, and in the last line nonlinear codes with dense ANF representation are considered.

For the comparison we chose for each k, $10,000$ random $(2^k, 2^k)$-codes and $10,000$ random $(2^{k+1}, 2^{k+1})$-codes and compute the average number of clock cycles t_1, t_2 to compute the minimum weight in each case. Then we report the number $\log_2(t_2/t_1)$.

We can see, as expected, that our method performs best in the case of sparse nonlinear ANF.

An Intel Core i7 CPU 920 processor at 2.67 GHz has been used for the computations.

Table 2 Coefficients of growth of our method compared with brute force

Method	*Code*	*Representation*	*k: 2-3*	*k: 3-4*	*k: 4-5*	*k: 5-6*	*k: 6-7*	*k: 7-8*	*k: 8-9*	*k: 9-10*
Brute force	Nonlinear	any	0.89	1.30	1.66	1.77	1.91	1.92	1.98	1.99
Brute force	Linear	Sparse ANF	0.87	1.28	1.59	1.81	1.91	1.95	1.96	2.01
Our method	Linear	Sparse ANF	0.32	0.60	0.82	1.03	1.13	1.24	1.36	1.49
Our method	Nonlinear	Sparse ANF	0.68	0.77	1.02	1.18	1.20	1.26	1.29	1.31
Our method	Nonlinear	Dense ANF	0.65	0.99	1.21	1.48	1.77	1.99	2.04	2.03

5.6 Comparison with Brouwer-Zimmermann method for linear codes

We now provide an experimental comparison of our method with brute force and Brouwer-Zimmermann methods to find the minimum weight of a linear code.

In Tables 3–5, different values of (n, k) and samples of 10,000 codes have been considered. For each (n, k)-pair, let us call

- t_1, the number of clock cycles needed to compute the minimum weight using MAGMA implementation of our method
- t_2, the number of clock cycles needed to compute the minimum weight using MAGMA command

```
MinimumWeight(C:Method:="Distribution");
```

- t_3, the number of clock cycles needed to compute the minimum weight using MAGMA command

```
MinimumWeight(C:Method:="Zimmermann").
```

Note that our method is applied to random linear codes represented as Boolean functions in NNF, while Brouwer-Zimmermann and brute force are applied to random linear code represented by their generator matrix.

In Table 3, the ratio t_1/t_2 is reported for random linear codes with low information rate ($n \sim 2^k$). In Table 4, the ratio t_1/t_3 is reported for random linear codes with low information rate ($n \sim 2^k$). In Table 5, the ratio t_1/t_3 is reported for random linear codes with low information rate ($n \sim k$).

An Intel Core i7 CPU 920 processor at 2.67 GHz has been used for the computations.

Table 3 Ratio between our method and MAGMA brute force implementation timings

kn	2^k	2^{k+1}	2^{k+2}	2^{k+3}	2^{k+4}	2^{k+5}	2^{k+6}	2^{k+7}	2^{k+8}	2^{k+9}	2^{k+10}
2	5.433	5.176	4.871	4.699	4.149	3.693	3.167	2.825	2.693	2.507	2.450
3	6.990	6.421	6.045	5.318	4.454	3.593	3.184	2.874	2.637	2.527	2.492
4	9.900	8.910	7.721	6.079	4.733	3.859	3.335	3.010	2.863	2.810	2.789
5	15.25	12.80	9.809	7.254	5.533	4.507	3.961	3.635	3.531	3.492	3.445
6	22.43	16.64	11.82	8.660	6.776	5.706	5.130	4.877	4.795	4.693	4.694

There are many cases, both in the high and low information rate range, where our method is faster than the Brouwer-Zimmermann method. This is not surprising, since it is known that there are cases where even brute force performs better than the Brouwer-Zimmermann method, especially for low information rate codes.

However, in the analysed range, our method never succeeded when compared to MAGMA implementation of brute force. Only for n exponentially growing with k the ratio t_1/t_2 tends to decrease.

Table 4 Ratio between our method and MAGMA Brouwer-Zimmermann implementation timings: low information rate codes

kn	2^k	2^{k+1}	2^{k+2}	2^{k+3}	2^{k+4}	2^{k+5}	2^{k+6}	2^{k+7}	2^{k+8}	2^{k+9}	2^{k+10}
2	0.474	0.300	0.152	0.212	0.130	0.073	0.089	0.041	0.033	0.018	0.009
3	0.344	0.250	0.344	0.203	0.109	0.055	0.054	0.040	0.022	0.011	0.005
4	0.452	0.232	0.302	0.154	0.074	0.064	0.046	0.025	0.013	0.006	0.003
5	0.514	0.645	0.308	0.140	0.104	0.071	0.038	0.019	0.008	0.004	0.002
6	1.208	0.568	0.248	0.092	0.109	0.057	0.028	0.012	0.006	0.003	0.001

Table 5 Ratio between our method and MAGMA Brouwer-Zimmermann implementation timings: high information rate codes

kn	$k+2$	$k+3$	$k+4$	$k+5$	$k+6$	$k+7$	$k+8$	$k+9$	$k+10$	$k+10$	$k+12$
2	0.465	0.394	0.327	0.336	0.269	0.255	0.234	0.256	0.189	0.222	0.186
3	0.755	0.438	0.410	0.406	0.357	0.321	0.356	0.294	0.311	0.290	0.255
4	1.347	0.958	0.559	0.604	0.655	0.708	0.491	0.522	0.540	0.559	0.412
5	2.689	2.216	1.471	0.954	1.072	1.121	1.425	1.149	0.839	0.893	1.148
6	5.278	4.471	3.877	2.880	1.613	2.138	2.424	3.133	2.496	2.517	1.253

6 Binary codes whose cardinality is not a power of 2

Algorithm 1 can be modified to work also with binary codes whose cardinality is not a power of 2. We only mention two techniques that can be used.
The first method consists of expanding the code until it reaches a size of 2^k. The key observation is that the minimum weight vector of a list of vectors in $\mathbb{F}^n$ (i.e., the codewords of C) is equal to the minimum weight vector of the same list concatenated to the list of some repeated words of C (even though this new list is not a code anymore).
A second approach is to divide the code C in sub-codes whose cardinality is a power of 2. Then to each of these codes we can apply Algorithm 1 and then take the minimum of all the results. See Bellini (2014) for details.

7 Conclusions

We presented a deterministic algorithm to compute the distance and weight distribution, and thus the minimum distance and the minimum weight, of any random binary code represented as a set of Boolean functions in NNF. We showed that our method performs better than brute force (applied instead to the vectorial representation of a code) for those codes with low information rate and sparse NNF representation, while in the general case, it achieves the same asymptotic computational complexity as brute force methods.

It is not straightforward to generalise our method to generic random codes over a finite field $\mathbb{F}_q$, and such a generalisation may drastically increase the complexity of the method. Also, a deeper analysis and a precise characterisation should be performed to understand when a Boolean NNF representation of a code is more convenient than a vectorial representation.

Acknowledgements

The main results of this paper are contained in the first author's PhD thesis at the University of Trento. The first author would like to thank his supervisor (the second author) and the thesis's referee, Eleonora Guerrini.

References

Bellini, E. (2014) *Computational Techniques for Nonlinear Codes and Boolean Functions*, PhD thesis, University of Trento.

Bellini, E., Guerrini, E. and Sala, M. (2014) 'Some bounds on the size of codes', *IEEE Trans. Inform. Theory*, Vol. 60, No. 3, pp.1475–1480.

Berlekamp, E.R., McEliece, R.J. and van Tilborg, H.C.A. (1978) 'On the inherent intractability of certain coding problems', *IEEE Trans. on Inf. Th.*, Vol. 24, No. 3, pp.384–386.

Borges, J., Fernández-Córdoba, C., Pujol, J., Rifà, J. and Villanueva, M. (2010) '$\mathbb{Z}_2\mathbb{Z}_4$-linear codes: generator matrices and duality', *Designs, Codes and Cryptography*, Vol. 54, No. 2, pp.167–179.

Bouyukliev, I. and Bakoev, V. (2008) 'A method for efficiently computing the number of codewords of fixed weights in linear codes', *Discrete Applied Mathematics*, Vol. 156, No. 15, pp.2986–3004.

Canteaut, A. and Chabaud, F. (1998) 'A new algorithm for finding minimum-weight words in a linear code: application to McEliece's cryptosystem and to narrow-sense BCH codes of length 511', *IEEE Transactions on Information Theory*, Vol. 44, No. 1, p.367.

Carlet, C. (2002) *On the coset weight divisibility and nonlinearity of resilient and correlation-immune functions*', *Sequences and their Applications*, Springer, Bergen, Norway, pp.131–144.

Carlet, C. (2010) 'Boolean functions for cryptography and error correcting codes', *Boolean Models and Methods in Mathematics, Computer Science, and Engineering*, Cambridge, UK, pp.257–397.

Carlet, C. and Guillot, P. (1999) *A new representation of Boolean functions*', *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, Springer, Honolulu, Hawaii, USA, pp.94–103.

Carlet, C. and Guillot, P. (2001) 'Bent, resilient functions and the numerical normal form', *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, Vol. 56, pp.87–96.

Guerrini, E., Meneghetti, A. and Sala, M. (2016) 'On optimal nonlinear systematic codes', *IEEE Transactions on Information Theory*, Vol. 62, No. 6, pp.3103–3112.

Guerrini, E., Orsini, M. and Sala, M. (2010) 'Computing the distance distribution of systematic non-linear codes', *Journal of Algebra and Its Applications*, Vol. 9, No. 2, pp.241–256.

Hammons, J.A.R., Kumar, P.V., Calderbank, A.R., Sloane, N.J.A. and Solé, P. (1994) 'The $\mathbf{Z}_4$-linearity of Kerdock, Preparata, Goethals, and related codes', *IEEE Trans. on Inf. Th.*, Vol. 40, No. 2, pp.301–319.

Leon, J.S. (1988) 'A probabilistic algorithm for computing minimum weights of large error-correcting codes', *IEEE Trans. on Inf. Th.*, Vol. 34, No. 5, part 2, pp.1354–1359, Coding techniques and coding theory.

Lisoněk, P. and Trummer, L. (2015) 'An extension of the Brouwer-Zimmermann minimum weight algorithm', *Coding Theory and Applications*, Springer, Palmela Castle, Portugal, pp.255–262.

MacWilliams, F.J. and Sloane, N.J.A. (1977) The theory of error-correcting codes. I, North-Holland Publishing Co., Amsterdam, North-Holland Mathematical Library, Vol. 16.

Mitchell, C. (1989) 'Distance-invariant error control codes from combinatorial designs', *Electronics Letters*, Vol. 25, No. 22, pp.1528–1529.

Preparata, F.P. (1968) 'A class of optimum nonlinear double-error correcting codes', *Inform. Control*, Vol. 13, No. 13, pp.378–400.

Pujol, J., Villanueva, M. and Zeng, F. (2012) *Minimum Distance of Binary Nonlinear Codes*.

Singleton, R.C. (1964) 'Maximum distance q-nary codes', *IEEE Trans. Information Theory*, Vol. IT-10, pp.116–118.

Stern, J. (1989) 'A method for finding codewords of small weight', *Coding Theory and Applications*, Springer, Toulon, France, pp.106–113.

Vardy, A. (1997) 'The intractability of computing the minimum distance of a code', *IEEE Trans. on Inf. Th.*, Vol. 43, No. 6, pp.1757–1766.

Villanueva, M., Zeng, F. and Pujol, J. (2014) 'Efficient representation of binary nonlinear codes: constructions and minimum distance computation', *Designs, Codes and Cryptography* pp.1–19.

White, G. and Grassl, M. (2006) 'A new minimum weight algorithm for additive codes', *IEEE International Symposium on Information Theory, 2006*, IEEE, Seattle, Washington, USA, pp.1119–1123.

Zimmermann, K. (1996) *Integral Hecke Modules, Integral Generalized Reed-Muller Codes, and Linear Codes*, Berichte des Forschungsschwerpunktes Informations- und Kommunikationstechnik, Techn. Univ. Hamburg-Harburg.