Noname manuscript No.
(will be inserted by the editor)

Efficient representation of binary nonlinear codes: constructions and minimum distance computation

Mercè Villanueva · Fanxuan Zeng · Jaume Pujol

Received: date / Accepted: date

Abstract A binary nonlinear code can be represented as a union of cosets of a binary linear subcode. In this paper, the complexity of some algorithms to obtain this representation is analyzed. Moreover, some properties and constructions of new codes from given ones in terms of this representation are described. Algorithms to compute the minimum distance of binary nonlinear codes, based on known algorithms for linear codes, are also established, along with an algorithm to decode such codes. All results are written in such a way that they can be easily transformed into algorithms, and the performance of these algorithms is evaluated.

Keywords nonlinear code, kernel, minimum distance, minimum weight, decoding, algorithms

Mathematics Subject Classification (2000) 94B60 · 94B25 · 94B35

1 Introduction

Let $\mathbb{Z}_2$ be the ring of integers modulo 2 and let $\mathbb{Z}_2^n$ be the set of all binary vectors of length n. The *(Hamming) distance* $d(u,v)$ between two vectors $u,v \in \mathbb{Z}_2^n$ is the number of coordinates in which u and v differ. The *(Hamming) weight* $wt(u)$ of $u \in \mathbb{Z}_2^n$ is $wt(u) = d(u,\mathbf{0})$, where $\mathbf{0}$ is the all-zero vector of length n. An (n,M,d) *binary code* C is a subset of $\mathbb{Z}_2^n$ with M vectors and minimum Hamming distance d. The vectors of a code are called *codewords* and the *minimum (Hamming) distance* of C, denoted by $d(C)$, is the minimum value of $d(u,v)$ for all $u,v \in C$ and $u \neq v$. The *minimum (Hamming) weight* of C, denoted by $wt(C)$, is the smallest weight of all nonzero codewords in C. The *distance* of a vector $x \in \mathbb{Z}_2^n$ from C is the number $d(x,C) = \min\{d(x,c) : c \in C\}$,

This work has been partially supported by the Spanish MICINN under Grants TIN2010-17358 and TIN2013-40524-P, and by the Catalan AGAUR under Grant 2014SGR-691. The material in this paper was presented in part at the Applications of Computer Algebra Conference, Málaga, Spain 2013 [22].

Dept. of Information and Communications Engineering, Universitat Autònoma de Barcelona, 08193-Bellaterra, Spain.

and the *covering radius* of C, denoted by $\rho(C)$, is $\rho(C) = \max\{d(x, C) : x \in \mathbb{Z}_2^n\}$. We say that a code C is *maximal* if $\rho(C) \leq d(C) - 1$.

Two binary codes C_1 and C_2 of length n are said to be *permutation equivalent* if there exists a coordinate permutation π such that $C_2 = \{\pi(c) : c \in C_1\}$. They are said to be *equivalent* if there exists a vector $a \in \mathbb{Z}_2^n$ and a coordinate permutation π such that $C_2 = \{a + \pi(c) : c \in C_1\}$. Note that two equivalent codes have the same minimum distance. If C is linear, $\mathbf{0} \in C$; but if C is nonlinear, $\mathbf{0}$ does not need to belong to C. In this case, we can always consider a new binary code $C' = C + c$ for any $c \in C$, which is equivalent to C, such that $\mathbf{0} \in C'$. Therefore, from now on, we assume that $\mathbf{0} \in C$.

Given a binary code C, the problem of storing C in memory is a well known problem. If C is linear, that is, it is a subgroup of $\mathbb{Z}_2^n$, then it can be compactly represented using a binary generator matrix. On the other hand, if C is nonlinear, then a solution would be to know whether it has another structure or not. 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 a quaternary generator matrix [4,12]. In general, binary codes without considering any such structure can be seen as a union of cosets of a binary linear subcode of C [1]. For example, the original Preparata and Kerdock codes are defined like this [16], and new binary nonlinear codes have also been constructed as an union of cosets of a linear code [9,18]. This allows us to represent a binary code as a set of representative codewords instead of as a set with all codewords. These representative codewords can be organized as a matrix, called *parity-check system* [13], which is a generalization of the parity-check matrix for linear codes [16].

The problem of computing the minimum distance of a binary code C is computationally difficult, and has been proven to be NP-hard for binary linear codes [23]. Note that in order to establish the error-correcting capability of a code, it is necessary to compute its minimum distance. If C is linear, the minimum distance coincides with the minimum weight, and the Brouwer-Zimmermann minimum weight algorithm for linear codes over finite fields [26,2] can be used. This algorithm can be found implemented in the computational algebra system Magma [8,10,24]. Other algorithms related to this problem for linear codes can be found in [3,6,17]. On the other hand, if C is nonlinear, the minimum weight and distance does not always coincide, and as far as we know there is no algorithm to compute them comparable to Brouwer-Zimmermann algorithm for linear codes. An algorithm based on Gröbner bases to compute the distance distribution for systematic nonlinear codes can be found in [11].

The hardest problem in the process of transmitting information is decoding. For linear codes, a general decoding algorithm is the syndrome decoding, which is only suitable for linear codes with a small codimension. Other general decoding methods are based on finding a vector of minimum weight in a linear code or a coset of a linear code. They have better performance than the syndrome decoding for codes with a moderate large codimension, and have been used to attack the McEliece public-key cryptosystem [7,24]. For specific families of linear codes used in applications, other more efficient decoding algorithms can be applied. On the other hand, there is no general decoding algorithm for nonlinear codes other than brute force. Using the representation as a union of cosets of a linear subcode and the minimum weight computation, we propose a nontrivial general method to decode nonlinear codes.

In this paper, we focus on binary nonlinear codes represented as a union of cosets of a binary linear subcode called kernel. In Section 2, we describe this coset representation, show that it can allow to store nonlinear codes in a more efficient way, and analyze the complexity of some algorithms to compute the kernel and coset representatives from a

given nonlinear code. In Section 3, we give some properties and constructions of new codes from given ones in terms of the kernel and coset representatives. In Section 4, we propose and analyze algorithms to compute the minimum weight and minimum distance of binary nonlinear codes, based on the coset structure and the known algorithms for linear codes. The performance of the different algorithms presented in the paper strongly relies on the nonlinear code to have a large kernel. In Section 5, we generalize previous general decoding methods for linear codes to nonlinear codes using the coset structure. All the new features are evaluated on theoretical level and partly using experimental data. Finally, in Section 6, we give some conclusions.

2 Representation of Binary Nonlinear Codes

Two structural properties of binary codes are the rank and dimension of the kernel. The *rank* of a binary code C, denoted by ϱ, is simply the dimension of the linear span, $\langle C \rangle$, of C. The *kernel* of a binary code C is defined as $K_C = \{x \in C \ : \ x + C = C\}$. Since $\mathbf{0} \in C$, K_C is a binary linear subcode of C. We denote by κ the dimension of K_C. In general, C can be written as a union of cosets of K_C, and K_C is the largest such linear code for which this is true [1]. Therefore,

$$C = \bigcup_{i=0}^{t} \Big(K_C + v_i \Big), \tag{1}$$

where $v_0 = \mathbf{0}$, $t+1 = M/2^{\kappa}$, $M = |C|$ and $v_1, \ldots, v_t$ are representatives of the cosets of K_C, called *coset representatives*. Note that $t \neq 1$, because if $t = 1$, $C = K_C \cup (K_C + v_1)$, but then C would be linear, so $C = K_C$. It is important to emphasize that the coset representatives are not necessarily the ones having minimum weight in each coset. The parameters ϱ and κ can be used to distinguish between nonequivalent binary codes, since equivalent ones must have the same ϱ and κ. Therefore, they provide a sufficient condition which is not necessary, since two nonequivalent binary codes could have the same parameters ϱ and κ.

Let C be a binary code of length n with kernel K_C of dimension κ and t coset representatives given by the set $L = \{v_1, \ldots, v_t\}$. Note that we can represent C as the kernel K_C plus the coset representatives L. Since K_C is linear, it can be compactly represented by its binary generator matrix G of size $\kappa \times n$. Therefore, considering L as the matrix where in the t rows there are the coset representatives, the binary code C can be also represented by the matrix $\binom{G}{L}$. Since the kernel takes up a memory space of order $O(n\kappa)$, the kernel plus the t coset representatives take up a memory space of order $O(n(\kappa + t))$. For the case $t = 0$, that is, when the binary code C is linear, we have that $C = K_C$ and the code can be represented by its binary generator matrix, so the memory space is of order $O(n\kappa)$. On the other hand, for the case $t + 1 = M$, this solution is as bad as representing the code as a set of all its codewords, so it takes up a memory space of order $O(nM)$.

For example, applying this representation to the set of all completely classified binary perfect codes of length 15 and extended perfect codes of length 16, we obtain a very significant storage memory reduction. It is known that there are exactly 5983 binary perfect codes of length 15 and 2165 binary extended perfect codes of length 16, each one having 2048 codewords [19]. The binary perfect codes of length 15 have kernels of different dimensions, distributed as it is shown in Table 1 [20]. Therefore, instead of

taking up $5983 \cdot 2048 \cdot 4 = 49012736$ hexadecimal numbers by encoding each codeword in hexadecimal notation, it only takes 3677928 by storing the codewords of a generator matrix of the kernel and the set of coset representatives for each binary code. This represents only 7.5% of the original stored codewords. Similarly, the extended perfect codes of length 16 can be compressed from $2165 \cdot 2048 \cdot 4 = 17735680$ hexadecimal numbers to 1439336, which represents only 8.1% of the original stored codewords. Note that although most of these codes have kernels with small dimension, that is, they are far from being linear, we obtain a high storage memory reduction.

Dimension of the kernel	1	2	3	4	5	6	7	8	9	11
Number of codes	19	177	1295	2896	1222	305	48	17	3	1

Table 1 Dimension of the kernels for the binary perfect codes of length 15.

As we have seen, the above matrix $\binom{G}{L}$ gives us a compact representation for binary codes. Equivalently, a binary code C can also be represented in a compact way using an $(n-\kappa) \times (n+t)$ binary matrix $(H\ S)$, where H is a generator matrix of the dual code $K_C^{\perp}$ and $S = (Hv_1^T\ Hv_2^T\ \ldots\ Hv_t^T)$. This matrix is called *parity-check system* of C, and the binary linear code generated by $(H\ S)$ is called the *super dual* of C [13]. Unlike the matrix $\binom{G}{L}$, any generator matrix $(H'\ S')$ of the super dual or $(H'\ \pi(S'))$, where π is a column permutation, can be used to represent the binary code C. Note that if C is a linear code, the super dual is the dual code $C^{\perp}$ and a parity-check system is a parity-check matrix of C [13].

By using just the definition of the kernel, we can define a straightforward algorithm to compute the kernel and coset representatives of a binary code C. This algorithm requires the classification of the M codewords of C. Moreover, if we assume that the codewords are sorted, then $M^2 \log M$ operations (additions and searches) would need to be executed. Since $M = 2^{\kappa}(t+1)$, this algorithm is exponential in κ, the dimension of K_C. Despite of the exponential behaviour, using some well known properties of the kernel, in most cases, it is possible to improve it in order to compute the kernel in a more efficient way. This improved algorithm is described in Algorithm 1.

Algorithm 1 uses the following three properties of the kernel. Let K' be a subset of the kernel of a binary code C, $K' \subseteq K_C$, (i) if $k \in K_C$, then $K' + k \subseteq K_C$; (ii) if $c \in C$ and $(C \backslash K') + c \subseteq C$, then $c \in K_C$; (iii) if $c \notin K_C$, then $(K' + c) \cap K_C = \emptyset$. Therefore, depending on κ, the complexity can be reduced. To analyze Algorithm 1 we study the worst and best case. In the worst case, $\kappa = 0$ and we still need $M^2 \log M$ operations as the previous algorithm. The best case is when C is linear, that is, $C = K_C$. Then, in each iteration the cardinality of the kernel is duplicated. Thus, we need $2^{\kappa} - 2^i$ additions and searches in each step i, $i \in \{0, \ldots, \kappa-1\}$, so $\sum_{i=0}^{\kappa-1}(2^{\kappa} - 2^i) = 2^{\kappa}(\kappa-1)+1$ additions and searches. Hence, the number of operations is $(M(\kappa-1)+1)\log M$, where $M = 2^{\kappa}$.

A *partial kernel* of C is a linear subcode of K_C. Note that in Algorithm 1 a partial kernel is built in each step. Therefore, in the general case, the number of operations depends strongly on how the partial kernel is growing. If the kernel is small or the partial kernel grows slowly up to the kernel, the number of operations is close to the worst case. Otherwise, the number of operations can be reduced significantly using also the following property.

Proposition 1 *Let K' be a partial kernel of a binary code C and $L' = \{v'_1, \ldots, v'_{t'}\}$ be the corresponding coset representatives, that is, $C = \bigcup_{i=0}^{t'}(K' + v'_i)$, where $v'_0 = \mathbf{0}$. Then, for each $v'_j \in L'$, $K' + v'_j \subseteq K_C$ if and only if $v'_j + v'_i \in C$ for all $i \in \{1, \ldots, t'\}$.*

Proof If $K' + v'_j \subseteq K_C$, then $v'_j \in K_C$ and, by the kernel definition, $v'_j + v'_i \in C$ for all $i \in \{1, \ldots, t'\}$. Suppose that $v'_j + v'_i \in C$ for all $i \in \{1, \ldots, t'\}$. To prove that $K' + v'_j \subseteq K_C$, it is enough to show that $v'_j \in K_C$. For any $c \in C$, there exist $k' \in K'$ and $v'_i \in L'$ such that $c = k' + v'_i$. Then, $v'_j + c = v'_j + k' + v'_i \in K' + v'_j + v'_i \subseteq C$, since $K' \subseteq K_C$ and $v'_j + v'_i \in C$. □

Note that if C is binary nonlinear and $M = 2^r$ ($r \geq 2$), then $|K_C| \leq 2^{r-2}$; and if $M = 2^r \cdot s$ ($r \geq 0$, $s \geq 3$ odd), then $|K_C| \leq 2^r$. Hence, in Algorithm 1, when $M = 2^r$ and the dimension of the partial kernel K' is $r - 1$, the code C is linear, so $K_C = C$; and when $M = 2^r \cdot s$ and the dimension of the partial kernel K' is r, $K_C = K'$.

Algorithm 1: Kernel and coset representatives computation.

```
Data: A sorted binary code C.
Result: The kernel K_C and coset representatives L = {v_1, ..., v_t}.
begin
    K_C ← {0}
    C* ← C\{0}
    L ← ∅
    R ← ∅
    while |C*| > 0 do
        c ← First(C*)
        if C* + c ⊆ C then
            C* ← C*\(K_C + c)
            K_C ← K_C ∪ (K_C + c)
        else
            R ← R ∪ (K_C + c)
            C* ← C*\(K_C + c)
    while R ≠ ∅ do
        v ← First(R)
        L ← L ∪ {v}
        R ← R\(K_C + v)
    return K_C, L
```

For large M, the computation of the kernel and coset representatives using Algorithm 1 can still be inefficient. However, some well known code constructions allow to compute the kernel and coset representatives of new codes in a very efficient way, as we can see in Section 3.

3 Properties and Constructions

Using the representation given in Section 2, rather than using the list of all codewords, we can manipulate and construct new binary nonlinear codes from old ones in a more efficient way. Specifically, in this section, we show how to check equality and inclusion of two given nonlinear codes from their kernels and coset representatives, and how to compute the kernel and coset representatives of new codes (union, intersection, extended code, punctured code, shortened code, direct sum, Plotkin sum) from given ones, which are already represented in this way. Note that all these results are written to be implemented easily as algorithms.

Let $col(S)$ denote the set of columns of the matrix S.

Proposition 2 *[13] Let C be a binary code of length n with parity-check system $(H\ S)$. Then, $c \in C$ if and only if $Hc^T \in \{\mathbf{0}\} \cup col(S)$.*

Proposition 3 *[13] Let C be a binary code of length n with rank ϱ, dimension of the kernel κ and parity-check system $(H\ S)$. Then, $\varrho = n - rank(H) + rank(S)$ and $\kappa = n - rank(H)$.*

Let C_1 and C_2 be two binary codes of length n_1 and n_2, respectively. Let $(H_1\ S_1)$ and $(H_2\ S_2)$ be the parity-check systems of C_1 and C_2, respectively. The matrices H_1 and H_2 are the generator matrices of the dual codes $K_{C_1}^{\perp}$ and $K_{C_2}^{\perp}$, and κ_1 and κ_2 the dimension of the kernel of C_1 and C_2, respectively. The coset representatives for C_1 and C_2 are the sets $\{v_1, \ldots, v_{t_1}\}$ and $\{w_1, \ldots, w_{t_2}\}$, which give us the matrices $S_1 = (H_1 v_1^T\ H_1 v_2^T\ \ldots\ H_1 v_{t_1}^T)$ and $S_2 = (H_2 w_1^T\ H_2 w_2^T\ \ldots\ H_2 w_{t_2}^T)$, respectively.

Proposition 4 (Equality) *Let C_1 and C_2 be two binary codes of length n. Then, $C_1 = C_2$ if and only if $K_{C_1} = K_{C_2}$, $t_1 = t_2$, and $H_2 v_i^T \in col(S_2)$ for all $i \in \{1, \ldots, t_1\}$.*

Proof It is clear that if $C_1 = C_2$, then $K_{C_1} = K_{C_2}$, $t_1 = t_2$ and $v_i \in C_2 \backslash K_{C_2}$ for all $i \in \{1, \ldots, t_1\}$. By Proposition 2, $v_i \in C_2 \backslash K_{C_2}$ if and only if $H_2 v_i^T \in col(S_2)$.

On the other hand, if $K_{C_1} = K_{C_2}$ and $t_1 = t_2$, then $|C_1| = |C_2|$. Moreover, given a codeword $c \in C_1$, $c = k + v_i$ for some $i \in \{0, 1, \ldots, t_1\}$ and $k \in K_{C_1}$, where $v_0 = \mathbf{0}$. If $i = 0$, then $c \in K_{C_1} = K_{C_2} \subseteq C_2$. If $i \in \{1, \ldots, t_1\}$, then $H_2 c^T = H_2 (k + v_i)^T = H_2 k^T + H_2 v_i^T = H_2 v_i^T \in col(S_2)$, since $k \in K_{C_1} = K_{C_2}$. Therefore, by Proposition 2, we have that $c \in C_2$. Since $C_1 \subseteq C_2$ and $|C_1| = |C_2|$, we obtain that $C_1 = C_2$. □

Proposition 5 (Inclusion) *Let C_1 and C_2 be two binary codes of length n. Let $K = K_{C_1} \cap K_{C_2}$ of dimension κ and $K_{C_1} = \bigcup_{j=0}^{h_1} (K + x_j)$, where $h_1 = 2^{\kappa_1 - \kappa} - 1$. Then, $C_1 \subseteq C_2$ if and only if $H_2 (x_j + v_i)^T \in \{\mathbf{0}\} \cup col(S_2)$, for all $i \in \{0, 1, \ldots, t_1\}$ and $j \in \{0, 1, \ldots, h_1\}$, where $v_0 = x_0 = \mathbf{0}$.*

Proof Note that $C_1 = \bigcup_{i=0}^{t_1} \bigcup_{j=0}^{h_1} (K + x_j + v_i)$. It is clear that if $C_1 \subseteq C_2$, then $x_j + v_i \in C_2$, which is equivalent to $H_2 (x_j + v_i)^T \in \{\mathbf{0}\} \cup col(S_2)$, by Proposition 2.

On the other hand, given $c \in C_1$, $c = k + x_j + v_i$ for some $i \in \{0, 1, \ldots, t_1\}$, $j \in \{0, 1, \ldots, h_1\}$ and $k \in K$. Thus, $H_2 c^T = H_2 (k + x_j + v_i)^T = H_2 (x_j + v_i)^T \in \{\mathbf{0}\} \cup col(S_2)$, since $k \in K \subseteq K_{C_2}$. By Proposition 2, $c \in C_2$, so $C_1 \subseteq C_2$. □

Proposition 6 (Intersection) *Let C_1 and C_2 be two binary codes of length n. Let $K = K_{C_1} \cap K_{C_2}$ of dimension κ and $K_{C_1} = \bigcup_{j=0}^{h_1} (K + x_j)$ and $K_{C_2} = \bigcup_{j=0}^{h_2} (K + y_j)$, where $h_1 = 2^{\kappa_1 - \kappa} - 1$, $h_2 = 2^{\kappa_2 - \kappa} - 1$ and $x_0 = y_0 = \mathbf{0}$. Let $C = C_1 \cap C_2$. Then, $K \subseteq K_C$ is a partial kernel of C and $C = \bigcup_{v \in L_I} (K + v)$, where $L_I = \{x_j + v_i\ :\ j \in \{0, 1, \ldots, h_1\},\ i \in \{0, 1, \ldots, t_1\}$ and $H_2 (x_j + v_i)^T \in \{\mathbf{0}\} \cup col(S_2)\}$.*

Proof First, we show that $K \subseteq K_C$. Given $k \in K$, for any $v \in C$, $k + v \in C_l$ since $v \in C_l$ and $k \in K_{C_l}$, $l \in \{1, 2\}$. Therefore, $k + v \in C$, and we have that $k \in K_C$. Note that $C_1 = \bigcup_{i=0}^{t_1} \bigcup_{j=0}^{h_1} (K + x_j + v_i)$. Since $K \subseteq K_{C_2}$, $K + x_j + v_i \subseteq C_2$ if and only if $x_j + v_i \in C_2$, which is equivalent to $H_2 (x_j + v_i)^T \in \{\mathbf{0}\} \cup col(S_2)$ by Proposition 2. □

Proposition 7 (Union) *Let C_1 and C_2 be two binary codes of length n. Let $K = K_{C_1} \cap K_{C_2}$ of dimension κ and $K_{C_1} = \bigcup_{j=0}^{h_1} (K + x_j)$ and $K_{C_2} = \bigcup_{j=0}^{h_2} (K + y_j)$, where $h_1 = 2^{\kappa_1 - \kappa} - 1$, $h_2 = 2^{\kappa_2 - \kappa} - 1$ and $x_0 = y_0 = \mathbf{0}$. Let $C = C_1 \cup C_2$. Then, $K \subseteq K_C$ is a partial kernel of C and $C = \bigcup_{v \in L_U} (K + v)$, where $L_U = \{x_j + v_i\ :\ j \in \{0, 1, \ldots, h_1\},\ i \in \{0, 1, \ldots, t_1\}\} \backslash L_I \cup \{y_j + w_i\ :\ j \in \{0, 1, \ldots, h_2\}, i \in \{0, 1, \ldots, t_2\}\}$.*

Proof Straightforward using the same arguments as in the proof of Proposition 6. □

Let C be a binary code of length n. The *extended binary code* of C, denoted by $\widehat{C}$, is defined as the set of codewords constructed by adding a parity coordinate, that is, a 0 at the end of every codeword of C of even weight, and a 1 at the end of every codeword with odd weight. The *punctured binary code* of C in the jth coordinate, denoted by C^j, is the code consisting of the codewords of C after deleting the jth coordinate. The *shortened binary code* of C in the jth coordinate, denoted by C_0^j, is the code consisting of the codewords of C having 0 in the jth coordinate and deleting this coordinate.

Proposition 8 (Extended Code) *Let C be a binary code of length n with kernel K_C and t coset representatives given by the set $L = \{v_1, \ldots, v_t\}$. Then, the extended binary code $\widehat{C}$ has kernel $\widehat{K_C}$ and t coset representatives given by the set $\{\hat{v}_1, \ldots, \hat{v}_t\}$, where $\hat{v}_i$ is v_i after adding a parity coordinate for $i \in \{1, \ldots, t\}$.*

Proof Obviously, the extended code $\widehat{C}$ can be written as $\widehat{C} = \bigcup_{i=0}^{t} (\widehat{K_C} + \hat{v}_i)$, where $\hat{v}_0 = \mathbf{0}$. Therefore, $\widehat{K_C} \subseteq K(\widehat{C})$, and we only need to prove that $K(\widehat{C}) \subseteq \widehat{K_C}$. Let $\hat{k} \in K(\widehat{C}) \subseteq \widehat{C}$. For any $\hat{c} \in \widehat{C}$, we have that $\hat{k} + \hat{c} \in \widehat{C}$. Therefore, $k \in C$, and $k + c \in C$ for any $c \in C$, so $k \in K_C$, which means that $\hat{k} \in \widehat{K_C}$ and $K(\widehat{C}) \subseteq \widehat{K_C}$. □

If C is an $(n, |C|, d)$ binary code, then the extended code $\widehat{C}$ contains only even weight codewords and is an $(n+1, |C|, \hat{d})$ binary code, where $\hat{d}$ equals d if d is even and equals $d+1$ if d is odd. It is known that this is true if C is linear [14], and it can be easily generalized to binary nonlinear codes. Note that if the distance between two codewords in C is even, after adding a parity coordinate, both codewords are at the same distance, and if the distance between them is odd, the distance increases by 1, after adding a parity coordinate.

Proposition 9 (Punctured Code) *Let C be an $(n, |C|, d)$ binary code with kernel K_C and t coset representatives given by the set $L = \{v_1, \ldots, v_t\}$. Let K^j be the punctured binary code of K_C in the jth coordinate. Then, the punctured binary code C^j in the jth coordinate is $C^j = \bigcup_{v \in L^j} (K^j + v)$, where $L^j \subseteq \{\mathbf{0}, v_1^j, \ldots, v_t^j\}$ and v_i^j is v_i after deleting the jth coordinate for $i = 1, \ldots, t$.*

(i) If $d > 1$, C^j is an $(n-1, |C|, d')$ binary code, where $d' = d-1$ if C has two codewords at distance d which differ at the jth coordinate and $d' = d$ otherwise.

(ii) If $d = 1$, C^j is an $(n-1, |C|, 1)$ binary code if the codewords in C at distance 1 do not differ at the jth coordinate; otherwise, C^j is an $(n-1, |C^j|, d')$ binary code with $d' \geq 1$ and $|C^j| < |C|$.

Proof Let v^j be the vector v after deleting the jth coordinate. Given any codeword $c \in C$, $c = k + v_i$ for some $k \in K_C$ and $i \in \{0, 1, \ldots, t\}$. Since $c^j = k^j + v_i^j$, $C^j \subseteq \bigcup_{v \in L^j} (K^j + v)$, where $L^j = \{\mathbf{0}, v_1^j, \ldots, v_t^j\}$.

If $d > 1$, or $d = 1$ but the codewords at distance 1 do not differ at the jth coordinate, then $|C^j| = |C|$. Since $C^j \subseteq \bigcup_{v \in L^j} (K^j + v)$ and $|C^j| = |C|$, $C^j = \bigcup_{v \in L^j} (K^j + v)$. Moreover, if $d > 1$, C^j is an $(n-1, |C|, d')$ binary code, where $d' = d-1$ if C has two codewords at distance d which differ at the jth coordinate and $d' = d$ otherwise.

Finally, if there exist $c, u \in C$ such that they only differ at the jth coordinate, then $c - u = e_j$, where e_j is the vector with 1 in the jth coordinate and zero elsewhere. First, if $e_j \in K_C$, then c and u are in the same coset, and the same happens with any such pair c, u, since K_C is linear. In this case, K^j has dimension $\kappa - 1$, and the result follows

with $L^j = \{\mathbf{0}, v_1^j, \ldots, v_t^j\}$. Second, if $e_j \notin K_C$, then c and u are in different cosets, that is, $c \notin K_C + u$. However, after deleting the jth coordinate, $K^j + c^j = K^j + u^j$ (or equivalently, $c^j \in K^j + u^j$) and the result follows with $L^j \subsetneq \{\mathbf{0}, v_1^j, \ldots, v_t^j\}$. Note that in both cases we have that $d' \geq 1$ and $|C^j| < |C|$. □

Proposition 10 (Shortened Code) *Let C be a binary code of length n with kernel K_C and t coset representatives given by the set $L = \{v_1, \ldots, v_t\}$. Let K_0^j be the shortened binary code of K_C in the jth coordinate. Then, the shortened binary code C_0^j in the jth coordinate is $C_0^j = \bigcup_{v \in L^j}(K_0^j + v)$, where $L^j = \{\mathbf{0}\} \cup \{v_i^j \ : \ i \in I\}$, $I = \{i \in \{1, \ldots, t\} \ : \ $ there exists $v_i' \in K_C + v_i$ such that has 0 in the jth coordinate$\}$ and v_i^j is v_i' after deleting the jth coordinate for $i \in I$. Moreover, C_0^j is an $(n-1, |C_0^j|, d')$ binary code with $d' \geq d(C)$ and $|C_0^j| \leq |C|$.*

Proof Let $C_0 \subseteq C$ and $K_0 \subseteq K_C$ be the subsets of C and K_C, respectively, containing the codewords having 0 in the jth coordinate.

If $K_0 \subsetneq K_C$, then $I = \{1, \ldots, t\}$. Therefore, $\bigcup_{v \in L^j}(K_0^j + v) \subseteq C_0^j$, where $L^j = \{\mathbf{0}, v_1^j, \ldots, v_t^j\}$. Moreover, for any $c_0 \in C_0 \subseteq C$, since $c_0 \in C$, there exist $k \in K_C$ and $i \in \{1, \ldots, t\}$ such that $c_0 = k + v_i'$. Since c_0 and v_i' have 0 in the jth coordinate, $k \in K_0$. Therefore, after deleting the jth coordinate, $C_0^j = \bigcup_{v \in L^j}(K_0^j + v)$.

If $K_0 = K_C$, then if v_i has 1 in the jth coordinate, does not exist any $v_i' \in K_C + v_i$ such that has 0 in the jth coordinate. Therefore, $C_0^j = \bigcup_{v \in L^j}(K_0^j + v)$, where L^j is the set defined in the statement. Note that, in this case, the number of cosets in C_0^j may be smaller than the number of cosets t in C.

Finally, it is straightforward to see that $d' \geq d(C)$ and $|C_0^j| \leq |C|$. □

Another way to construct new codes is to combine two codes together in a proper way. The most known such constructions are called *direct sum* and *Plotkin sum*. For $i \in \{1, 2\}$, let C_i be a binary code of length n_i. The direct sum of C_1 and C_2 is the binary code $C_1 \oplus C_2 = \{(c_1|c_2) : c_1 \in C_1, c_2 \in C_2\}$ [14]. The Plotkin sum of C_1 and C_2 with $n_1 = n_2$ is the binary code $C_1|C_2 = \{(c_1|c_1 + c_2) : c_1 \in C_1, c_2 \in C_2\}$ [2].

Proposition 11 (Plotkin Sum) *[5] Let C_1 and C_2 be two $(n, |C_1|, d_1)$ and $(n, |C_2|, d_2)$ binary codes with kernels K_{C_1} and K_{C_2}, and coset representatives $L_1 = \{v_1, \ldots, v_{t_1}\}$ and $L_2 = \{w_1, \ldots, w_{t_2}\}$, respectively. The Plotkin sum $C_1|C_2$ is the $(2n, |C_1| \cdot |C_2|, \min\{2d_1, d_2\})$ code with kernel $K_{C_1}|K_{C_2}$, and $(t_1+1)(t_2+1)-1$ coset representatives given by the set $\{(v|v+w) : v \in L_1 \cup \{\mathbf{0}\}, w \in L_2 \cup \{\mathbf{0}\}\} \backslash \{(\mathbf{0}, \mathbf{0})\}$.*

Proposition 12 (Direct Sum) *Let C_1 and C_2 be two $(n_1, |C_1|, d_1)$ and $(n_2, |C_2|, d_2)$ binary codes with kernels K_{C_1} and K_{C_2}, and coset representatives $L_1 = \{v_1, \ldots, v_{t_1}\}$ and $L_2 = \{w_1, \ldots, w_{t_2}\}$, respectively. The direct sum $C_1 \oplus C_2$ is the $(n_1 + n_2, |C_1| \cdot |C_2|, \min\{d_1, d_2\})$ binary code with kernel $K_{C_1} \oplus K_{C_1}$, and $(t_1+1)(t_2+1)-1$ coset representatives given by the set $\{(v|w) : v \in L_1 \cup \{\mathbf{0}\}, w \in L_2 \cup \{\mathbf{0}\}\} \backslash \{(\mathbf{0}, \mathbf{0})\}$.*

Proof Straightforward using the same arguments as in Proposition 11 [5]. □

Note that we can obtain the kernel and coset representatives of an extended binary code directly from the kernel and coset representatives of the code. The same happens for the direct sum and Plotkin sum constructions. For all other constructions, we obtain a partial kernel and the corresponding coset representatives. Although we cannot assure which are the final kernel and coset representatives in these cases, we can speed up the kernel computation by starting from a partial kernel and using the algorithms shown in Section 2.

4 Minimum Distance Computation

Using the representation given in Section 2, we can describe new algorithms to compute the minimum weight and minimum distance of a binary nonlinear code. Specifically, in this section, we present new algorithms based on computing the minimum weight of linear subcodes, using the known Brouwer-Zimmermann algorithm. Moreover, we study the performance of these algorithms, by giving an estimation of the number of enumerated codewords needed in the computations.

For linear codes, the best enumerative algorithm is the Brouwer-Zimmermann algorithm, which was first brought up by A. Brouwer in the 1980s and due to K.-H. Zimmermann [2,24,26]. This algorithm is based on the result given by Proposition 13. Let G be a generator matrix of a linear code K of dimension κ. Any set of κ coordinate positions such that the corresponding columns of G are linear independent is called an *information set* for K. A generator matrix G is said to be *systematic* on the information set I if the corresponding columns of I form a $\kappa \times \kappa$ identity matrix.

Proposition 13 *[24, 25] Let $G_1, \ldots, G_h$ be h systematic generator matrices of a linear code K of dimension κ over a finite field $\mathbb{F}_q$ such that they have pairwise disjoint information sets. For any $r < \kappa$, if $S_i = \{mG_i : m \in \mathbb{F}_q^k,\ wt(m) \leq r\}$ for each matrix G_i, then all $c \in C \setminus \bigcup_{i=1}^{h} S_i$ satisfy $wt(c) \geq h(r+1)$.*

The systematic matrices $G_1, \ldots, G_h$ can be obtained, for example, by applying Gaussian elimination to the column positions which are not contained in the information sets of the previous computed matrices. Moreover, note that in every step of the enumeration process, r rows of the h generator matrices are used. After each step, a lower bound $h(r+1)$ and an upper bound of the minimum weight, which is the minimum weight of the enumerated codewords, are obtained. When the upper bound is equal or smaller than the lower bound, the minimum weight of the linear code is obtained, without enumerating necessarily all codewords. This algorithm can be adapted in order to use systematic generator matrices with overlapping information sets. In this case, every column position can be included in a generator matrix. Moreover, since it uses more generator matrices, the lower bound can grow faster during the enumeration process. Recently, a new approach to efficiently find a sequence of systematic generator matrices having the maximum number of pairwise disjoint information sets has been presented [15].

Given a binary linear code K of length n and dimension κ, and a vector $v \in \mathbb{Z}_2^n \setminus K$, the linear span $K_v = \langle K, v \rangle = K \cup (K + v)$ of dimension $\kappa + 1$ is called *extended coset*.

Proposition 14 *Let $C = \bigcup_{i=0}^{t} \left(K_C + v_i\right)$ with $t \geq 2$, where $v_0 = \mathbf{0}$. The minimum weight of C can be computed as $\min\{wt(K_{v_i}) \ : \ i \in \{1, \ldots, t\}\}$, and the minimum distance of C as $\min\{wt(K_{v_i + v_j}) \ : \ i \in \{0, \ldots, t-1\},\ j \in \{i+1, \ldots, t\}\}$.*

Proof For $wt(C)$, note that the linear codes K_{v_i} include exactly all codewords of C. For $d(C)$, considering all different cases, the statement follows: If c_1, $c_2 \in K_C$ or c_1, $c_2 \in K_C + v_i$, then $d(c_1, c_2) = wt(c_1 + c_2) = wt(k)$, where $k \in K_C$. If $c_1 \in K_C$ and $c_2 \in K_C + v_i$, then $d(c_1, c_2) = wt(k + v_i)$. Finally, if $c_1 \in K_C + v_i$ and $c_2 \in K_C + v_j$ with $i \neq j$, then $d(c_1, c_2) = wt(k + v_i + v_j)$. □

Using the representation given in Section 2, Proposition 14, and the known Brouwer-Zimmermann algorithm to compute the minimum weight of a linear code, we can design

Algorithms 2 (MinW) and 3 (MinD) to compute the minimum weight and minimum distance of a binary nonlinear code, respectively. Note that the complexity of these two algorithms depends strongly on the number of coset representatives t and the complexity of the Brouwer-Zimmermann algorithm. For the minimum weight, we compute t times the minimum weight of a linear code K_v, and for the minimum distance, $\binom{t+1}{2}$ times. In order to study the efficiency of these algorithms, we compare them with the brute force method, which consists in enumerating all codewords.

Algorithm 2: Minimum weight computation (MinW)

Data: A binary code C given by the kernel K_C and coset representatives $L = \{v_1, \ldots, v_t\}$.
Result: The minimum weight $wt(C)$.
begin
  $wt(C) \leftarrow \text{Length}(C)$
  for $i \in [1, \ldots, t]$ **do**
    $K_{v_i} \leftarrow K_C \cup (K_C + v_i)$
    $wt(C) \leftarrow \min\{wt(K_{v_i}), wt(C)\}$
  return $wt(C)$

Example 1 *Let K be the $(30, 2^{12}, 9)$ binary linear code, given in* Magma *as the best known linear code of length 30 and dimension 12, that is, the linear code generated by*

$$\begin{pmatrix}
100000000100011011011000110011\\
010000000100010000100100001111\\
001000000100000010010010111100\\
000100000000000000111010011011\\
000010000100011101010100101000\\
000001000100010011110101111101\\
000000100000010101101100100010\\
000000010100001111101011110001\\
000000001100011100110010011101\\
000000000010010111011111000111\\
000000000001011111101000011000\\
000000000000100101110111101100
\end{pmatrix}.$$

Let $C = \bigcup_{i=0}^{3} \left(K_C + v_i\right)$, where $K_C = K$, $v_0 = \mathbf{0}$, and the coset representatives are

$$v_1 = (010101100011011000000000100100),$$
$$v_2 = (010001000110111101001100011111),$$
$$v_3 = (111101000011000111111000100001).$$

It is easy to check that $wt(C) = 6$ and $d(C) = 5$. The time of computing $wt(C)$ using brute force and Algorithm 2 (MinW) are 9.0×10^{-3} and 1.0×10^{-3} seconds, respectively. Note that sometimes a brute force calculation can be a faster way to obtain the minimum weight. On the other hand, the time of computing $d(C)$ using brute force and Algorithm 3 (MinD) are 123.4 and 2.0×10^{-3} seconds, respectively, so it is much faster to use Algorithm 3 (MinD) than brute force. All these tests have been performed in Magma *version V2.18-3, running on a server with an Intel Xeon processor (clock speed 2.40 GHz). For a better time comparison between both methods, the same internal* Magma *functions have been used.*

Algorithm 3: Minimum distance computation (MinD)

Data: A binary code C given by the kernel K_C and coset representatives $L = \{v_1, \ldots, v_t\}$.
Result: The minimum distance $d(C)$.
begin
  $d(C) \leftarrow \text{Length}(C)$
  for $i \in [0, \ldots, t-1]$ **do**
    for $j \in [i+1, \ldots, t]$ **do**
      $K_{v_i+v_j} \leftarrow K_C \cup (K_C + v_i + v_j)$
      $d(C) \leftarrow \min\{wt(K_{v_i+v_j}), d(C)\}$
  return $d(C)$

All these algorithms are based on the enumeration of codewords, adding together codewords and determining the minimum weight of these codewords. As it is defined in [24], the nature of these computations gives rise to a natural performance measure, which is referred to as *work*. One unit of work represents both an addition and the weight computation of a single bit position. An estimate of the total work an algorithm performs is referred to as *work factor*. Therefore, work factors provide us with a suitable tool for comparing the performance of algorithms based on enumeration. Of course, note that this measure does not take into account that, depending on the algorithm used to speed up the computation of the number of 1-bits in a vector and the addition of vectors, there can be jumps in the run time whenever the length of the code forces the algorithm to use more computer words.

Recall that a binary nonlinear code of length n with a kernel of dimension κ and t coset representatives has $M = 2^\kappa(t+1)$ codewords. Therefore, it is easy to see that the work factor for computing $wt(C)$ and $d(C)$ using brute force is, respectively,

$$n2^\kappa(t+1) \qquad \text{and} \qquad n\binom{2^\kappa(t+1)}{2}. \tag{2}$$

Lemma 1 *[24] Let K be a binary linear code of dimension κ and length n. Then, the work factor for computing $wt(K)$ using Brouwer-Zimmermann algorithms is*

$$(n-\kappa)\lceil n/\kappa \rceil \sum_{r=1}^{\bar{r}} \binom{\kappa}{r}, \tag{3}$$

where $\bar{r}$ is the smallest integer such that $\lfloor n/\kappa \rfloor(\bar{r}+1) + \max(0, \bar{r}+1-(\kappa - n \bmod \kappa)) \geq wt(K)$.

Let C be a binary nonlinear code of length n with kernel of dimension κ and coset representatives $L = \{v_1, \ldots, v_t\}$. By Proposition 14 and Lemma 1, it is easy to establish the work factor for computing $wt(C)$ and $d(C)$. Specifically, the work factor for computing $wt(C)$ using Algorithm 2 (MinW) is

$$\sum_{j=1}^{t} \left((n-\kappa-1)\lceil n/(\kappa+1) \rceil \sum_{r=1}^{\bar{r}_{0,j}} \binom{\kappa+1}{r} \right); \tag{4}$$

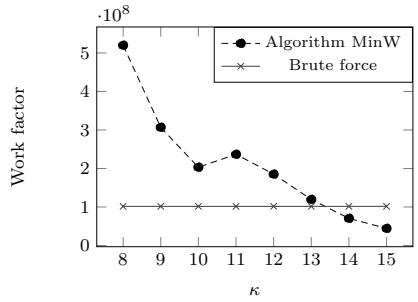

Fig. 1 Work factor for computing $wt(C)$ using Algorithm 2 (MinW) compared with brute force, for $(100, 2^{15} \cdot 31)$ binary codes with kernels of dimension $\kappa \in \{8, \ldots, 15\}$.

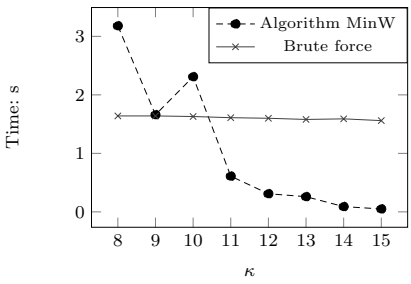

Fig. 2 Time of computing $wt(C)$ using Algorithm 2 (MinW) compared with brute force, for $(100, 2^{15} \cdot 31)$ binary codes with kernels of dimension $\kappa \in \{8, \ldots, 15\}$.

and for computing $d(C)$ using Algorithm 3 (MinD) is

$$\sum_{i=0}^{t-1} \Big(\sum_{j=i+1}^{t} \Big((n-\kappa-1)\lceil n/(\kappa+1) \rceil \sum_{r=1}^{\bar{r}_{i,j}} \binom{\kappa+1}{r} \Big) \Big), \tag{5}$$

where $\bar{r}_{i,j}$ is the smallest integer such that $\lfloor n/(\kappa+1) \rfloor (\bar{r}_{i,j}+1) + \max(0, \bar{r}_{i,j}+1-(\kappa+1-n \bmod (\kappa+1))) \geq wt(K_{v_i+v_j})$.

Example 2 *Let us consider random* $(100, 2^{15} \cdot 31)$ *binary codes* C *with kernels of dimension* $\kappa \in \{8, \ldots, 15\}$*, and random* $(100, 2^7 \cdot 31)$ *binary codes* C *with kernels of dimension* $\kappa \in \{3, \ldots, 7\}$*. Figures 1 and 2 show the work factors given by (4) and (2), and the real time cost, for computing* $wt(C)$ *using Algorithm 2 (MinW) and brute force, respectively. Equivalently, Figures 3 and 4 show the work factors given by (5) and (2), and the real time cost, for computing* $d(C)$ *using Algorithm 3 (MinD) and brute force, respectively.*

It can be seen from these figures that the work factors and real time cost follow the same trend. Moreover, keeping the same length and number of codewords, the time cost of using Algorithms 2 (MinW) and 3 (MinD) decreases sharply while the dimension of the kernel increases (or equivalently, while the number of cosets decreases). Note that when κ *is large, Algorithms 2 and 3 save a lot of time. More specifically, Algorithm 2 when* $\kappa = 15$ *and Algorithm 3 when* $\kappa = 7$ *use only* $1/31$ *and* $1/21$ *time compared with brute force, respectively.*

From Algorithms 2 (MinW) and 3 (MinD), it is easy to see that the weight of some codewords in the kernel K_C is computed several times, specifically, once for each $K_{v_i+v_j} = K_C \cup (K_C + v_i + v_j)$, where $i, j \in \{0, 1, \ldots, t\}$ and $i < j$. However, we will show that we can make a little adjustment, in order to avoid this repetition.

In Brouwer-Zimmermann algorithm, the enumerating process is divided into several steps. In the rth step, it enumerates all linear combinations of r rows of the generator matrix of $K_{v_i+v_j}$ of dimension $\kappa+1$, examines the minimum weight of each combination and compares it with the lower bound. In order to avoid enumerate some codewords several times, we can modify the previous algorithms and enumerate only the codewords

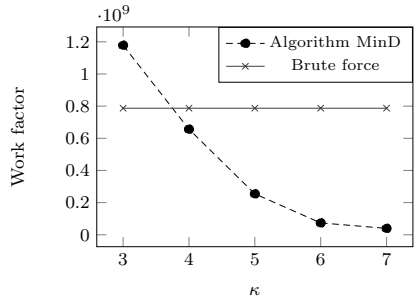

Fig. 3 Work factor for computing $d(C)$ using Algorithm 3 (MinD) compared with brute force, for $(100, 2^7 \cdot 31)$ binary codes with kernels of dimension $\kappa \in \{3, \ldots, 7\}$.

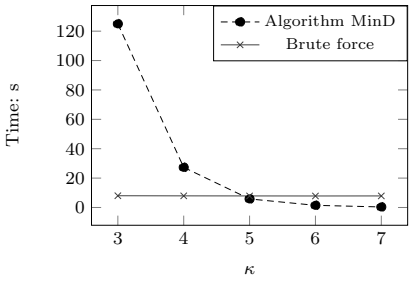

Fig. 4 Time of computing $d(C)$ using Algorithm 3 (MinD) compared with brute force, for $(100, 2^7 \cdot 31)$ binary codes with kernels of dimension $\kappa \in \{3, \ldots, 7\}$.

in each coset $K_C + v_i + v_j$. Then, in the rth step, we enumerate all linear combinations of r rows of the generator matrix of K_C of dimension κ and compute the weight of each combination adding the vector $v_i + v_j$. The codewords in the kernel are considered by adding the all-zero vector to the set of coset representatives. After this adjustment, the work factor using the improved Algorithms 2 and 3, which are referred as Algorithms IMinW and IMinD, respectively, is reduced as it is shown in the following proposition.

Proposition 15 *Let C be a binary nonlinear code of length n with kernel of dimension κ and coset representatives $L = \{v_1, \ldots, v_t\}$. The work factor for computing $wt(C)$ using improved Algorithm 2 (IMinW) is*

$$\sum_{j=0}^{t} \Big((n-\kappa)\lceil n/\kappa \rceil \sum_{r=1}^{\bar{r}_{0,j}} \binom{\kappa}{r} \Big); \tag{6}$$

and for computing $d(C)$ using improved Algorithm 3 (IMinD) is

$$\sum_{i=0}^{t-1} \Big(\sum_{j=i+1}^{t} \Big((n-\kappa)\lceil n/\kappa \rceil \sum_{r=1}^{\bar{r}_{i,j}} \binom{\kappa}{r} \Big) \Big) + (n-\kappa)\lceil n/\kappa \rceil \sum_{r=1}^{\bar{r}_{0,0}} \binom{\kappa}{r}, \tag{7}$$

where $\bar{r}_{i,j}$ is the smallest integer such that $\lfloor n/\kappa \rfloor (\bar{r}_{i,j}+1) + \max(0, \bar{r}_{i,j} + 1 - (\kappa - n \bmod \kappa)) \geq wt(K_{v_i+v_j})$. Note that $\bar{r}_{0,0} = \bar{r}$ given in Lemma 1.

The work factor for computing $wt(C)$ and $d(C)$ of a binary code C relies on the parameters $\bar{r}_{i,j}$, which depend on $wt(K_{v_i+v_j})$, and they may be different for any i, j. Therefore, it is impossible to estimate the work factor if only the values n, κ and t of the binary code C are given. However, we can consider an upper bound of the work factor, and from that be able to estimate easily the work factor for computing $wt(C)$ and $d(C)$. Since for any extended coset $K_{v_i+v_j}$ we have that $wt(K_{v_i+v_j}) \leq wt(K_C)$, we can obtain an upper bound for the previous given work factors by replacing $wt(K_{v_i+v_j})$ with $wt(K_C)$.

Proposition 16 *Let C be a binary nonlinear code of length n with kernel K_C of dimension κ and t coset representatives. An upper bound for the work factor for computing $wt(C)$ using improved Algorithm 2 (IMinW) is*

$$(t+1)(n-\kappa)\lceil n/\kappa \rceil \sum_{r=1}^{\bar{r}} \binom{\kappa}{r}; \tag{8}$$

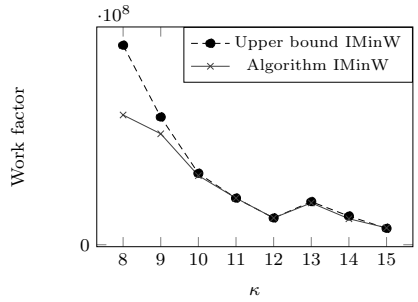

Fig. 5 Work factor and work factor upper bound for computing $wt(C)$ for $(100, 2^{15} \cdot 31)$ binary codes.

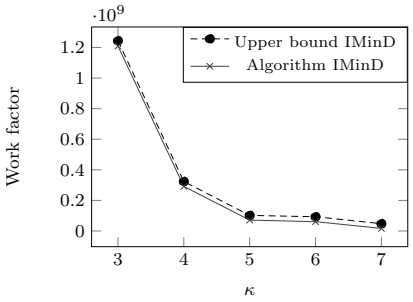

Fig. 6 Work factor and work factor upper bound for computing $d(C)$ for $(100, 2^7 \cdot 31)$ binary codes.

and for computing $d(C)$ using improved Algorithm 3 (IMinD) is

$$\left(\binom{t+1}{2}+1\right)(n-\kappa)\lceil n/\kappa\rceil \sum_{r=1}^{\bar{r}} \binom{\kappa}{r}, \tag{9}$$

where $\bar{r}$ is the smallest integer such that $\lfloor n/\kappa\rfloor(\bar{r}+1)+\max(0,\bar{r}+1-(\kappa-n \bmod \kappa)) \geq wt(K_C)$.

Proof Given n and κ, $f(r) = \lfloor n/\kappa\rfloor(r+1) + \max(0, r+1-(\kappa - n \bmod \kappa))$ is an increasing function. Let $\bar{r}_{i,j}$ be the smallest integer r such that $f(r) \geq wt(K_{v_i+v_j})$, that is, as in Proposition 15. Let $\bar{r}$ be the smallest integer r such that $f(r) \geq wt(K_C)$. Since $wt(K_{v_i+v_j}) \leq wt(K_C)$, $\bar{r}_{i,j} \leq \bar{r}$ and the result follows by Proposition 4. □

Example 3 *Considering the same binary codes as in Example 2, Figures 5 and 6 show the differences between the work factors and their upper bounds for computing the minimum weight and distance using improved Algorithms 2 (IMinW) and 3 (IMinD). Note that the upper bound of work factor is quite close to the work factor and is much easier to estimate, since we just need $wt(K_C)$ along with the values of n, κ and t of C. For the minimum distance and the considered codes, the difference between both work factors is very small, so both lines coincide in Figure 6.*

Example 4 *Considering again the same binary codes as in Example 2, Figures 7 and 8 show the upper bounds for the work factors for both algorithms presented in the paper and brute force. Through these examples, we can see the improvement on Algorithms 2 (IMinW) and 3 (IMinD).*

Note that the results on these upper bounds for the work factors allow to establish from which parameters of the given code, it is better to use the new presented algorithms instead of the brute force method.

5 Minimum Distance Decoding

For linear codes, any algorithm to compute the minimum weight can easily be applied to the decoding problem. For example, the algorithms described to attack the McEliece

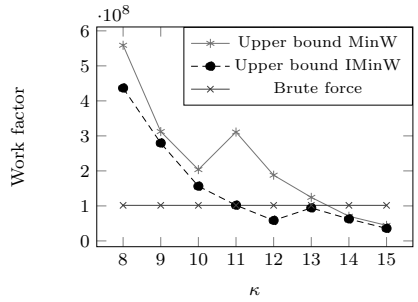

Fig. 7 Work factor upper bounds for computing $wt(C)$ for $(100, 2^{15} \cdot 31)$ binary codes.

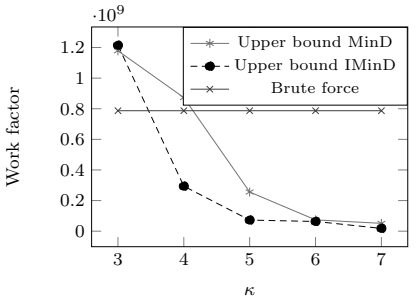

Fig. 8 Work factor upper bounds for computing $d(C)$ for $(100, 2^7 \cdot 31)$ binary codes.

public-key cryptosystem can be used to decode a general linear code [7,24]. In this section, we generalize this idea presenting a nontrivial general decoding method for nonlinear codes. This method uses the coset structure and any algorithm for finding a vector of minimum weight in a linear code or a coset of a linear code.

Let K be a binary linear code of length n, dimension κ and minimum distance d. The general decoding algorithm for linear codes is called *syndrome decoding* [16]. Before the decoding process starts, it is necessary to compute a syndrome table pairing each syndrome $s \in \mathbb{Z}_2^{n-\kappa}$ with an error vector e of minimum weight in the coset associated to that syndrome. Although creating the syndrome table is a one-time task, which is carried out before decoding the received vectors, sometimes it can be difficult to create and store it. Moreover, if it contains many elements, it can also be difficult to find the corresponding error vector from a given syndrome. In these cases, it is necessary to use another method, which is summarized in the following proposition [7,16,24].

Proposition 17 *Let K be a binary linear code with minimum distance d. For a received vector $u = c + e \notin K$, where $c \in K$, let $K_u = K \cup (K + u)$. If $wt(e) < d$, then u can be decoded as $c' = u - e' \in K$, where e' is a vector of minimum weight in K_u, so $wt(e) = wt(e')$. Note that if $wt(e) \leq \lfloor \frac{d-1}{2} \rfloor$, then $e' = e$ and $c' = c$.*

In this way, we can decode a received vector as long as less than d errors have been added to the transmitted codeword. When d or more than d errors occurs during the transmission, the vector of minimum weight in K_u could come from K, and then an error vector e' cannot be found by Proposition 17. Therefore, this method, called *coset decoding*, provides a complete decoding but only up to $d - 1$ errors. Note that if $\rho(K) \leq d - 1$, that is when K is maximal, we actually obtain a complete decoding.

Example 5 *Let K be the simplex code of length* 31*, dimension 5 and minimum distance 16 [16]. Figure 9 shows the time in seconds to decode random received vectors by using the coset and syndrome decoding, both implemented in* Magma *through the functions* `McEliecesAttack` *and* `Decode`*, respectively. Note that $\rho(K) = 15 = d - 1$, so K is maximal and both decoding methods perform a complete decoding. According to the implementation in* Magma*, the syndrome decoding uses 2836 MB of memory, and the coset decoding method uses a negligible amount of memory.*

Apparently, the coset decoding has a big advantage if the number of received vectors to be decoded is small. Since the syndrome table needs to be computed at the first

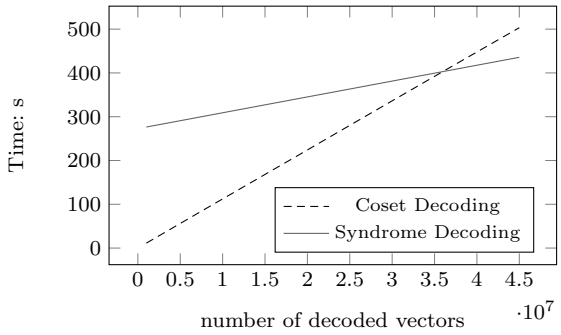

Fig. 9 Time for decoding using the binary simplex code of length 31.

decode procedure, it costs the most the first time. However, the biggest advantage of the coset decoding is on the memory usage. For the syndrome decoding, it is necessary to store a syndrome table to be used in the decoding process. If this syndrome table is too big to be stored, which will happen when the codimension of the code is moderately large, it will be impossible to decode by syndrome decoding. By contrast, the coset decoding based on computing the minimum weight of a linear code does not need to store anything significant (just the very few data needed for the enumeration process), which makes it especially useful for codes with a large codimension.

Example 6 *Let K be the simplex code of length* 63*, dimension 6 and minimum distance 32 [16]. As the code in Example 5, $\rho(K) = 31 = d - 1$, so it is maximal and both decoding methods allow to perform a complete decoding. By using the syndrome decoding,* Magma *returns "Runtime error in 'Decode': Code has too many cosets". However, in this case, we can still perform a complete decoding by using the coset decoding method, which takes* 11.58 *seconds to decode* 500000 *random received vectors and uses a negligible amount of memory.*

We can generalize the coset decoding seen for linear codes to decode nonlinear codes using their coset representation. The following proposition summarizes this decoding process for nonlinear codes.

Proposition 18 *Let C be a binary nonlinear code with minimum distance d, kernel K_C and coset representatives $\{v_1, \ldots, v_t\}$. For a received vector $u = c + e \notin C$, where $c \in C$, let $C_u = \bigcup_{i=0}^{t}(K_C \cup (K_C + v_i + u))$. If $wt(e) < d$, then u can be decoded as $c' = u - e' \in C$, where e' is a vector of minimum weight in C_u, so $wt(e) = wt(e')$. Note that if $wt(e) \leq \lfloor \frac{d-1}{2} \rfloor$, then $e' = e$ and $c' = c$.*

Proof For a received vector $u \in \mathbb{Z}_2^n$, in order to decode it, we look for a vector $e' \in \mathbb{Z}_2^n$ of minimum weight such that $u - e' \in C$. This is equivalent to find a vector e' of minimum weight in $C + u$. We have that $C + u = \bigcup_{i=0}^{t}(K_C + v_i + u)$. Let e_i be a vector of minimum weight in the linear code $K_C \cup (K_C + v_i + u)$ for all $i \in \{0, \ldots, t\}$. If $wt(e) < d$, then we can take e' as the vector of minimum weight in $\{e_i : i \in \{0, \ldots, t\}\}$, since it is also a vector of minimum weight in $C + u$ such that $wt(e) = wt(e')$. Then, the received vector u can be decoded as $c' = u - e' \in C$. □

Note that the performance of the coset decoding for nonlinear codes highly depends on the number of coset representatives. Moreover, any algorithm to find a vector of minimum weight in a linear code or a coset of a linear code can be applied.

Example 7 *Let C_κ be a $(31, 2^9 \cdot 5, 5)$ binary nonlinear code with a kernel of dimension κ, for $\kappa \in \{5, \ldots, 9\}$. The following table shows the time in seconds to decode 5000 random received vectors using the coset decoding for each one of the codes C_κ, $\kappa \in \{5, \ldots, 9\}$.*

κ	5	6	7	8	9
Time: s	63.30	30.35	18.53	12.57	10.82

6 Conclusions

In this paper, we have presented some results to represent, manipulate, store and construct binary nonlinear codes in an efficient way, mainly when the codes have a large kernel. Based on these results, we have developed some algorithms to compute the minimum weight and distance of these codes, along with algorithms to decode them. All these results can be easily generalized to q-ary nonlinear codes, that is, to subsets of $\mathbb{F}_q^n$, where $\mathbb{F}_q$ is a finite field with q elements. Just note that a q-ary nonlinear code can also be written as a union of cosets of K_C, where $K_C = \{x \in C : \lambda x + C = C, \ \forall \lambda \in \mathbb{F}_q\}$ [21]. Moreover, the minimum weight of $K_v = \langle K, v \rangle = \bigcup_{\lambda \in \mathbb{F}_q} (K + \lambda v)$ is equal to the minimum weight of $K \cup (K + v)$, and a vector of minimum weight in $K \cup (K + v)$ can be computed from one in K_v.

We established the relationship between the performance of these algorithms and the parameters of the code: length n, dimension of the kernel κ, and number of coset representatives t, in order to estimate and decide which algorithm to use depending on these parameters. These new algorithms are especially suitable for codes with a large kernel, while the brute force method works better for codes with a small kernel.

Most of the results and algorithms described in the paper have been implemented by the authors as a new package in Magma, which is available on the web page `http://ccsg.uab.cat` together with a manual describing all functions.

References

1. H. Bauer, B. Ganter, and F. Hergert, "Algebraic techniques for nonlinear codes," *Combinatorica*, vol. 3, pp. 21-33, 1983.
2. A. Betten, M. Braun, H. Fripertinger, A. Kerber, A. Kohnert, and A. Wassermann, *Error-Correcting Linear Codes: Classification by Isometry and Applications*, Berlin: Springer, 2006.
3. M. Borges-Qintana, M. A. Borges-Trenard, I. Márquez-Corbella, and E. Martínez-Moro, "Computing coset leaders and leader codewords of binary codes," to appear in the *Journal of Algebra and Its Applications*, 2014.
4. J. Borges, C. Fernández, J. Pujol, J. Rifà, and M. Villanueva, "$\mathbb{Z}_2\mathbb{Z}_4$-linear codes: generator matrices and duality," *Designs, Codes and Cryptography*, vol. 54, no. 2, pp. 167-179, 2010.
5. J. Borges and C. Fernández, "Plotkin construction: rank and kernel," *arXiv:0707.3878v1*, July 26, 2007.
6. I. Bouyukliev and V. Bakoev, "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, 2008.

7. A. Canteaut and F. Chabaud, "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 Trans. on Information Theory*, vol. 44, no. 1, pp. 367-378, 1998.
8. J. J. Cannon and W. Bosma (Eds.), *Handbook of* MAGMA *Functions*, Edition 2.13, 4350 pages, 2006. (`http://magma.maths.usyd.edu.au/magma/`)
9. K. Elssel and K.-H. Zimmermann, "Two new nonlinear binary codes," *IEEE Trans. on Information Theory*, vol. 51, no. 3, pp. 1189-1190, 2005.
10. M. Grassl, "Searching for linear codes with large minimum distance," in: W. Bosma and J. Cannon (Eds.) *Discovering Mathematics with Magma*, Springer, 2006.
11. E. Guerrini, E. Orsini, and M. Sala, "Computing the distance distribution of systematic non-linear codes," *Journal of Algebra and Its Applications*, vol. 9, no. 2, pp. 241-256, 2010.
12. A. R. Hammons, P. V. Kumar, A. R. Calderbank, N. J. A. Sloane, and P. Solé, "The $\mathbb{Z}_4$-linearity of kerdock, preparata, goethals and related codes," *IEEE Trans. on Information Theory*, vol. 40, no. 2, pp. 301-319, 1994.
13. O. Heden, "On perfect p-ary codes of length $p+1$," *Designs, Codes and Cryptography*, vol. 46, no. 1, pp. 45-56, 2008.
14. W. C. Huffman and V. Pless, *Fundamentals of Error-Correcting Codes*, Cambridge University Press, 2003.
15. P. Lisoněk and L. Trummer, "An extension of the Brouwer-Zimmermann minimum weight algorithm," presented at the *4th International Castle Meeting on Coding Theory and Applications*, Palmela, Portugal, 15-18 September, 2014.
16. F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*, Amsterdam: North-Holland Publishing Company, 1977.
17. M. Mohri and M. Morii, "On computing the minimum distance of linear codes," *Electronics and Communications in Japan*, Part 3, vol. 83, no. 11, pp. 32-42, 2000.
18. P. R. J. Östergård, "Two new four-error-correcting binary codes," *Designs, Codes and Cryptography*, vol. 36, no. 3, pp 327-329, 2005.
19. P. R. J. Östergård and O. Pottonen, "The perfect binary one-error-correcting codes of length 15: part I-classification," *IEEE Trans. on Information Theory*, vol. 55, no. 10, pp. 4657-4660, 2009.
20. P. R. J. Östergård, O. Pottonen, and K. T. Phelps, "The perfect binary one-error-correcting codes of length 15: Part IIproperties," *IEEE Trans. on Information Theory*, vol. 56, no. 6, pp. 2571-2582, 2010.
21. K. T. Phelps, J. Rifà, and M. Villanueva, "Kernels and p-kernels of p^r-ary 1-perfect codes," *Designs, Codes and Cryptography*, vol. 37, no. 2, pp. 243-261, 2001.
22. J. Pujol, F. Zeng, and M. Villanueva, "Representation, constructions and minimum distance computation of binary nonlinear codes," Proceedings in Aplications of Computer Algebra, Málaga, Spain, July 2-6, 2013.
23. A. Vardy, "The intractability of computing the minimum distance of a code," *IEEE Trans. on Information Theory*, vol. 43, no. 6, pp. 1757-1773, 1997.
24. G. White, *Enumeration-based Algorithms in Coding Theory*, PhD Thesis, University of Sydney, 2006.
25. G. White and M. Grassl, "A new minimum weight algorithm for additive codes," ISIT 2006, Seattle, USA, July 9-14, pp. 1119-1123, 2006.
26. K.-H. Zimmermann, "Integral Hecke Modules, Integral Generalized Reed-Muller Codes, and Linear Codes," Tech. Rep. 3-96, Technische Universität Hamburg-Harburg, 1996.