GENETIC ALGORITHMS WITH PERMUTATION-BASED REPRESENTATION FOR COMPUTING THE DISTANCE OF LINEAR CODES.

M. P. CUÉLLAR, J. GÓMEZ-TORRECILLAS, F. J. LOBILLO, AND G. NAVARRO

Abstract. Finding the minimum distance of linear codes is an NP-hard problem. Traditionally, this computation has been addressed by means of the design of algorithms that find, by a clever exhaustive search, a linear combination of some generating matrix rows that provides a codeword with minimum weight. Therefore, as the dimension of the code or the size of the underlying finite field increase, so it does exponentially the run time. In this work, we prove that, given a generating matrix, there exists a column permutation which leads to a reduced row echelon form containing a row whose weight is the code distance. This result enables the use of permutations as representation scheme in metaheuristics, in contrast to the usual discrete representation. This makes the search of the optimum polynomial time dependent from the base field. Experimental results have been carried out employing codes over fields with two and eight elements, which suggests that evolutionary algorithms with our proposed permutation encoding are competitive with regard to existing methods in the literature. As a by-product, we have found and amended some inaccuracies in the Magma Computational Algebra System concerning the stored distances of some linear codes.

1. Introduction

Coding Theory is one of the foundational pillars of Information Theory. In his seminal paper [1], Shannon looks for the fundamental limits on signal processing and communication issues. In a noisy channel, redundancy is added to a message in order to recover it whenever transmission errors occur. Shannon's paper relates the correction capability with the error probability of the channel. Given a finite alphabet $\mathcal{A}$, a block code of length n is a subset $\mathcal{C} \subseteq \mathcal{A}^n$. The encoding process assigns to each message a codeword, i.e. an element in $\mathcal{C}$, the codeword is transmitted via a noisy channel and the receiver infers the transmitted codeword from the received word. There are a lot of techniques that can be considered to achieve efficiently the latter task. Historically, a successful strategy consists in taking a finite field $\mathbb{F}$ as an alphabet, and endowing the codes with some algebraic structure. In this scenario, a linear (block) code is therefore a vector subspace of $\mathbb{F}^n$. Examples of linear codes are the well known Hamming, Reed-Muller, Reed-Solomon, Goppa, AG or LDPC codes, which have been successfully implemented in CD players, deep space transmissions, wireless communications and more [2].

The correction capability of a code depends on some distance measure, being the Hamming distance the most usual one. Thus, the computation of the minimum distance of a linear code, or equivalently the minimum weight of all non-zero codewords, becomes a primary task to achieve the error correction capability of the code. However, finding a word with minimum weight for a code is not an easy task. Vardy showed in [3] that the decision problem associated to the computation of the minimum distance of a binary linear code is NP-complete. Hence, unless P = NP, we cannot expect to find a general polynomial time algorithm to compute the distance of an arbitrary linear code. For this reason, some particular codes have been designed using higher algebraic structures to obtain lower bounds of the minimum distance and therefore to ensure a good enough correction capability. For instance, different types of cyclicity have been introduced for linear codes, as it happens in BCH [4] or Reed-Solomon codes [5].

An approach to the general problem is to design algorithms that set the problem forth as an optimization task and make a search over the solution space by means of some type of heuristic. In this category, the designed algorithms attempt to find bounds for the code distance. The most known algorithm was designed by Brouwer and Zimmermann [6] and later extended by Lisonek and Trummer [7]. It is an exact algorithm, and attempts to build information sets from the original generating matrix. Since the algorithm is exact, the complete solution space has to be explored in the worst case, although the algorithm can be exploited effectively for small codes. However, as the code dimension or field size increase, the algorithm's time complexity makes its application infeasible.

On then other hand, approximate algorithms have also been considered previously to find the minimum distance of a linear code. For instance, Leon [8] provides an algorithm which gives probabilities for upper and lower bounds. Metaheuristics have also been used to solve the problem. The first attempts we have found in the literature are the use of Genetic Algorithms [9] and Simulated Annealing [10] to find upper bounds of the code distance. Later, Brand [11] proposes an Ant Colony Optimization method, with a Tabu Search procedure as local search intensification, providing promising results over QR and BCH codes in his experiments. One of the latest approaches, for generic linear codes, belongs to Askali et al. [12]. They design a genetic algorithm to perform the search, and the Multiple Impulse Method (MIM), which is an heuristic that raises the possibility of making random perturbations around nearest zero codewords to find linear combinations that yields an upper bound. The algorithm is tested over a high variety of QR, BCH and double cyclic codes and it is able to obtain the exact distance up to length 512 in a relatively low computational time. Genetic algorithms have also been employed to exploit properties of some types of codes to improve the search, as in [13] for extended QR codes. Other interesting approaches attempt to design the code subject to an established lower bound of the distance by means of Genetic Algorithms, as in [14].

The reader may notice that, even if the problem of finding the minimum distance is hard, there is not a wide variety of strategies that address the general problem, although there are studies that take into consideration specific properties of sub-families of codes that make tackling the problem effectively. As our interest in this manuscript is to cope with general linear codes, we highlight the following features of the aforementioned ways that will help us to show the contribution of this work: 1) Almost all studied methods focus on the case of binary codes; 2) the solution representation scheme in metaheuristics is therefore the binary encoding, or discrete encoding at most; and 3) the solution representation depends on the size of the alphabet, which is usually the field $\mathbb{F}_2$ with two elements as it has been mentioned. However, codes over larger fields also appear in many applications, see e.g. [15] for video streaming. In this work, we prove that, given a generating matrix, there exists a column permutation which leads to a reduced row echelon form containing a row whose weight is the code distance. This enables the possibility of exploiting the order (permutation) encoding as the representation scheme in search algorithms to solve the addressed problem. Thus, a solution in the proposed algorithms encodes a column permutation of the generating matrix. This representation is independent of the size of the finite field, so that it can applied to linear codes over any finite field. Also, our experiments suggest that this change in the solution representation allows us to overcome local optima solutions and speed up the search of the minimum distance with respect to the traditional discrete representation.

The article is structured as follows: Section 2 contains background concepts on mathematics and coding theory to make this article self-contained and introduce the notation. After that, Section 3 proposes the permutation representation as a solution encoding for the problem of finding the minimum distance of linear codes using metaheuristics. Section 4 describes both the baseline and proposed algorithms used to solve the problem, and Section 5 shows the experiments performed. Section 6 extends the experimentation for specific cases for which we have found new upper bounds for the code distance, and Section 7 contains the conclusions.

2. Background

The mathematical background of this paper relies in several aspects of finite fields, linear algebra and coding theory, which are revised in this section to fix the notation and for self-completeness.

2.1. Finite fields.

Finite fields, also called Galois fields, are natural generalizations of prime fields $\mathbb{F}_p = \mathbb{Z}/(p)$, the ring of integers modulo a prime number p. The elements of a finite field $\mathbb{F}_q$ of $q = p^r$ elements can be seen as polynomials over $\mathbb{F}_p$ of degree less than r, and the arithmetic is performed modulo an irreducible polynomial of degree r. For example, the field $\mathbb{F}_8$ with eight elements can be presented as polynomials in $\mathbb{F}_2[a]$ of degree less than 3, and in this paper the field operations are done modulo the polynomial $a^3 + a + 1$. The sum is the usual sum of polynomials, whilst the product is obtained by performing the usual product over polynomials modulo $a^3 + a + 1$. For example, the product of $(a^2 + 1)$ and $(a^2 + a + 1)$ in $\mathbb{F}_8$ is calculated as follows:

$$(a^2 + 1)(a^2 + a + 1) = a^4 + a^3 + a + 1 \bmod a^3 + a + 1 = a^2 + a,$$

or in bit notation

$$101 \cdot 111 = 110.$$

Non zero elements of a finite field form a cyclic group, i.e. they can be represented as powers of an element (called primitive). Primitive elements allow a more efficient multiplication. The following table shows that $a \in \mathbb{F}_8$ is a primitive element.

Power	Polynomial	Bits
a^0	1	001
a^1	a	010
a^2	a^2	100
a^3	$a+1$	011
a^4	a^2+a	110
a^5	a^2+a+1	111
a^6	a^2+1	101
a^7	1	001

The product can be computed using the corresponding primitive representation and that $a^7 = 1$. For instance,

$$(a^2 + 1)(a^2 + a + 1) = a^6 a^5 = a^{11 \bmod 7} = a^4 = a^2 + a.$$

A good resource about finite fields is [16].

2.2. **Reduced row echelon form.** Let $\mathbb{F}$ be a field and let $\mathbb{F}^{k\times n}$ denote the set of $k\times n$ matrices over $\mathbb{F}$. Recall that $H \in \mathbb{F}^{k\times n}$ is in reduced row echelon form (RREF for short) if all zero rows are below the non zero rows, the pivot (the first non zero element) in a non zero row is 1, the pivot of a non zero row is strictly to the right of the pivot of the row above, and each column containing a pivot has zeros everywhere else. The notion of echelon form can be found in almost every Linear Algebra book. We suggest [17, Chapter 5] for additional reading. For completeness we recall some basic facts about the RREF because they are essential for the mathematical basis of our algorithms. Two matrices $M, N \in \mathbb{F}^{k\times n}$ are said to be row equivalent if there exists a non singular matrix $S \in \mathbb{F}^{k\times k}$ such that $M = SN$. Hence:

(1) Two matrices $M, N \in \mathbb{F}^{k\times n}$ are row equivalent if and only if their rows generate the same vector subspace of $\mathbb{F}^n$.
(2) Each $M \in \mathbb{F}^{k\times n}$ is row equivalent to a matrix $H \in \mathbb{F}^{k\times n}$ in RREF.
(3) If $H, H' \in \mathbb{F}^{k\times n}$ are in RREF and row equivalent, then $H = H'$.
(4) If $v \in \mathbb{F}^n \setminus \{0\}$ belongs to the vector subspace generated by the rows of $M \in \mathbb{F}^{k\times n}$, then M is row equivalent to a matrix $\left(\frac{M'}{v}\right)$, where $M' \in \mathbb{F}^{(k-1)\times n}$.

As a consequence, any matrix is row equivalent to an unique matrix in RREF. Moreover, the non zero rows of the RREF are a basis for this subspace.

2.3. **Linear error correcting codes.** Let $\mathbb{F}_q$ be the field with q elements. An $[n, k]_q$-linear code $\mathcal{C}$ is a k-dimensional vector subspace of $\mathbb{F}_q^n$. The elements in $\mathcal{C}$ are called codewords. A generating matrix for $\mathcal{C}$ is any matrix $G \in \mathbb{F}_q^{k\times n}$ such that $\mathcal{C} = \{mG \ ; \ m \in \mathbb{F}_q^k\}$, i.e. each codeword can be obtained in a unique way multiplying a message to be encoded $m \in \mathbb{F}^k$ by the matrix G. The rows of G form a basis of the code. The (minimum) distance of $\mathcal{C}$ is $d = \min\{\mathrm{w}(c) \ ; \ c \in \mathcal{C} \setminus \{0\}\}$, where $\mathrm{w}(c)$ denotes the number of non zero entries in c, i.e. the Hamming weight of c. If a lower bound d of the distance is known, we refer to $\mathcal{C}$ as an $[n, k, d]_q$-linear code. The distance of a code is linked to its correction capability. Concretely, if $t = \lfloor \frac{d-1}{2} \rfloor$, the balls centered in the codewords with radius t are pairwise disjoint, so, if $y = c + e$ with $c \in \mathcal{C}$ and $\mathrm{w}(e) \le t$, then c is the closest codeword of y. A decoding algorithm finds the closest codeword of a given word.

Example 1. Let $\mathcal{C}$ be the $[6, 3]_8$-linear code defined by the generating matrix

$$G = \begin{pmatrix} 1 & 0 & 0 & a^5 & a^2 & a^4 \\ 0 & 1 & 0 & a & a^5 & 1 \\ 0 & 0 & 1 & 0 & a^5 & a^5 \end{pmatrix}$$

The reader may check that $(1, a^4, 0, 0, 0, 0)$ is a codeword of minimum weight. Hence $\mathcal{C}$ is a $[6, 3, 2]_8$-linear code. Observe that no row of G, which is in RREF, has Hamming weight 2.

The computation of the minimum distance, and a corresponding non zero codeword of minimal weight, is therefore a major issue for a linear code. As shown in [3, Theorem 5], finding the distance of a linear code is an NP-hard problem. Hence, unless P = NP, we cannot expect to design an efficient algorithm to compute the distance of an arbitrary linear code. To overcome this limitation, most codes are constructed with additional algebraic structures in order to fix a known lower bound of their distances. Examples can be found in [18, 19, 20] for cyclic codes and [21] for skew cyclic codes. However, in this paper we assume linear codes that are not constrained by any additional structure, and therefore our approach can be used for any family of linear codes.

3. Permutation representation scheme

Let $0 < k \le n$ be two non negative integers, and $\mathbb{F}_q$ the field with q elements. We denote by $\mathcal{S}_n$ the set of permutations of n symbols. Two $[n, k]_q$-linear codes $\mathcal{C}_1$ y $\mathcal{C}_2$ are permutation equivalent if there exists a permutation $x \in \mathcal{S}_n$ such that $(c_0, c_1, \dots, c_{n-1}) \in \mathcal{C}_1$ if and only if $(c_{x(0)}, c_{x(1)}, \dots, c_{x(n-1)}) \in \mathcal{C}_2$. It is easy to see that G is a generating matrix for $\mathcal{C}_1$ if and only if GP_x is a generating matrix for $\mathcal{C}_2$, where P_x is the permutation matrix associated to x. Although permutation equivalent codes could be different, they have the same minimum distance, since right multiplication by P_x preserves the Hamming weight.

Our new perspective to compute the minimum distance relies in the the following result, whose proof uses the properties of the RREF stated in subsection 2.2. See also [22].

Theorem 2. *Let G be a $k \times n$ generating matrix of a $[n, k]_q$-linear code $\mathcal{C}$ over the finite field $\mathbb{F}_q$. There exists a permutation $x \in \mathcal{S}_n$ such that the RREF, R, of GP_x, where P_x is the permutation matrix of x, satisfies that the Hamming weight of some of its rows reaches the minimum distance of $\mathcal{C}$. Consequently, if b is a row of R verifying such property, then bP_x^{-1} is a non zero codeword of $\mathcal{C}$ with minimal weight.*

Proof. Let d be the minimum distance of $\mathcal{C}$ and let $a \in \mathcal{C}$ be a codeword such that $\mathrm{w}(a) = d$. Since a is non zero, there exists a non singular matrix $A \in \mathbb{F}_q^{k \times k}$ such that

$$AG = \left(\frac{G_1}{a} \right)$$

for some $G_1 \in \mathbb{F}_q^{(k-1)\times n}$. Since $\mathrm{w}(a) = d$, there exists a permutation $x \in \mathcal{S}_n$ moving the non zero entries of a to the last positions, i.e.

$$aP_x = (0, \dots, 0, a_1, \dots, a_d),$$

where $a_1, \dots, a_d \in \mathbb{F}_q \setminus \{0\}$. Hence

$$AGP_x = \left(\frac{G_1 P_x}{aP_x} \right) = \left(\begin{array}{c|c} \multicolumn{2}{c}{G_1 P_x} \\ \hline 0 \cdots 0 & a_1 \cdots a_d \end{array} \right).$$

Now, let $A' \in \mathbb{F}_q^{(k-1)\times(k-1)}$ invertible such that $A' G_1 P_x = H'$ is in RREF. Hence,

$$\left(\begin{array}{c|c} A' & 0 \\ \hline 0 & 1 \end{array} \right) AGP_x = \left(\begin{array}{c|c} A' & 0 \\ \hline 0 & 1 \end{array} \right) \left(\begin{array}{c|c} \multicolumn{2}{c}{G_1 P_x} \\ \hline 0 \cdots 0 & a_1 \cdots a_d \end{array} \right) = \left(\begin{array}{c|c} \multicolumn{2}{c}{H'} \\ \hline 0 \cdots 0 & a_1 \cdots a_d \end{array} \right) \tag{1}$$

Since G_1 has rank $k-1$, the last row of H' is nonzero. Suposse that the pivot of this row is in the i_0-th column. If $i_0 < n-d+1$, then the last row of (1) is the last row of the RREF of GP_x up to non zero scalar multiplication, and we are done. Otherwise, the last two rows of (1) are linearly independent and their nonzero coordinates are placed at the last d coordinates. Hence, there exists a linear combination of both whose hamming weight is lower than d, a contradiction. The last statement is straightforward. □

By Theorem 2, finding the minimum distance of an $[n, k]_q$-linear code is reduced to find the minimum of the map $\mathfrak{d} : \mathcal{S}_n \to \mathbb{N}$ defined by

$$\mathfrak{d}(x) = \min \{\mathrm{w}(b) \mid b \text{ is a row of the RREF of } GP_x\}, \tag{2}$$

the *fitness* of the permutation x. This encoding is then invariant with respect to the base field. Obviously, the computation of $\mathfrak{d}(x)$, for some permutation x, does depend on q and n. However the row reduction can be performed in $\mathcal{O}(n^3)$ operations in $\mathbb{F}_q$ using the classical scholar algorithm. In fact, using fast matrix multiplication and the so called striped matrix reduction, the complexity could be reduced to $\mathcal{O}(n^{\log_2 7})$ and $\mathcal{O}(n^3/\log n)$, see [23, 24], respectively.

Anyway, there are just a few permutations which provide the minimum of $\mathfrak{d}$. Concretely, assume there is only one codeword which achieves the minimum weight d of $\mathcal{C}$ up to scalar multiplication. Then the probability of finding the minimum of $\mathfrak{d}$ by a random search is

$$\frac{d!\,(n-d)!}{n!} = \binom{n}{d}^{-1},$$

since, given a permutation $x \in \mathcal{S}_n$ such that $aP_x = (0, \dots, 0, a_1, \dots, a_d)$ for a codeword $a \in \mathcal{C}$ of minimal weight $\mathrm{w}(a) = d$, any other permutation of the last d and the first $n-d$ positions of aP_x also yields a RREF whose last row reaches the minimum weight. In the general case we obtain a lower bound of the probability of finding the minimum of $\mathfrak{d}$ by random search. There are codes with just one codeword of minimum weight (up to scalar multiplication), hence this lower bound can be achieved.

4. Algorithms

In this section we design a generational genetic algorithm and a CHC evolutionary algorithm with traditional discrete representation as baseline methods for comparison, and we adapt these algorithms to the new order representation scheme proposed in Section 3.

4.1. **Generational Genetic Algorithm with discrete representation (GGA-Discrete).** The GGA-Discrete algorithm follows the procedure designed in Algorithm 1, and it is the classic genetic algorithm with elitism and reinitialization [25].

As we have mentioned previously in the introduction, the representation scheme used to the date in the literature is the discrete representation. Suppose a linear code $\mathcal{C}$ of length n and dimension k over the finite field with q elements $\mathbb{F}_q$ is given by a generating matrix $G \in \mathbb{F}_q^{k\times n}$. The representation of a solution x is a vector of k elements, where each component belongs to $\mathbb{F}_q$. The fitness of the solution x is calculated as the number of non-zero components of the resulting word xG, i.e. its weight, except for the trivial non-valid solution $(0, \dots, 0) \in \mathbb{F}_q^k$. Assuming that the arithmetic over the finite field is in $\mathcal{O}(1)$, then the fitness calculation is in $\mathcal{O}(kn)$.

The algorithm works as follows: A population $P(t)$ is initialized and evaluated with N random solutions at iteration $t = 0$. Then, the main loop of the algorithm is executed until a stopping condition is met. In this paper, the stopping criterion is to evaluate a maximum number of solutions so that the algorithm can be compared in performance. In order to test the algorithms, an additional stopping criterion has been included when a solution in the population reaches a previously known lower bound of the distance.

The loop of the algorithm starts by selecting N parents according to the binary tournament selection operator [26]. A crossover operator is applied to two parents to generate a pair of new solutions with probability p_c. If they are not combined, the mutation operator acts on the parents to generate two mutated solutions. Several known crossover and mutation operators for discrete representation [27] can be selected, i.e. one point, two points, uniform crossover, random mutation by gene, etc. We performed a pre-experimentation to find out the best crossover and mutation settings for our experiments, and we have concluded that the uniform crossover and random mutation with mutation probability by gene are the best choices, as it is discussed in the experimental section.

All the N new solutions generated by either crossover or mutation form the population at the next iteration $P(t+1)$. After that, we check if all solutions in $P(t+1)$ are valid and, if not, each non-valid solutions is replaced with a new random solution. Finally, the solutions in $P(t+1)$ are evaluated. An elitism component is included before the next iteration starts: If no solution in $P(t+1)$ has a fitness equal or better than the best in $P(t)$, then then worst in $P(t+1)$ is replaced with the best in $P(t)$. Also, as the algorithm shows a high selective pressure towards the non-valid solution $(0,\ldots,0) \in \mathbb{F}_q^k$, we include a reinitialization of $P(t+1)$ with N new random solutions after $MaxReinit$ solution evaluations with no improvement in the fitness of the best solution found.

Algorithm 1 Generational Genetic Algorithm

Require: $N =$ Even natural number with the size of the population
Require: $p_c =$ Crossover probability
Require: $MaxReinit =$ Number of solution evaluations with no fitness improvement before reinitialization
Set t=0
Initialize $P(t)$ with N random valid solutions
Evaluate solutions in $P(t)$
while stopping criterion not satisfied **do**
  Set $P(t+1) = \emptyset$
  Set $parents(1..N)$= Selection of N solutions in $P(t)$ with binary tournament selection
  for i in $0..N/2-1$ **do**
    if random number from uniform distibution in $[0,1]$ is less than p_c **then**
      Set c_1, c_2= solutions generated from crossover over $parents(2i+1)$ and $parents(2i+2)$
    else
      Set c_1= mutation of $parent(2i+1)$, c_2= mutation of $parent(2i+2)$
    end if
    if c_1 (resp. c_2) is not valid **then**
      replace c_1 (resp. c_2) with a random valid solution
    end if
    Update $P(t+1) = P(t+1) \bigcup \{c_1, c_2\}$
  end for
  Evaluate solutions in $P(t+1)$
  if no solution fitness in $P(t+1)$ is better or equivalent to the best solution fitness in $P(t)$ **then**
    Replace worst solution in $P(t+1)$ with the best solution in $P(t)$
  end if
  if $MaxReinit$ solutions were evaluated with no improvement regarding the best solution in $P(t+1)$ **then**
    Replace solutions in $P(t+1)$ with $N-1$ random solutions and the best solution in $P(t)$
  end if
  Update $t = t+1$
end while
return Best solution in $P(t)$

4.2. **CHC algorithm with discrete representation (CHC-Discrete).** The Cross generational elitist selection, Heterogeneous recombination, and Cataclysmic mutation algorithm (CHC) is an evolutionary algorithm whose initial version was proposed for binary encoding [28]. This algorithm holds a balance between genotypic diversity in the solutions of the population, and convergence to local optima. It is based on four main components: elitist selection, the HUX solution recombination operator, an incest prevention check to avoid the recombination of similar solutions, and a population reinitialization method when a local optimum is found. Later versions of this algorithm are proposed for real and permutation encoding in [29, 30, 31].

An adaptation of this algorithm was designed and implemented in this work, with the goal of preventing premature convergence when the discrete representation is considered. We also use it to compare the performance of the discrete and the proposed order representation with algorithms that control the diversity in the population. Algorithm 2 describes the pseudocode, where the procedure $distance(x, y)$ computes the Hamming distance between two solutions x and y. The implemented crossover for the experimentation is the uniform crossover.

Algorithm 2 CHC Algorithm

Require: N = Even natural number with the size of the population
Require: τ = Crossover threshold update rate
Set t=0
Initialize $P(t)$ with N random valid solutions
Evaluate solutions in $P(t)$
Set d = *Average distance of solutions in* $P(t)$
Set $dec = \tau \times$ *Maximum distance of solutions in* $P(t)$
while stopping criterion is not satisfied **do**
    Set $C(t) = \emptyset$
    Set $parents(1, \dots, N)$ = random shuffle of solutions in $P(t)$
    for $i = 0, \dots, N/2 - 1$ **do**
        if $distance(parents(2i + 1), parents(2i + 2)) < d$ **then**
            Set c_1, c_2 = solutions generated from crossover over $parents(2i + 1)$ and $parents(2i + 2)$
            Update $C(t) = C(t) \cup \{c_1, c_2\}$
        end if
    end for
    Evaluate Solutions in $C(t)$
    Set $P(t + 1)$ = Best N solutions in $C(t) \cup P(t)$
    if $P(t) = P(t + 1)$ **then**
        Update $d = d - dec$
        if $d \leq 0$ **then**
            Initialize $P(t + 1)$ with the best solution of $P(t)$ and $N - 1$ random solutions.
            Evaluate new solutions in $P(t + 1)$
            Set d = *Average distance of solutions in* $P(t + 1)$
            Set $dec = \tau \times$ *Maximum distance of solutions in* $P(t + 1)$
        end if
    end if
    $t = t + 1$
end while
return Best solution in $P(t)$

The algorithm starts by initializing a population $P(t)$ with N random solutions. Then, the average and maximum distances between all solutions are computed. The crossover threshold d is assigned to the average distance, and a threshold update rate dec is initialized to τ multiplied by the maximum distance, where $\tau \in [0, 1]$ is an update rate, an input parameter to the algorithm. The main loop of the algorithm finishes when the aforementioned stopping condition is met. It works as follows: Firstly, the solutions in $P(t)$ are randomly shuffled and matched by pairs. These pairs of solutions are the parents to be combined. Then, the crossover operator is applied to each pair of parents to generate two offsprings, only if the distance between the two parents is not under the distance threshold d. If so, the offsprings are evaluated, and the population at the next iteration $P(t+1)$ is created containing the best N solutions coming from $P(t)$ and the new generated solutions by crossover. If $P(t+1)$ is the same as $P(t)$, the crossover threshold d is decreased by dec. Only when d is zero or under zero, the population is reinitialized. In our work, the reinitialization procedure replaces $P(t + 1)$ with $N - 1$ randomly generated solutions, and the best solution in $P(t)$. The values d and dec are recalculated for this new population.

4.3. **Generational genetic algorithm and CHC with order representation (GGA-Order and CHC-Order).** We specify the GGA and CHC procedures described in Algorithm 1 and Algorithm 2 to evolve solutions with the order (permutation) representation proposed in this Section 3, with a novel crossover described in this section. In these algorithms, solutions are permutations encoded as arrays of size n, where each array contains a column permutation of the initial generating matrix G.

The fitness of a solution $x \in \mathcal{S}_n$ is its image under $\mathfrak{d} : \mathcal{S}_n \to \mathbb{N}$ described in (2). Three steps are then performed to find such fitness: 1) Calculate GP_x, the matrix obtained after permuting the columns of G following the permutation x; 2) Calculate R, the RREF of GP_x; and 3) Find the row of R with minimum weight.

Classic crossover operators do not consider the group structure of the set of all permutations $\mathcal{S}_n$. Intuitively, for a permutation, the more non pivot columns are moved to the first positions, the better fitness it has. Therefore, one could expect that the composition of permutations with good fitness, may produce a solution with better fitness. Additionally, since two (or more) random permutations probably form a generator system [32, Theorem 1], the whole space of solutions is reached by their composition. In this paper we propose the algebraic crossover AX_r with $r > 1$: Given r solutions from the population $\{x_1, x_2, \ldots, x_r\}$, we construct the set of permutations given by their composition in any order, that is,

$$\mathcal{T} = \{x_{\tau(1)} \circ x_{\tau(2)} \circ \cdots \circ x_{\tau(r)} \text{ such that } \tau \in \mathcal{S}_r\}.$$

From this set, we select the r solutions with lower image under $\mathfrak{d}$ (that is, with better fitness) which replace the original r solutions. Therefore, taking r a divisor of n, the algebraic crossover operator AX_r partitions the population into subsets of r elements and, for each subset, with a given probability p_c, it recombines the elements as described above. Obviously, the crossover complexity increases exponentially with the parameter r. In the experiments performed in this paper, we use a fixed value $r = 2$.

In the mutation step, we follow a standard mutation operator: the 2-swap mutation, i.e. the composition with a transposition. Nevertheless, permuting two columns which are beyond the last pivot does not modify the fitness. For reasons of efficiency, we simply choose a column from the first k columns and other from the remaining $n - k$ columns in the generating matrix. The mutation operator is then applied to those solutions that were not recombined with the crossover operator.

Figure 1 illustrates an example of the AX_2 crossover operator and the generational relieve for a population of 4 solutions. The code $\mathcal{C}$ under consideration is the $[6, 3]_8$-linear code with generating matrix

$$G = \begin{pmatrix} a^5 & 0 & a^5 & a^6 & a & 0 \\ a^4 & a & 1 & 0 & a & a^2 \\ a^5 & a^4 & a^6 & a^4 & a^2 & 1 \end{pmatrix} \in \mathbb{F}_8^{3\times 6}.$$

We set $p_c = 0.5$. We have marked with yellow color the best solution of the population.

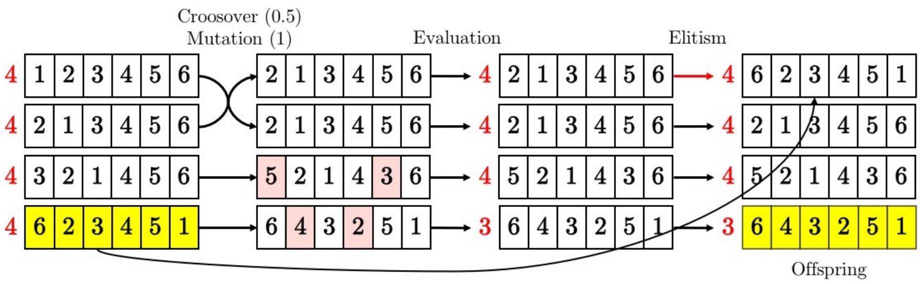

Figure 1. Construction of the next generation

5. Experimentation

We consider 30 different datasets to test our approach. The first 20 are generating matrices of linear codes over $\mathbb{F}_8$, from the database Magma [33], a specific software for computer algebra, using the command `BKLC` (an acronym of *Best Known Linear Code*). They are labelled as $(n, k, d_1 - d_2)$, where n is the length, k the dimension, d_1 a lower bound and d_2 an upper bound for the distance. The other 10 datasets are generating matrices of codes over $\mathbb{F}_2$. Five of them are the generating matrices of narrow sense BCH (Bose–Chaudhuri–Hocquenghem) taken from Magma with the command `BCHCode`. They are labelled BCH(n, k, δ), where n, k are the length and the dimension, and δ is the designed distance, a lower bound established during the code design. Finally, remaining five datasets are the generating matrices of EQR (Extended Quadratic Residues) codes. These matrices were obtained adding a parity-check bit to the generating matrices of suitable Quadratic Residues codes obtained from Magma via the command `QRCode`. They are labelled EQR(n, k, d) where n, k denote the length and dimension, and d the code distance obtained from [8].

5.1. **Experimental design.** The experiments were conducted in two stages: In the first one (Section 5.2), the order representation proposed in this work is compared with the classic discrete representation. The objective of such study is to test if the genetic algorithms using the order representation outperform the ones with discrete representation, regarding the quality of the results for codes over $\mathbb{F}_8$. The second phase (Section 5.3) compares the results of our proposal with some state-of-the-art methods existing in the literature. More specifically, the baseline method employed for comparison is published in [12]. As no previous study was found over a field greater than $\mathbb{F}_2$, only the EQR and BCH codes are contemplated in this stage. Section 5.4 discusses the results and describes the conclusions obtained.

Field	(n,k,d_1-d_2)	(n,k,d_1-d_2)	(n,k,d_1-d_2)	(n,k,d_1-d_2)	(n,k,d_1-d_2)
$\mathbb{F}_8$	$(30,16,10-13)$	$(30,18,9-11)$	$(45,22,15-21)$	$(45,24,14-19)$	$(45,26,12-17)$
$\mathbb{F}_8$	$(45,28,11-15)$	$(60,28,21-28)$	$(60,30,20-27)$	$(60,32,19-25)$	$(60,34,17-23)$
$\mathbb{F}_8$	$(75,30,28-40)$	$(75,35,24-35)$	$(75,40,20-31)$	$(75,45,17-27)$	$(90,19,49-63)$
$\mathbb{F}_8$	$(90,50,21-35)$	$(90,60,16-26)$	$(90,70,11-17)$	$(130,85,23-40)$	$(130,95,18-30)$
	$\mathrm{EQR}(n,k,d)$	$\mathrm{EQR}(n,k,d)$	$\mathrm{EQR}(n,k,d)$	$\mathrm{EQR}(n,k,d)$	$\mathrm{EQR}(n,k,d)$
$\mathbb{F}_2$	$\mathrm{EQR}(272,176,40)$	$\mathrm{EQR}(338,169,40)$	$\mathrm{EQR}(368,184,48)$	$\mathrm{EQR}(432,216,48)$	$\mathrm{EQR}(440,220,48)$
	$\mathrm{BCH}(n,k,\delta)$	$\mathrm{BCH}(n,k,\delta)$	$\mathrm{BCH}(n,k,\delta)$	$\mathrm{BCH}(n,k,\delta)$	$\mathrm{BCH}(n,k,\delta)$
$\mathbb{F}_2$	$\mathrm{BCH}(511,76,171)$	$\mathrm{BCH}(511,103,123)$	$\mathrm{BCH}(511,121,117)$	$\mathrm{BCH}(511,166,95)$	$\mathrm{BCH}(511,184,91)$

TABLE 1. Description of the codes selected for the experimentation

Finally, 100 experiments with different random seed were performed to obtain a set of results that can be analyzed statistically. We used a desktop equipped with CPU Intel Core I7, 8GB RAM, and Ubuntu 18.04 O.S. as experiment host.

5.2. **Analysis of results over $\mathbb{F}_8$.** The experimentation was performed with two different algorithms: GGA and CHC. Both were implemented using discrete and order representation, as described in Section 4, in separate experiments, so that there are four different configurations to be tested: GGA-Discrete, GGA-Order, CHC-Discrete, and CHC-Order. The reason of including the CHC algorithms in the experimentation is their ability to overcome local optima due to the their component design to establish a balance between diversity and convergence. This strategy is beneficial to obtain better solutions by avoiding premature convergence of traditional GGAs.

The algorithms' parameters were set after a trial-and-error procedure. As the number of datasets is high, these parameters were not fitted specifically for each problem, but as those that can provide good results in average. Additional experiments were conducted individually for the algorithms with the worst performance to find better parameter settings, with no success.

The parameters used in the experimentation are shown in Table 2. The row *Mutation probability by gene* is describes the probability of mutation of each coordinate in a solution. The row *Evaluations* stands for the main stopping criterion of each algorithm (i.e. to reach a maximum number of evaluations). The row *Reinitialization* shows the number of solutions evaluated with no improvements required to reinitialize the population in GGA proposals. A secondary stopping criterion was set for all algorithms: if the true known distance of the code being evaluated (or a lower bound) is found, the algorithm stops even if the main stopping criterion is not satisfied yet.

Parameter	**GGA-Discrete**	**CHC-Discrete**	**GGA-Order**	**CHC-Order**
Population size	400	400	400	400
Crossover	Uniform	Uniform	AX_2	AX_2
Mutation	Random	–	2-swap	–
Crossover probability	0.7	–	0.8	–
Mutation probability by gene	0.01	–	–	–
Evaluations	500000	500000	500000	500000
Reinitialization	100000	–	100000	–

TABLE 2. Algorithms' parameters for experimentation with codes over $\mathbb{F}_8$

Tables 3 and 4 show the results obtained for each dataset containing a code over $\mathbb{F}_8$. The columns **Dataset** display the parameters of the corresponding best known linear codes according to [34]. Columns **Best** describe the minimum weight found by each algorithm, and the subindices indicate the number of experiments that provided such result. Columns **Worst** introduce the worst minimum weight obtained. Columns **Mean** show the average of the minimum weights obtained in the 100 experiments. Finally, Columns **Time/Eval.** print the average time in seconds of each experiment, together with the average number of evaluations required to finish it.

Statistical tests were applied over the mean of the upper bounds obtained in the experiments for each data set (columns **Mean**), to know if there are significant differences between the average behavior of the algorithms. For each dataset, all algorithms were sorted by its mean in ascending order, and the non-parametric Wilcoxon test with 95% of confidence level was applied to compare them in the aforementioned sequence. According to the results, we ranked the algorithms from 1 to 4, where 1 is the best algorithm, and 4 the worst one. The rank order of each algorithm is depicted in columns **Mean** as subindices. We remark that two algorithms with the same rank says that there is no statistical difference between their performance.

According to the statistical tests, Table 4 shows that CHC-Order and GGA-Order return the best results, and there are no statistical difference between both algorithms in terms of performance. In fact, both are able to output the true distance in all datasets except for the code $(90,50,21-35)$, where a new upper bound, 22,

	CHC (discrete)				GGA (discrete)			
Dataset	**Best**	**Worst**	**Mean**	**Time / Eval.**	**Best**	**Worst**	**Mean**	**Time / Eval.**
$(30, 16, 10-13)$	$10_{(100)}$	10	$10_{(1)}$	$0.136_{(4)}/7117.3$	$10_{(100)}$	10	$10_{(1)}$	$0.015_{(3)}/6074.6$
$(30, 18, 9-11)$	$9_{(100)}$	9	$9_{(1)}$	$0.152_{(4)}/7745.6$	$9_{(100)}$	9	$9_{(1)}$	$0.02_{(3)}/6742.8$
$(45, 22, 15-21)$	$15_{(90)}$	16	$15.1_{(2)}$	$1.865_{(4)}/148352.3$	$15_{(100)}$	15	$15_{(1)}$	$0.304_{(3)}/85878.5$
$(45, 24, 14-19)$	$14_{(100)}$	14	$14_{(1)}$	$0.245_{(4)}/15373.6$	$14_{(100)}$	14	$14_{(1)}$	$0.032_{(3)}/10307$
$(45, 26, 12-17)$	$12_{(100)}$	12	$12_{(1)}$	$0.717_{(4)}/43823.4$	$12_{(100)}$	12	$12_{(1)}$	$0.193_{(3)}/49539.9$
$(45, 28, 11-15)$	$11_{(100)}$	11	$11_{(1)}$	$0.309_{(4)}/13732$	$11_{(100)}$	11	$11_{(1)}$	$0.039_{(3)}/10605.2$
$(60, 28, 21-28)$	$21_{(80)}$	22	$21.2_{(2)}$	$2_{(4)}/164512.4$	$21_{(81)}$	22	$21.2_{(2)}$	$0.867_{(3)}/196025.9$
$(60, 30, 20-27)$	$20_{(100)}$	20	$20_{(1)}$	$0.4_{(4)}/21209.3$	$20_{(100)}$	20	$20_{(1)}$	$0.062_{(3)}/13229.9$
$(60, 32, 19-25)$	$19_{(100)}$	19	$19_{(1)}$	$0.26_{(4)}/19573.3$	$19_{(100)}$	19	$19_{(1)}$	$0.053_{(3)}/12651.8$
$(60, 34, 17-23)$	$17_{(98)}$	18	$17_{(1)}$	$1.929_{(4)}/122798.7$	$17_{(100)}$	17	$17_{(1)}$	$0.421_{(3)}/78914.9$
$(75, 30, 28-40)$	$28_{(2)}$	32	$30.1_{(2)}$	$3.995_{(4)}/309250.5$	$29_{(3)}$	32	$30.4_{(3)}$	$2.754_{(3)}/500000$
$(75, 35, 24-35)$	$24_{(1)}$	28	$27.6_{(2)}$	$3.685_{(3)}/228937$	$26_{(2)}$	28	$27.6_{(2)}$	$3.369_{(2)}/500000$
$(75, 40, 20-31)$	$20_{(7)}$	24	$22.4_{(2)}$	$3.968_{(4)}/262879.6$	$20_{(2)}$	25	$22.8_{(3)}$	$1.005_{(3)}/144057$
$(75, 45, 17-27)$	$18_{(97)}$	20	$18.1_{(2)}$	$10.35_{(2)}/500000$	$18_{(88)}$	20	$18.2_{(3)}$	$4.208_{(1)}/500000$
$(90, 19, 49-63)$	$49_{(6)}$	54	$51.9_{(2)}$	$2.151_{(4)}/260640.5$	$49_{(5)}$	52	$51.8_{(2)}$	$1.56_{(3)}/285425.6$
$(90, 50, 21-35)$	$27_{(17)}$	29	$27.8_{(2)}$	$9.174_{(2)}/500000$	$25_{(1)}$	29	$27.8_{(2)}$	$4.77_{(1)}/500000$
$(90, 60, 16-26)$	$18_{(1)}$	21	$20.2_{(2)}$	$12.475_{(2)}/500000$	$19_{(29)}$	21	$20.1_{(2)}$	$5.71_{(1)}/500000$
$(90, 70, 11-17)$	$12_{(2)}$	14	$13_{(2)}$	$17.405_{(4)}/500000$	$11_{(1)}$	14	$13_{(2)}$	$1.554_{(3)}/120408$
$(130, 85, 23-40)$	$29_{(1)}$	32	$31.3_{(3)}$	$20.469_{(2)}/500000$	$29_{(2)}$	32	$31_{(2)}$	$10.538_{(1)}/500000$
$(130, 95, 18-30)$	$22_{(8)}$	24	$23.4_{(3)}$	$23.758_{(2)}/500000$	$22_{(21)}$	24	$22.9_{(2)}$	$11.745_{(1)}/500000$

TABLE 3. Results of CHC-Discrete and GGA-Discrete over $\mathbb{F}_8$ codes

	CHC (order)				GGA (order)			
Dataset	**Best**	**Worst**	**Mean**	**Time / Eval.**	**Best**	**Worst**	**Mean**	**Time / Eval.**
$(30, 16, 10-13)$	$10_{(100)}$	10	$10_{(1)}$	$0.004_{(2)}/1.6$	$10_{(100)}$	10	$10_{(1)}$	$0.003_{(1)}/1.6$
$(30, 18, 9-11)$	$9_{(100)}$	9	$9_{(1)}$	$0.005_{(2)}/2$	$9_{(100)}$	9	$9_{(1)}$	$0.004_{(1)}/2$
$(45, 22, 15-21)$	$15_{(100)}$	15	$15_{(1)}$	$0.01_{(2)}/57.8$	$15_{(100)}$	15	$15_{(1)}$	$0.009_{(1)}/57.8$
$(45, 24, 14-19)$	$14_{(100)}$	14	$14_{(1)}$	$0.011_{(2)}/13.4$	$14_{(100)}$	14	$14_{(1)}$	$0.01_{(1)}/13.4$
$(45, 26, 12-17)$	$12_{(100)}$	12	$12_{(1)}$	$0.011_{(2)}/9.5$	$12_{(100)}$	12	$12_{(1)}$	$0.01_{(1)}/9.5$
$(45, 28, 11-15)$	$11_{(100)}$	11	$11_{(1)}$	$0.011_{(2)}/1.9$	$11_{(100)}$	11	$11_{(1)}$	$0.01_{(1)}/1.9$
$(60, 28, 21-28)$	$21_{(100)}$	21	$21_{(1)}$	$0.019_{(2)}/42.8$	$21_{(100)}$	21	$21_{(1)}$	$0.017_{(1)}/42.8$
$(60, 30, 20-27)$	$20_{(100)}$	20	$20_{(1)}$	$0.019_{(2)}/5$	$20_{(100)}$	20	$20_{(1)}$	$0.018_{(1)}/5$
$(60, 32, 19-25)$	$19_{(100)}$	19	$19_{(1)}$	$0.021_{(2)}/8.3$	$19_{(100)}$	19	$19_{(1)}$	$0.02_{(1)}/8.3$
$(60, 34, 17-23)$	$17_{(100)}$	17	$17_{(1)}$	$0.021_{(2)}/18.4$	$17_{(100)}$	17	$17_{(1)}$	$0.021_{(1)}/18.4$
$(75, 30, 28-40)$	$28_{(100)}$	28	$28_{(1)}$	$0.029_{(2)}/19.1$	$28_{(100)}$	28	$28_{(1)}$	$0.028_{(1)}/19.1$
$(75, 35, 24-35)$	$24_{(100)}$	24	$24_{(1)}$	$0.12_{(1)}/1269.2$	$24_{(100)}$	24	$24_{(1)}$	$0.134_{(1)}/1716.2$
$(75, 40, 20-31)$	$20_{(100)}$	20	$20_{(1)}$	$0.037_{(2)}/5.9$	$20_{(100)}$	20	$20_{(1)}$	$0.036_{(1)}/5.9$
$\mathbf{(75, 45, 17-27)}$	$\mathbf{15}_{(100)}$	**15**	$\mathbf{15}_{(1)}$	$11.695_{(2)}/72438.6$	$\mathbf{15}_{(100)}$	**15**	$\mathbf{15}_{(1)}$	$12.713_{(2)}/77303.3$
$(90, 19, 49-63)$	$49_{(100)}$	49	$49_{(1)}$	$0.179_{(1)}/3279.7$	$49_{(100)}$	49	$49_{(1)}$	$0.252_{(2)}/5249.6$
$(90, 50, 21-35)$	$22_{(34)}$	24	$22.7_{(1)}$	$41.898_{(4)}/267888.3$	$22_{(28)}$	24	$22.8_{(1)}$	$38.437_{(3)}/239954.9$
$(90, 60, 16-26)$	$16_{(100)}$	16	$16_{(1)}$	$13.265_{(3)}/78140$	$16_{(99)}$	17	$16_{(1)}$	$15.42_{(3)}/86775.2$
$(90, 70, 11-17)$	$11_{(100)}$	11	$11_{(1)}$	$0.066_{(1)}/48.9$	$11_{(100)}$	11	$11_{(1)}$	$0.066_{(2)}/48.9$
$(130, 85, 23-40)$	$23_{(1)}$	26	$24.9_{(1)}$	$63.325_{(3)}/135700$	$23_{(2)}$	26	$25_{(1)}$	$145.695_{(4)}/293522$
$(130, 95, 18-30)$	$18_{(88)}$	19	$18.1_{(1)}$	$84.811_{(3)}/185147.7$	$18_{(80)}$	19	$18.2_{(1)}$	$87.796_{(3)}/177852.8$

TABLE 4. Results of CHC-Order and GGA-Order over $\mathbb{F}_8$ codes

is stablished. In particular, the optimum was reached in all experiments for 17 datasets (CHC-order) and 16 datasets (GGA-order). The worst behavior was obtained by both algorithms for the code $(130, 85, 23-40)$, where the optimum was reached in 1 and 2 experiments for CHC-order and GGA-order, respectively. Something similar happens with the code $(130, 95, 18-30)$. In this case, the true minimum distance was found in 88 experiments by CHC-order, and in 80 experiments by GGA. Both algorithms return a distance of 19 in the worst case.

We highlight the results for the code $(70, 45, 17-27)$ for which we found out an upper bound of the distance that equals 15. According to [34], the code is constructed using the $(77, 46, 18)$ linear code proposed in [35]. As our result falls below the claimed lower bound, we made additional analyses and compute a codeword whose weight equals 15 both in $(70, 45, 17-27)$ and $(77, 46, 18)$ codes. Both generating matrices are provided by the webpage [34] and MAGMA [33]. Regarding the discovered inaccuracy, Section 6 shows our findings and describes explicitly some codewords of weight 15.

In contrast, table 3 shows that CHC-Discrete and GGA-Discrete provided the worst performance, and there are significant differences being compared with the algorithms that use the order representation. We may see that discrete representation algorithms are able to find the optimum code distance for small codes, up to length

$n = 75$. For larger codes, they get trapped into local optima. In fact, discrete representation-based approaches suffer from high selective pressure and get trapped into local optima near to the $(0, \ldots, 0) \in \mathbb{F}_q^k$ trivial solution, which is not a valid solution, and this makes difficult to explore the solution space suitably. To give support to this statement, Figure 2 plots the evolution of the diversity during the execution of the algorithms GGA-Discrete and GGA-Order (right-hand figure), and GGA-Discrete and CHC-Discrete (left-hand figure), for the best solutions found for the code $(90, 50, 21)$. The diversity is shown in log scale. If we compare GGA-Discrete and GGA-Order, we may see how diversity in GGA-Discrete drops fast to near zero, and only returns to high values when the population is reinitialized. However, GGA-Order remains constant through all the iterations. On the other hand, if we compare CHC-Discrete *vs* GGA-Discrete, they show a similar behavior. The main difference between the two graphics displayed in Figure 2 is that the number of iterations in CHC-Discrete is higher, and the diversity drop is softer. This behavior is explained by the components of CHC to find a balance between diversity and convergence. However, according to Table 3, it is not enough to overcome the high selective pressure of the trivial solution, and no statistical difference can be found regarding performance.

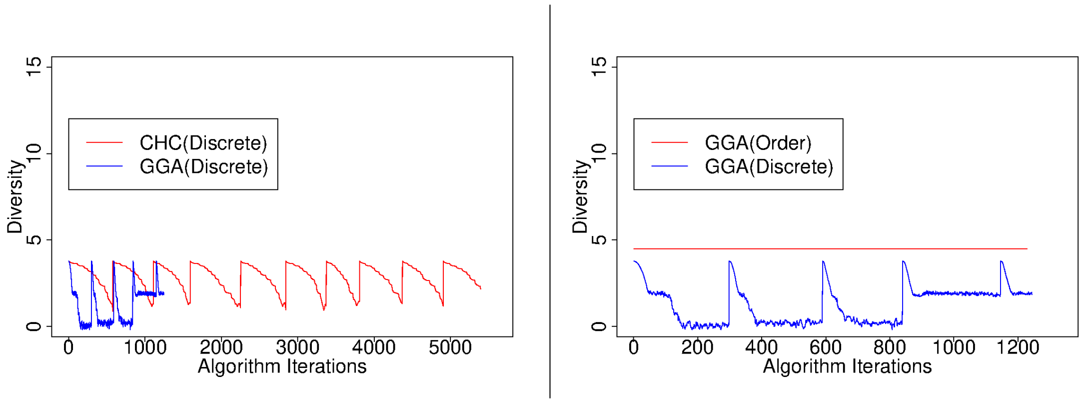

FIGURE 2. Evolution of the population diversity: GGA-Discrete vs CHC-Discrete (left), and GGA-Order vs GGA-Discrete (right)

To finish the analysis of the results, we are also interested in the comparison of the time complexities. The four algorithms were sorted by their average execution time in ascending order, and the Wilcoxon test with 95% of confidence level was applied to rank them all. The subindices in columns **Time/Eval.** of Tables 3 and 4 show the rank of each algorithm, where a lower value means that the algorithm is faster. The main bottleneck between discrete and order approaches is the fitness evaluation function, which are in $\mathcal{O}(kn)$ and $\mathcal{O}(k^2n)$ respectively. Under this consideration, it could be expected that GGA-Discrete and CHC-Discrete were faster than GGA-Order and CHC-Order. This occurs only for larger codes, where the number of evaluations is high. In the case of small codes, discrete representation approaches are worse than the order representation ones in terms of execution time. In fact, if we focus on GGA-Order and CHC-Order for small codes, the average number of evaluations required to obtain the distance is even lower than the size of the population, meaning that those optimal solutions could be easily found by a random search. This fact provides an additional interest in the proposed method in this manuscript, since it shows empirically that using the RREF simplifies the search of the minimum distance at an experimental level.

To conclude this section, we remark the following outcomes: (a) The proposed order representation scheme, in conjunction with the strategy of computing the RREF of the generating matrix, help to overcome the limitations of the classic discrete representation in terms of falling into local optima; (b) despite the higher time complexity to compute the RREF, this strategy changes the search space which eases finding optimal solutions, even with a random search for small problems, as it is suggested by analyzing the number of evaluations needed to reach the optimum, printed in Tables 3 and 4; and (c) the experimental results suggest that designing metaheuristics with permutation representation can be an efficient and competitive tool to calculate minimum distance. In the codes studied, for which the minimum distance was unknown, our approach was able to find the true code distance in 18 of the 20 datasets, and a new upper bound was found for the remaining. It was also able to detect an inaccuracy in the code $(75, 45, 17-27)$, establishing that 15 is a new upper bound of its distance, see Section 6.

5.3. **Analysis of results over $\mathbb{F}_2$.** In this section we analyze the results for BCH and EQR codes over $\mathbb{F}_2$. We also set the parameters from Table 2, except the maximum number of evaluations (stopping criterion) and the size of the population, fixed after a trial-and-error procedure. The number of evaluations to stop the algorithms was set to 1000000 in the case of BCH codes, and 500000 for EQR codes. The size of the population was set to 200 for BCH$(511, 103, 123)$, BCH$(511, 166, 95)$ and BCH$(511, 184, 91)$, since in a preliminary experimentation we observed a high space exploration but low exploitation. The size of the population was 400 for the other codes. Finally, other changes regarding crossover probability, crossover method, reinitialization, etc., did not provide improvements with respect to accuracy of results, and they were set as in Table 2.

We remark that the designed distance of a BCH codes is a lower bound of the minimum distance, although it is not ensured that they coincide. This is the case of BCH$(511, 103, 123)$, with designed distance 123 and minimum distance 127, and BCH$(511, 121, 117)$ (designed distance 117, minimum distance 119). The minimum distances for these codes are computed in [36, Table I].

	CHC (order)				GGA (order)			
Dataset	**Best**	**Worst**	**Mean**	**Time / Eval.**	**Best**	**Worst**	**Mean**	**Time / Eval.**
BCH$(511, 103, 123)$	$127_{(2)}$	144	$132.5_{(1)}$	$1340.28_{(1)}/1000000$	$127_{(4)}$	144	$133.6_{(1)}$	$1379.63_{(2)}/1000000$
BCH$(511, 121, 117)$	$120_{(3)}$	139	$133.3_{(1)}$	$1725.28_{(2)}/1000000$	$120_{(1)}$	139	$134.1_{(1)}$	$1712.15_{(1)}/1000000$
BCH$(511, 166, 95)$	$95_{(1)}$	119	$114.3_{(1)}$	$2912.57_{(2)}/519668$	$95_{(1)}$	119	$114.4_{(1)}$	$2780.14_{(1)}/196196$
BCH$(511, 76, 171)$	$171_{(100)}$	171	$171_{(1)}$	$0.627_{(1)}/4424.6$	$171_{(100)}$	171	$171_{(1)}$	$0.55_{(1)}/254$
BCH$(511, 58, 183)$	$183_{(100)}$	183	$183_{(1)}$	$0.245_{(2)}/805$	$183_{(100)}$	183	$183_{(1)}$	$0.21_{(1)}/1175.5$
EQR$(272, 136, 40)$	$40_{(100)}$	40	$40_{(1)}$	$0.35_{(2)}/3604.8$	$40_{(100)}$	40	$40_{(1)}$	$0.327_{(1)}/4784.8$
EQR$(338, 169, 40)$	$40_{(94)}$	50	$40.3_{(1)}$	$5.058_{(1)}/156803.8$	$40_{(88)}$	50	$40.6_{(1)}$	$11.551_{(2)}/170266.8$
EQR$(368, 184, 48)$	$48_{(13)}$	56	$54.2_{(1)}$	$317.353_{(1)}/345111.2$	$48_{(10)}$	60	$54.7_{(1)}$	$801.654_{(2)}/246819$
EQR$(432, 216, 48)$	$48_{(11)}$	68	$62_{(1)}$	$561.347_{(1)}/197260.3$	$48_{(5)}$	68	$63.6_{(2)}$	$1453.67_{(2)}/180514.4$
EQR$(440, 220, 48)$	$48_{(17)}$	72	$65.8_{(2)}$	$526.696_{(2)}/217904.2$	$48_{(82)}$	68	$50.9_{(1)}$	$176.835_{(1)}/167620.9$

TABLE 5. Results of CHC-Order and GGA-Order over $\mathbb{F}_2$ codes

Table 5 shows the results obtained for BCH and EQR codes using the algorithms CHC-Order and GGA-Order. Both algorithms reach the true minimum distance in all cases, except for BCH$(511, 121, 117)$ whose distance is 119 according to [36, Table I]. In spite of this, they were able to find an upper bound of distance with the same error correcting capability.

If we focus on the number of experiments that reached the optimal code distance in the datasets, we may observe the same behavior for both algorithms: As the dimension k increases, the probability of success for an experiment to reach the optimum decreases. Thus, for the codes BCH$(511, 58, 183)$ and BCH$(511, 76, 171)$, and EQR$(272, 136, 40)$, both algorithms obtain the optimum in all experiments, whilst for the codes BCH$(511, 166, 95)$, EQR$(432, 216, 48)$ and EQR$(440, 220, 48)$, the number of experiments that provide the optimum decreases substantially, specially for CHC-Order for the latter code. Nevertheless, the proposal still seems to be competitive for length 511.

On the other hand, if we analyze the worst/average solution with regard to the codes over $\mathbb{F}_8$ in the previous section, we may see that the differences with true distances are greater for the codes studied in this section. This is also expected, since the problem size increases, and the algorithms get trapped into local optima more easily. This also affects the computational time of each experiment, which requires a few minutes to end with the larger codes.

Regarding performance, we analyzed the results of the average minimum distance of the outputs using the Wilcoxon test with 95% of confidence level. We follow the same methodology that is described in the previous section. In the case of $\mathbb{F}_2$-codes, no significant statistical difference was noticed except for EQR$(440, 220, 48)$, where GGA-Order is clearly better than CHC-Order. No statistical difference was also found in the computational time for each dataset, except for EQR$(432, 216, 48)$, where CHC-Order is better. These results are consistent with the analysis carried out over $\mathbb{F}_8$-codes in the previous subsection.

We also compare the results of our approach with previous existing methods in the literature. More specifically, we compare our method over BCH codes with the proposal of Askali et al. in [12], where two variants of genetic algorithms are designed with discrete representation. This work was selected for comparison for the following reasons: (a) there are not much methods in the literature that attempt to calculate the distance of general linear codes, and this is the most recent (2013), as far as we know; (b) it also deals with genetic algorithms, so it is close to our approach; and (c) it calculates the distance for BCH codes of medium size, as we do. The second baseline is the work of Leon [8], where a probabilistic method is designed to estimate bounds of some EQR codes. This baseline helps to ensure that our method was able to obtain the true distance for these EQR codes.

Other approaches for specific codes exist in the literature, as for instance for BCH codes [36], but they impose hard constraints to exploit the code geometry, and therefore they are not comparable with our work, which is a general proposal for any linear code.

Table 6 summarizes the comparison between the results of the CHC-Order and the baseline methods in [12, 8]. In the case of [8], the probability of being the minimum distance is also provided. As we may observe, the proposal described in this manuscript outperforms clearly the baseline methods for BCH codes, and confirms the distance calculated with a given probability in [8]. In the case of [12], the same conclusion that we pointed out in Section 5.2 for discrete representation applies: As the baseline method uses discrete representation, a high selective pressure over the trivial solution $(0, \dots, 0) \in \mathbb{F}_q^k$ makes the algorithms get trapped into local optima. The order representation does not suffer from premature convergence, and the search is speeded up by the computation of the RREF, so it improves the results.

Dataset	CHC-Order	Baselines ([12, 8])
BCH(511, 103, 123)	127	160 [12]
BCH(511, 121, 117)	120	155 [12]
BCH(511, 166, 95)	95	135 [12]
BCH(511, 76, 171)	171	176 [12]
BCH(511, 58, 183)	183	183 [12]
EQR(272, 136, 40)	40	40 ($p = 10^{-100}$)[8]
EQR(338, 169, 40)	40	40 ($p = 10^{-100}$)[8]
EQR(368, 184, 48)	48	48 ($p = 10^{-20}$)[8]
EQR(432, 216, 48)	48	48 ($p = 10^{-10}$)[8]
EQR(440, 220, 48)	48	48 ($p = 10^{-10}$)[8]

Table 6. Distances found by the proposal CHC-order and the baseline methods.

5.4. **Discussion.** Finding the minimum distance of a linear code is an NP-hard problem. Actually, the associated decision problem is NP-complete for binary codes, see [3]. In this setting, our experiments have shown that genetic algorithms are good candidates to handle this problem. The results suggest that the discrete representation, in spite of being efficient and adequate to include the classic standard techniques, specially for binary codes, suffers of premature convergence. Such an issue seems to be due to the high selective pressure over the zero tuple, which is not a valid solution. Nevertheless, the weight of a message is not necessary correlated with the weight of its encoded word, so that local optima near the zero message could be far from an optimal codeword. Our proposal helps to overcome these drawbacks. This order representation is able to provide good results in conjunction with the strategy of computing the RREF of a generating matrix. Although computing the RREF, as fitness evaluation function, increases the theoretical time complexity from $\mathcal{O}(kn)$ to $\mathcal{O}(k^2n)$, we have shown that this disadvantage is palliated by the speed of the convergence to the optimal solution, which requires much less evaluations. The experiments over the field $\mathbb{F}_8$ reveal significant differences between the standard use of the discrete representation and the algorithms with order representation that we present in this paper. Actually, it is foreseeable achieving bigger differences when working over larger fields. The experiments over $\mathbb{F}_2$ also show that GGA-Order and CHC-Order outperform the best solutions found in [12], and they are be able to find the approximations in [8].

Finally, the experiments carried out in Sections 5.2 and 5.3 show that there is no significant difference between GGA-Order and CHC-Order regarding accuracy and computational time for the codes of largest length. Actually, they were able to find the true distance for all cases but for the code BCH(511, 121, 117), for which we found an upper bound of 120 whilst the true distance [36, Table I] is 119. In this case, we could assess the benefits of both algorithms qualitatively: While the designed GGA-Order requires to set a mutation operator and a population reinitialization criterion, CHC-Order does not use mutation, and it includes internal mechanisms to maintain diversity in the population. Thus, from the point of view of technical experimentation, CHC-Order has the advantage that it requires less parameter settings than GGA-Order and it could be preferable. However, the distance between solutions has to be computed, which makes the algorithm design more complex.

6. Analysis of the code $(75, 45, 17-27)$ over $\mathbb{F}_8$

According to [34] and Magma [33], the best known linear code over $\mathbb{F}_8$ with length 75 and dimension 45 is $(75, 45, 17-27)$ as it can be verified executing the first 7 lines of the Magma source code in Figure 3. However, our algorithms obtain the column permutation in (3), which provides a matrix whose RREF contains a row whose weight is 15. A codeword with this weight is exhibited in (4). Magma source code to verify our findings is set in lines 9 to 13 in Figure 3, which can be tested in the online version of Magma at the URL `http://magma.maths.usyd.edu.au/calc/`. Thus, the minimum distance is less or equal to 15, which is below of the lower bound provided by Magma.

$$(53, 11, 58, 36, 17, 27, 44, 8, 73, 24, 20, 71, 69, 46, 10, 43, 26, 61, 29, 57, 23, 6, 5, 67, 14, 4, 50, 45, 72, 59, 18, 25, 47, 28, 51, 68, 22, 48, 52, 42, 60, 21, 38, 64, 16, 34, 2, 3, 31, 33, 49, 63, 74, 54, 62, 19, 7, 1, 0, 15, 13, 66, 39, 65, 41, 30, 12, 37, 32, 70, 56, 40, 55, 9, 35) \tag{3}$$

$$(a^2+a, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, a, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, a, 0, 0, 0, 0, 0, 0, a^2+a, a^2, 0, a^2+a+1, 1, 0, a+1, 0, 0, a, 0, 0, 0, 0, 0, a^2+a+1, 0, 0, 0, a+1, 1, 0, 0, 0, 0, 0, a^2+a+1, 0, 0, 0, 0, 0, 0, 0, a+1, 0, 0, 0, 0) \tag{4}$$

According to [34], the linear code $(75, 45, 17-27)$ is constructed from the code with parameters $(77, 46, 18)$ described in [35]. By additional experiments we checked if the minimum distance of the code $(77, 46, 18)$ is 18.

FIGURE 3. MAGMA source code exhibiting a codeword with weight 15 for the code $(75, 45, 17-27)$

```
/* Build the code */
F<a> := GF(8);
V := VectorSpace(F,75);
C := BKLC(F,75,45);
C:Minimal;
/* Show the lower and upper bounds for the code */
BKLCLowerBound(F,75,45), BKLCUpperBound(F,75,45);
/* Set the word of weight 15 */
c := V ! [a^2 + a, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, a, 0, 0, 1, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, a, 0, 0, 0, 0, 0, 0, a^2 + a, a^2, 0, a^2 + a + 1, 1, 0, a + 1,
0, 0, a, 0, 0, 0, 0, 0, a^2 + a + 1, 0, 0, 0, a + 1, 1, 0, 0, 0, 0, 0, a^2 + a + 1,
0, 0, 0, 0, 0, 0, a + 1, 0, 0, 0, 0];
/* Checks if the word c belongs to code C */
c in C;
/* Prints the weight of the codeword c in code C */
Weight(c);
```

FIGURE 4. MAGMA source code exhibiting a codeword with weight 15 for the code $(77, 46, 18)$

```
/* Build the code */
F<a> := GF(8);
V := VectorSpace(F,77);
C := BKLC(F,77,46);
C:Minimal;
/* Show the lower and upper bounds for the code */
BKLCLowerBound(F,77,46), BKLCUpperBound(F,77,46);
/* Set the word of weight 15 */
c := V ! [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, a^2 + 1, 0,
a + 1, 0, 0, 0, 0, 1, a + 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, a + 1, 1, 0, 0,
a + 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, a + 1, 0, 0, 0, 0, 0,
a, 0, 0, 0, a^2 + 1, 0, 0];
/* Checks if the word c belongs to code C */
c in C;
/* Prints the weight of the codeword c in code C */
Weight(c);
```

We found out that the column permutation in (5) leaded to find the codeword shown in (6), as a row of the RREF of the modified matrix. The weight of this word is also 15, which is a new upper bound for the code. Figure 4 prints the MAGMA code that ensured these findings.

$$(62, 4, 38, 66, 31, 76, 36, 56, 35, 15, 57, 23, 39, 18, 68, 12, 40, 2, 61, 32, 8, 7, 21, 55, 34, 65, 24, 13, 25, 29, 11, 46, 10, 50, 9, 16, 51, 19, 72, 75, 71, 59, 43, 44, 17, 0, 26, 1, 22, 74, 73, 45, 3, 28, 67, 63, 49, 30, 33, 70, 54, 42, 60, 64, 53, 69, 6, 14, 52, 5, 20, 48, 37, 41, 47, 58, 27) \tag{5}$$

$$(1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, a^2+1, 0, a+1, 0, 0, 0, 0, 1, a+1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, a+1, 1, 0, 0, a+1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, a+1, 0, 0, 0, 0, 0, a, 0, 0, 0, a^2+1, 0, 0) \tag{6}$$

Additional tests were performed for other codes considered in [35]. More specifically, we calculated the permutations in (7) and (9) for the codes $(75, 44, 18)$ and $(76, 45, 18)$, respectively, which helps us finding the corresponding codewords in (8) and (10), both with weight 15. The MAGMA source codes which check that an upper bound for these codes is 15 is shown in Figures 5 and 6, respectively.

Figure 5. Magma source code exhibiting a codeword with weight 15 for the code $(75, 44, 18)$

```
/* Build the code */
F<a> := GF(8);
V := VectorSpace(F,75);
C := BKLC(F,75,44);
C:Minimal;
/* Show the lower and upper bounds for the code */
BKLCLowerBound(F,75,44), BKLCUpperBound(F,75,44);
/* Set the word of weight 15 */
c := V ! [a^2 + a, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, a + 1,
0, 1, 0, 0, 0, 0, a^2 + a, 1, 0, 0, 0, 0, 0, 0, 0, 0, a^2 + a, 0, 0, 0, 1, a^2 + a,
0, 0, 1, 0, a^2 + a, a^2 + a, 0, 0, 0, 0, 0, a^2 + a, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
0, 0, 0, 0, 0, a^2 + a + 1, 0, 0, 0, a + 1];
/* Checks if the word c belongs to code C */
c in C;
/* Prints the weight of the codeword c in code C */
Weight(c);
```

$$(33, 23, 13, 29, 53, 30, 71, 7, 12, 35, 34, 57, 60, 39, 52, 5, 24, 4, 56, 44, 31, 25, 43, 40, 11, 6, 22, 32, 26, 63, 62, 17, 18, 38, 9, 72, 2, 8, 19, 69, 36, 73, 21, 65, 42, 68, 14, 10, 20, 59, 58, 27, 28, 51, 66, 47, 48, 37, 67, 54, 70, 74, 16, 46, 3, 1, 15, 50, 55, 0, 41, 45, 61, 64, 49) \tag{7}$$

$$(a^2+a, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, a+1, 0, 1, 0, 0, 0, 0, a^2+a, 1, 0, 0, 0, 0, 0, 0, 0, 0, a^2+a, 0, 0, 0, 1, a^2+a, 0, 0, 1, 0, a^2+a, a^2+a, 0, 0, 0, 0, 0, a^2+a, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, a^2+a+1, 0, 0, 0, a+1) \tag{8}$$

$$(25, 56, 43, 14, 58, 11, 61, 57, 29, 10, 26, 51, 31, 8, 32, 73, 7, 53, 46, 60, 4, 12, 35, 67, 59, 2, 55, 36, 17, 6, 34, 24, 16, 72, 52, 44, 21, 9, 71, 19, 65, 62, 68, 45, 49, 75, 15, 42, 0, 48, 64, 13, 28, 40, 23, 27, 33, 3, 70, 37, 1, 50, 74, 54, 38, 41, 30, 63, 69, 5, 22, 18, 20, 66, 39, 47) \tag{9}$$

$$(a^2+a, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, a+1, 0, 1, 0, 0, 0, 0, a^2+a, 1, 0, 0, 0, 0, 0, 0, 0, 0, a^2+a, 0, 0, 0, 1, a^2+a, 0, 0, 1, 0, a^2+a, a^2+a, 0, 0, 0, 0, 0, a^2+a, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, a^2+a+1, 0, 0, 0, a+1, 0) \tag{10}$$

7. Conclusions

We have shown that, for a given generating matrix of a linear code, there exists a column permutation for which its reduced row echelon form has a row whose weight reaches the minimum distance. This opens the possibility to use a permutation representation in metaheuristics to find the minimum distance of an arbitrary linear code, as an alternative to the classic discrete representation. The proposed model is polynomial time-dependent with respect to the dimension of the code and the size of the base finite field. We have developed different schemes of evolutionary algorithms to prove the benefits of our approach experimentally. We have concluded that usual limitations of discrete representation, such as high selective pressure, are palliated by means of the new representation. Therefore, our approach is able to find true minimum distances of general linear codes of medium size, and it is the first work to address the problem for codes over finite fields larger than $\mathbb{F}_2$ using metaheuristics. Comparison with the existing methods in the literature suggests that our proposal is accurate and efficient for BCH and EQR codes over $\mathbb{F}_2$, and able to outperform previous approaches. In addition, we have been able to find some inaccuracies in the list of best known linear codes in [34] and [33].

The run time of the calculation of the reduced row echelon form increases polynomically with the code length and dimension. Future works will be conducted to reduce this execution time, and also to include information about specific code geometry and properties into the evolutionary process to speed up convergence to optimal solutions.

FIGURE 6. MAGMA source code exhibiting a codeword with weight 15 for the code $(76, 45, 18)$

```
/* Build the code */
F<a> := GF(8);
V := VectorSpace(F,76);
C := BKLC(F,76,45);
C:Minimal;
/* Show the lower and upper bounds for the code */
BKLCLowerBound(F,76,45), BKLCUpperBound(F,76,45);
/* Set the word of weight 15 */
c := V ! [a^2 + a, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, a + 1,
0, 1, 0, 0, 0, 0, a^2 + a, 1, 0, 0, 0, 0, 0, 0, 0, 0, a^2 + a, 0, 0, 0, 1, a^2 + a,
0, 0, 1, 0, a^2 + a, a^2 + a, 0, 0, 0, 0, 0, a^2 + a, 0, 0, 0, 0, 0, 0, 0, 0, 1,
0, 0, 0, 0, 0, a^2 + a + 1, 0, 0, 0, a + 1];
/* Checks if the word c belongs to code C */
c in C;
/* Prints the weight of the codeword c in code C */
Weight(c);
```

Acknowledgements

Research partially supported by grant PID2019-110525GB-I00 from Agencia Estatal de Investigación (AEI) and from Fondo Europeo de Desarrollo Regional (FEDER), and by grant A-FQM-470-UGR18 from Consejería de Economía y Conocimiento de la Junta de Andalucía and Programa Operativo FEDER 2014-2020.

References

[1] C. E. Shannon, A mathematical theory of communication, Bell System Technical Journal 27 (1948) 379–423 & 623–656.
[2] W. C. Huffmann, V. Pless, Fundamentals of Error-Correcting Codes, Cambridge University Press, 2003.
[3] A. Vardy, The intractability of computing the minimum distance of a code, IEEE Transactions on Information Theory 43 (6) (1997) 1757–1766. doi:10.1109/18.641542.
[4] E. R. Berlekamp, Algebraic Coding Theory, WORLD SCIENTIFIC, 2015. doi:10.1142/9407.
URL https://www.worldscientific.com/doi/abs/10.1142/9407
[5] S. B. Wicker, Reed-Solomon Codes and Their Applications, IEEE Press, 1994.
[6] A. Betten, M. Braun, H. Fripertinger, A. Kerber, A. Kohnert, A. Wassermann, Error-Correcting Linear Codes, Vol. 18 of Algorithms and Computation in Mathematics, Springer, 2006.
[7] P. Lisonek, L. Trummer, Algorithms for the minimum weight of linear codes, Adv. in Math. of Comm. 10 (2016) 195–207.
[8] J. S. Leon, A probabilistic algorithm for computing minimum weights of large error-correcting codes, IEEE Transactions on Information Theory 34 (5) (1988) 1354–1359. doi:10.1109/18.21270.
[9] K. Dontas, K. De Jong, Discovery of maximal distance codes using genetic algorithms, in: [1990] Proceedings of the 2nd International IEEE Conference on Tools for Artificial Intelligence, 1990, pp. 805–811. doi:10.1109/TAI.1990.130442.
[10] Z. MUXIANG, M. FULONG, Simulated annealing approach to the minimum distance of error-correcting codes, International Journal of Electronics - INT J ELECTRON 76 (1994) 377–384. doi:10.1080/00207219408925934.
[11] J. Bland, Local search optimisation applied to the minimum distance problem, Advanced Engineering Informatics 21 (4) (2007) 391 – 397. doi:https://doi.org/10.1016/j.aei.2007.01.002.
URL http://www.sciencedirect.com/science/article/pii/S1474034607000079
[12] M. Askali, A. Azouaoui, S. Nouh, M. Belkasmi, On the computing of the minimum distance of linear block codes by heuristic methods, International Journal of Communications, Network and System Sciences 05 (03 2013). doi:10.4236/ijcns.2012.511081.
[13] S. Nouh, M. Belkasmi, Genetic algorithms for finding the weight enumerator of binary linear block codes, International Journal of Applied Research on Information Technology and Computing 2 (03 2013). doi:10.5958/j.0975-8070.2.3.022.
[14] A. Barbieri, S. Cagnoni, G. Colavolpe, Genetic design of linear block error-correcting codes, in: B. Apolloni, M. Marinaro, R. Tagliaferri (Eds.), Biological and Artificial Intelligence Environments, Springer Netherlands, Dordrecht, 2005, pp. 107–116.
[15] V. Tomás, J. Rosenthal, R. Smarandache, Decoding of convolucional codes over the erasure channel., IEEE Transactions on Information Theory 58 (1) (2010) 90–108.
[16] R. Lidl, H. Niederreiter, Finite Fields, 2nd Edition, Encyclopedia of Mathematics and its Applications, Cambridge University Press, 1996. doi:10.1017/CBO9780511525926.
[17] J. Liesen, V. Mehrmann, Linear Algebra, Springer Undergraduate Mathematical Series, Springer International Publishing, 2015. doi:10.1007/978-3-319-24346-7.
[18] R. C. Bose, D. K. Ray-Chaudhuri, On a class of error correcting binary group codes, Information and Control 3 (1) (1960) 68 – 79. doi:10.1016/S0019-9958(60)90287-4.
[19] A. Hocquenghem, Codes correcteurs d'erreurs, Chiffres 2 (1959) 147–156.
[20] C. R. P. Hartmann, K. K. Tzeng, Generalizations of the BCH-bound., Information and Control 20 (1972) 489–498.
[21] J. Gómez-Torrecillas, F. J. Lobillo, G. Navarro, A. Neri, Hartmann–Tzeng bound and skew cyclic codes of designed Hamming distance., Finite Fields and Their Applications 50 (2018) 84–112. doi:10.1016/j.ffa.2017.11.001.
[22] J. Gómez-Torrecillas, F. J. Lobillo, G. Navarro, Minimum distance computation of linear codes via genetic algorithms with permutation encoding, ACM Communications in Computer Algebra 52 (3) (2019) 71–74. doi:10.1145/3313880.3313886.
URL http://dx.doi.org/10.1145/3313880.3313886

[23] J. R. Bunch, J. E. Hopcroft, Triangular factorization and inversion by fast matrix multiplication, Mathematics of Computation 28 (125) (1974) 231–236.

[24] D. Andrén, L. Hellström, K. Markström, On the complexity of matrix reduction over finite fields, Advances in Applied Mathematics 39 (2007) 428–452.

[25] T. Back, D. B. Fogel, Z. Michalewicz, Handbook of Evolutionary Computation, 1st Edition, IOP Publishing Ltd., GBR, 1997.

[26] T. Blickle, L. Thiele, A comparison of selection schemes used in evolutionary algorithms, Evolutionary Computation 4 (1997) 361–394.

[27] T. D. Gwiazda, Genetic Algorithms Reference, Tomasz Gwiazda, 2006.

[28] L. J. Eshelman, The CHC adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination, Vol. 1 of Foundations of Genetic Algorithms, Elsevier, 1991, pp. 265 – 283. doi:https://doi.org/10.1016/B978-0-08-050684-5.50020-3.
URL http://www.sciencedirect.com/science/article/pii/B9780080506845500203

[29] L. J. Eshelman, J. D. Schaffer, Real-coded genetic algorithms and interval-schemata, in: Foundations of Genetic Algorithms, Vol. 2 of Foundations of Genetic Algorithms, Elsevier, 1993, pp. 187 – 202.

[30] O. Cordón, S. Damas, J. Santamaría, Feature-based image registration by means of the chc evolutionary algorithm, Image and Vision Computing 24 (5) (2006) 525 – 533.

[31] A. Simões, E. Costa, Memory-based CHC algorithms for the dynamic traveling salesman problem, in: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, GECCO '11, Association for Computing Machinery, New York, NY, USA, 2011, pp. 1037–1044. doi:10.1145/2001576.2001717.
URL https://doi.org/10.1145/2001576.2001717

[32] J. D. Dixon, The probability of generating the symmetric group, Mathematische Zeitschrift 110 (1969) 199–205.

[33] W. Bosma, J. Cannon, C. Playoust, The Magma algebra system. I. The user language, J. Symbolic Comput. 24 (3-4) (1997) 235–265, computational algebra and number theory (London, 1993). doi:10.1006/jsco.1996.0125.
URL http://dx.doi.org/10.1006/jsco.1996.0125

[34] M. Grassl, Bounds on the minimum distance of linear codes and quantum codes, Online available at http://www.codetables.de, accessed 2020-01-22 (2007).

[35] C. Xing, S. Ling, A class of linear codes with good parameters from algebraic curves, IEEE Transactions on Information Theory 46 (4) (2000) 1527–1532.

[36] D. Augot, P. Charpin, N. Sendrier, Studying the locator polynomials of minimum weight codewords of bch codes, IEEE Transactions on Information Theory 38 (3) (1992) 960–973. doi:10.1109/18.135638.

E-mail address: manupc@decsai.ugr.es

Department of Computer Science and Artificial Intelligence and CITIC University of Granada, Spain

E-mail address: gomezj@ugr.es

Department of Algebra, University of Granada

E-mail address: jlobillo@ugr.es

Department of Algebra and CITIC, University of Granada

E-mail address: gnavarro@ugr.es

Department of Computer Science and Artificial Intelligence and CITIC University of Granada, Spain