

On the computation of the automorphisms group of low density parity check codes using genetic algorithm

Bellfkih El Mehdi¹, Said Nouh², Imrane Chemseddine Idrissi², Abdelaziz Ettaoufik², Khalid Louartiti¹, Jamal Mouline¹

¹L3A Lab, Faculty of Sciences Ben M'sik, Hassan II University of Casablanca, Casablanca, Morocco

²LTIM Lab, Faculty of Sciences Ben M'sik, Hassan II University of Casablanca, Casablanca, Morocco

Article Info

Article history:

Received Aug 3, 2021

Revised Nov 5, 2021

Accepted Dec 1, 2021

Keywords:

Automorphism group

Crossover

Genetic algorithm

Low density parity check

codes

Mutation

ABSTRACT

The genetic algorithm (GA) is an adaptive metaheuristic search method based on the process of evolution and natural selection theory. It is an efficient algorithm used for solving the combinatorial optimization problems, e.g., travel salesman problem (TSP), linear ordering problem (LOP), and job-shop scheduling problem (JSP). The simple GA applied takes a long time to reach the optimal solution, the configuration of the GA parameters is vital for a successful GA search and convergence to optimal solutions, it includes population size, crossover operator, and mutation operator rates. Also, very recently, many research papers involved the GA in coding theory, In particular, in the decoding linear block codes case, which has heavily contributed to reducing the complexity, and guaranting the convergence of searching in fewer iterations. In this paper, an efficient method based on the genetic algorithm is proposed, and it is used for computing the Automorphisms groups of low density parity check (LDPC) codes, the results of the aforementioned method show a significant efficiency in finding an important set of Automorphisms set of LDPC codes.

This is an open access article under the [CC BY-SA](https://creativecommons.org/licenses/by-sa/4.0/) license.



Corresponding Author:

Bellfkih El Mehdi

L3A Lab, Faculty of Sciences Ben M'sik, Hassan II University of Casablanca

Casablanca, Morocco

Email: elmehdi.bellfkih@gmail.com

1. INTRODUCTION AND PRILIMINARIES

There are varying methods in coding theory which addresses its application, one of them is through determining the Automorphisms groups of codes, they allow us to determine the structure of the codes, classifying them and help the decoding algorithm. This remains a challenge since determining the whole automorphisms groups of codes is difficult, except finite simple groups which have been realized using the sporadic groups [1] (e.g, the aumorphism group of golay codes are mathieu groups).

Recalling that the hamming distance between any two codewords (vectors) c, c' in \mathbb{F}_2^n is defined to be the number of coordinates in which c and c' differ. A binary linear $[n,k,d]$ -code C over \mathbb{F}_2 is a k -dementional subspace of the vector space \mathbb{F}_2^n , where:

$$d = d(C) = \min_{c \neq c' \in C} d(c, c') = \min_{c \in C \setminus \{0\}} wt(c) \quad (1)$$

and its generator matrix G is a $k \times n$ matrix whose rows is the basis of C .

Let C be a binary linear code and G its generator matrix, considering the action of the symmetric group \mathcal{S}_n on the G columns. For all σ in \mathcal{S}_n , denote by $G \cdot \sigma$ the matrix obtained from the permutation of the G

columns. Let $\sigma \in \mathcal{S}_n, c = (c_1, c_2, \dots, c_n) \in C$:

$$\sigma(c) = \sigma(c_1, c_2, \dots, c_n) = (c_{\sigma(1)}, c_{\sigma(2)}, \dots, c_{\sigma(n)}) \tag{2}$$

$$\sigma(C) = \{\sigma(c), c \in C\} \tag{3}$$

any permutation of the G columns which maps the rows of G into rows of the same matrix, is called an automorphism of C. The set of all automorphism permutations forms a subgroup of \mathcal{S}_n , denoted by $Aut(C)$:

$$Aut(C) = \{\sigma \in \mathcal{S}_n, \sigma(C) = C\} \tag{4}$$

let A be a group, A is an Automorphism group of C if $A \subseteq Aut(C)$ and A is the Automorphism group of C if $A = Aut(C)$ [2].

This paper, mainly focuses on the computation of the automorphisms groups of LDPC codes. The section 2, includes some definitions, details, also it presents related works using genetic algorithm (GA). In section 3, the GA-based method is described, including the fitness function, stochastic crossbreeding, and stochastic operators. The results are presented in section 4. Section 5 is devoted to the conclusion and perspectives.

2. RELATED WORKS

2.1. Low density parity check codes

Gallager devised the low density parity check (LDPC) codes, often known as Gallager codes, in 1962, they are class of linear block codes, defined by sparse parity check matrices, where each column contains a small fixed number w_c of 1s and each row contains a small fixed number $w_r > w_c$ of 1s [3]. Due to the limited characteristics of computers at that times, this class of linear code was absent till 1990s where they have been reinvented through the Macky and Neal works, its has been shown that LDPC codes performance is near to Shannon limit performance with belief propagation algorithm (BPA) [4]. There are characteristics that distinguish LDPC codes from Turbo codes, such as superior performance when the block length is large, enormous flexibility, easy description and subsequent theoretical venerability, decreased decoding complexity, and so on [5].

There is an algebraic representation, the LDPC code is denoted as (n, w_c, w_r) , where n is the binary linear code length, w_c is the number of 1s in the column of the sparse parity check matrix (i.e. the column weight), and w_r is the number of 1s in the row in of the sparse parity check matrix (i.e. the row weight) as illustrated in Figure 1, if w_c and w_r are invariant, it's called regular LDPC codes, else it's called irregular LDPC codes. Both of the two must satisfy this following condition:

$$cH^T = 0 \tag{5}$$

where c is a codeword and H is the sparse parity check matrix. There is another representation for LDPC codes which is trough Tanner graphs (graphical representation of the sparse parity check matrix), they contain two class of nodes, variables nodes, they represent the sparse parity check matrix columns, and check nodes, they represent the sparse parity check matrix rows. for each nonzero h_{ij} of H, an edge will be presented between check node i and variable node j as illustrated in Figure 2.

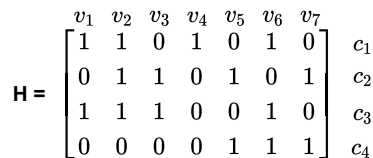


Figure 1. A sparse parity check matrix of some LDPC code

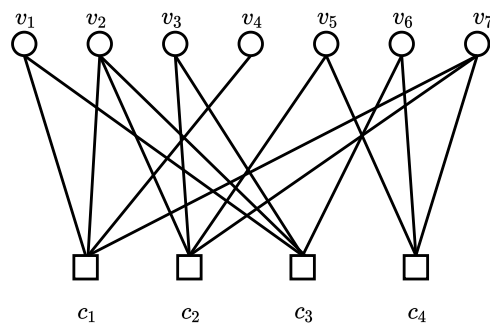


Figure 2. A tanner graph of the left LDPC code

2.2. Genetic algorithm

The GA is a type of evolutionary algorithm that belongs to the family of algorithms known as genetic algorithms. A GA's population evolves via genetic operators inspired by biology's evolutionary process [6], Darwin recognized that species evolution is driven by two processes: the process of selection and reproduction. The reproduction of the fittest and most vigorous individuals is provided by selection, while reproduction is a phase in which evolution takes place. Travel salesman problem (TSP), job-shop scheduling problem (JSP), bandwidth-reduction problem (BRP), and linear ordering problem (LOP) are examples of permutation problems. [7], [8] which is a class of combinatorial optimization problems, the task is to arrange some genes (objects) in chromosome, with no duplicates, in a certain order that optimizes an objective function, where the representation of the chromosomes depend on types of the optimization problems [9], [10].

GA addresses the permutation issue by searching fast via the search space. It employs the selection, crossover, and mutation operators to produce superior chromosomes at the lowest possible cost [11]. The efficiency of using evolutionary algorithms to solve combinatorial optimization problems has been demonstrated [12]-[16]. It exists powerful algorithms, a nature-inspired algorithms like gaining-sharing knowledge based algorithm (GSK) [17]-[19] which it has shown better results in solving optimization problems. The GA has several advantages such as:

- Uses only the objective function's evaluation, regardless of its nature (continuity, differentiability...), as a result of which there is more flexibility and a broader variety of applications.
- Instead of a single iteration as in standard algorithms, generation adopts a parallel form by operating on several points at once.
- Probabilistic transition rules (selection, crossover, and mutation probability) rather than deterministic ones.

Many research have indicated that exhibiting a comprehension of the GA parameters' interaction process, notably crossover probability, mutation probability, and population size, is the most important factor in evaluating the process. These factors are connected to each other in some way that impacts the GA efficiency. The optimal circumstance to use GA is when there is variety in the starting population with a high crossover chance and a low mutation probability [20].

It is important to note that the traditional crossover operator can not be applied to perform of permutation problems solution due to chromosomes arrangement of the genes is crucial, and no genes should be duplicated or missing [11]. Also, In comparison to other scenarios, it is more computationally expensive. The reason for this is that for offspring with duplicate numbers, a legalization step is necessary after each substring exchange. In such a case, the time required to complete a crossover operation increases fast as chromosome size increases, which can reduce the efficiency of permutation-based GAs [21]. Liu and Kroll in their research article [22] developed a genetic algorithm did not use the crossover operator. It is important to note again, that GA has been used to find Automorphisms set for some block codes like bose–chaudhuri–hocquenghem (BCH) and quadratic residue (QR) codes of small length [23], also to compute the minimum distance of linear block codes [24].

3. GENETIC ALGORITHM-BASED METHOD

In this section of the article, the genetic algorithm-based method is proposed, which uses an encoding that consists of treating an individual (permutation) as a sequence of numbers from 1 to the length of the code n . Also, these proposed method components work as explained in the next subsections. These components of the algorithm, which are the fitness function, which is used in the calculation of an individual's fitness value, those fitness values are crucial in the choosing and construction of the individuals of the next generation through operators. The search space consists of $n!$ individuals, each with n digits. The selection, crossover, and mutation operators will be explained and illustrated with figures. Then an overall organigram that shows how the algorithm works will be presented, identifying inputs and outputs.

3.1. The search space and fitness function

Let C be a binary linear code of length n , since our problem of finding the stabilizers set belongs to the optimization problems, the size of search space is linked to the code length n , This search space where the our proposed method will search, contains $n!$ permutations. For all permutation $\sigma \in S_n$, each permutation will be associated to its corresponding permutation matrix, so every permutation of codewords will be in matrix form, including calculation of fitness values P_σ :

$$P_\sigma = \sigma(I_n) \tag{6}$$

$\mathcal{M}_c \subset C$ is a codewords set, such that, $\forall c_i \in \mathcal{M}_c$:

$$c_i H^T = 0 \tag{7}$$

M_{S_c} is matrix where its rows formed by codewords:

$$M_{S_c} = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_k \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ c_{k1} & c_{k2} & \dots & c_{kn} \end{bmatrix} \tag{8}$$

applying the action of S_n on $M_{S_c}, \forall \sigma \in S_n$ s.t:

$$\sigma(M_{S_c}) = M_{S_c} P_\sigma = \begin{bmatrix} c_{\sigma(11)} & c_{\sigma(12)} & \dots & c_{\sigma(1n)} \\ c_{\sigma(21)} & c_{\sigma(22)} & \dots & c_{\sigma(2n)} \\ \vdots & \vdots & \vdots & \vdots \\ c_{\sigma(k1)} & c_{\sigma(k2)} & \dots & c_{\sigma(kn)} \end{bmatrix} \tag{9}$$

$$\sigma(M_{S_c})H^T = \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_k \end{bmatrix}, \text{ where H is the sparse parity-check matrix} \tag{10}$$

The permutation of M_{S_c} columns will generate another matrix of codewords if $\sigma(M_{S_c})H^T = 0$ (5). The selection of best permutations (individuals) will be based on the fitness values of permutations using the fitness function which is defined as follows:

$$f_\sigma = \sum_{i=1}^k wt(s_i) \tag{11}$$

where s_i is the syndrome of a codeword c_i [25], and wt is the weight. The selection operator will need the values of each permutation which is calculated using the fitness function (11) in order to select that permutation or not.

The Figure 3 shows the crossover operator which bases on the composition, which is chosen in order to ensure that all produced individuals within the search space and elements of S_n without relying on mutation due to the mutation operator probability of which is very low. Also, our method will use the mutation operator that consists a swapping of two gene's position of an individual as figured in the Figure 4, this mutation type is chosen to enhance the convergence of the algorithm and to obtain new individual fitness of which are better.

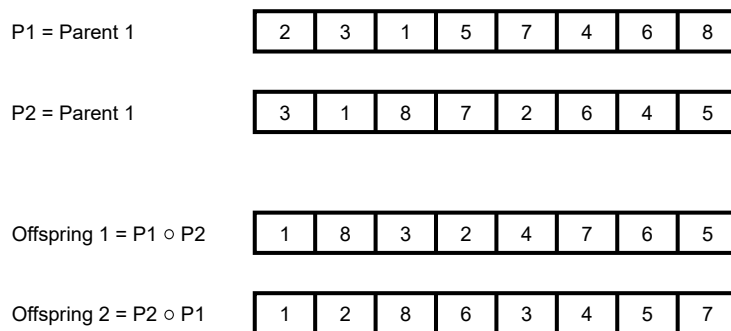


Figure 3. Crossover operator

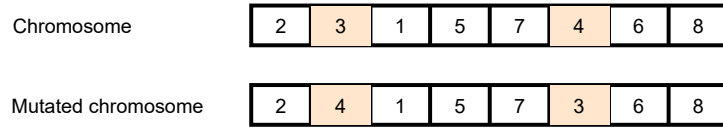


Figure 4. Mutation operator

3.2. The method inputs and outputs

The following is how the GA-based method works:

Inputs:

- A codewords set M_{S_c}
- The initial population size N_i
- The number of generations N_g
- The crossover probability p_c
- The mutation probability p_m

Outputs:

- The Automorphisms permutations set

The Figure 5 is the genetic algorithm-based method organigram where the selection operator uses the fitness function values (6), and the stochastic crossover and the stochastic mutation operators are explained in Figures 3 and 4.

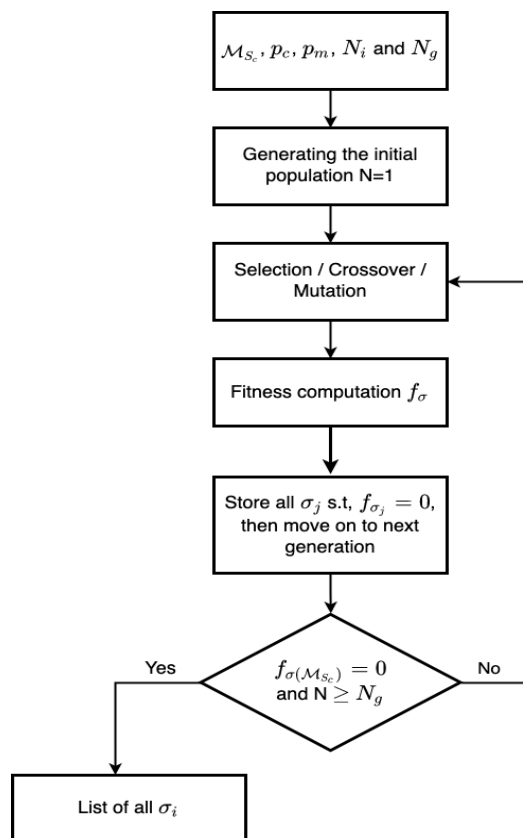


Figure 5. Genetic algorithm-based method organigram

4. RESULTS AND DISCUSSION

The results are obtained using parameters cited in Table 1. The permutation is presented as a list where the positions are numerated from 1 to the length of LDPC code. Every error correcting code has an automorphisms group, therefore the set of automorphism permutations set exist for LDPC codes. Figure 6 contains 160 automorphisms permutations produced by our GA-based method for [8,4,2] LDPC code and 12 automorphisms permutations for [16,8,3] LDPC code listed in the Figure 7.

Table 1. Paramters of GA-based method

Parameter	Value
Initial population size	200
Selection	elitism
Crossover rate	0.85
Mutation rate	0.02
Number of generations	30

[1, 2, 4, 3, 8, 6, 7, 5]	[1, 2, 4, 8, 3, 6, 7, 5]	[1, 2, 8, 4, 5, 6, 7, 3]	[1, 2, 8, 5, 4, 6, 7, 3]	[1, 3, 2, 5, 4, 8, 7, 6]	[1, 3, 5, 6, 2, 8, 7, 4]	[1, 3, 6, 5, 4, 8, 7, 2]
[1, 4, 3, 2, 6, 5, 7, 8]	[1, 4, 3, 6, 2, 5, 7, 8]	[1, 4, 6, 8, 3, 5, 7, 2]	[1, 4, 8, 2, 6, 5, 7, 3]	[1, 5, 2, 8, 3, 4, 7, 6]	[1, 5, 3, 2, 6, 4, 7, 8]	[1, 5, 3, 6, 2, 4, 7, 8]
[1, 6, 3, 4, 5, 2, 7, 8]	[1, 6, 8, 4, 5, 2, 7, 3]	[1, 8, 5, 2, 6, 3, 7, 4]	[1, 8, 5, 6, 2, 3, 7, 4]	[1, 8, 6, 5, 4, 3, 7, 2]	[2, 1, 3, 4, 5, 7, 6, 8]	[2, 1, 3, 5, 4, 7, 6, 8]
[2, 1, 4, 8, 3, 7, 6, 5]	[2, 1, 5, 3, 8, 7, 6, 4]	[2, 3, 1, 5, 4, 8, 6, 7]	[2, 3, 7, 4, 5, 8, 6, 1]	[2, 4, 1, 3, 8, 5, 6, 7]	[2, 4, 7, 3, 8, 5, 6, 1]	[2, 4, 7, 8, 3, 5, 6, 1]
[2, 5, 1, 3, 8, 4, 6, 7]	[2, 5, 1, 8, 3, 4, 6, 7]	[2, 5, 3, 1, 7, 4, 6, 8]	[2, 7, 3, 5, 4, 1, 6, 8]	[2, 7, 4, 8, 3, 1, 6, 5]	[2, 7, 5, 3, 8, 1, 6, 4]	[2, 7, 5, 8, 3, 1, 6, 4]
[2, 8, 1, 5, 4, 3, 6, 7]	[2, 8, 4, 1, 7, 3, 6, 5]	[2, 8, 5, 1, 7, 3, 6, 4]	[3, 1, 2, 5, 4, 7, 8, 6]	[3, 1, 4, 2, 6, 7, 8, 5]	[3, 1, 6, 5, 4, 7, 8, 2]	[3, 2, 1, 4, 5, 6, 8, 7]
[3, 2, 1, 5, 4, 6, 8, 7]	[3, 2, 4, 1, 7, 6, 8, 5]	[3, 2, 4, 7, 1, 6, 8, 5]	[3, 4, 1, 2, 6, 5, 8, 7]	[3, 4, 1, 6, 2, 5, 8, 7]	[3, 4, 7, 6, 2, 5, 8, 1]	[3, 5, 6, 1, 7, 4, 8, 2]
[3, 5, 7, 2, 6, 4, 8, 1]	[3, 6, 1, 4, 5, 2, 8, 7]	[3, 6, 4, 1, 7, 2, 8, 5]	[3, 6, 7, 4, 5, 2, 8, 1]	[3, 7, 5, 2, 6, 1, 8, 4]	[4, 1, 2, 8, 3, 7, 5, 6]	[4, 1, 3, 2, 6, 7, 5, 8]
[4, 1, 3, 6, 2, 7, 5, 8]	[4, 1, 6, 3, 8, 7, 5, 2]	[4, 1, 8, 6, 2, 7, 5, 3]	[4, 2, 1, 3, 8, 6, 5, 7]	[4, 2, 1, 8, 3, 6, 5, 7]	[4, 2, 7, 8, 3, 6, 5, 1]	[4, 2, 8, 7, 1, 6, 5, 3]
[4, 3, 1, 2, 6, 8, 5, 7]	[4, 6, 1, 3, 8, 2, 5, 7]	[4, 6, 1, 8, 3, 2, 5, 7]	[4, 6, 8, 1, 7, 2, 5, 3]	[4, 7, 2, 3, 8, 1, 5, 6]	[4, 7, 8, 2, 6, 1, 5, 3]	[4, 7, 8, 6, 2, 1, 5, 3]
[5, 1, 8, 4, 5, 7, 2, 3]	[5, 1, 2, 3, 8, 7, 4, 6]	[5, 1, 2, 8, 3, 7, 4, 6]	[5, 1, 3, 6, 2, 7, 4, 8]	[5, 2, 1, 8, 3, 6, 4, 7]	[5, 2, 3, 7, 1, 6, 4, 8]	[5, 2, 8, 1, 7, 6, 4, 3]
[5, 3, 2, 7, 1, 8, 4, 6]	[5, 3, 6, 1, 7, 8, 4, 2]	[5, 3, 6, 7, 1, 8, 4, 2]	[5, 6, 1, 8, 3, 2, 4, 7]	[5, 6, 8, 1, 7, 2, 4, 3]	[5, 7, 2, 3, 8, 1, 4, 6]	[5, 8, 1, 2, 6, 3, 4, 7]
[5, 8, 1, 6, 2, 3, 4, 7]	[5, 8, 2, 1, 7, 3, 4, 6]	[5, 8, 2, 7, 1, 3, 4, 6]	[5, 8, 7, 2, 6, 3, 4, 1]	[5, 8, 7, 6, 2, 3, 4, 1]	[6, 1, 5, 3, 8, 7, 2, 4]	[6, 1, 5, 8, 3, 7, 2, 4]
[6, 1, 8, 4, 5, 7, 2, 3]	[6, 1, 8, 5, 4, 7, 2, 3]	[6, 3, 1, 4, 5, 8, 2, 7]	[6, 3, 1, 5, 4, 8, 2, 7]	[6, 3, 7, 5, 4, 8, 2, 1]	[6, 4, 1, 3, 8, 5, 2, 7]	[6, 4, 1, 8, 3, 5, 2, 7]
[6, 4, 3, 7, 1, 5, 2, 8]	[6, 4, 8, 7, 1, 5, 2, 3]	[6, 5, 7, 8, 3, 4, 2, 1]	[6, 7, 3, 5, 4, 1, 2, 8]	[6, 7, 5, 8, 3, 1, 2, 4]	[6, 7, 8, 5, 4, 1, 2, 3]	[6, 8, 1, 4, 5, 3, 2, 7]
[6, 8, 1, 5, 4, 3, 2, 7]	[6, 8, 4, 7, 1, 3, 2, 5]	[6, 8, 5, 1, 7, 3, 2, 4]	[6, 8, 5, 7, 1, 3, 2, 4]	[6, 8, 7, 4, 5, 3, 2, 1]	[7, 2, 3, 4, 5, 6, 1, 8]	[7, 2, 4, 3, 8, 6, 1, 5]
[7, 2, 4, 8, 3, 6, 1, 5]	[7, 2, 5, 8, 3, 6, 1, 4]	[7, 3, 2, 4, 5, 8, 1, 6]	[7, 3, 2, 5, 4, 8, 1, 6]	[7, 3, 4, 6, 2, 8, 1, 5]	[7, 3, 5, 2, 6, 8, 1, 4]	[7, 4, 2, 3, 8, 5, 1, 6]
[7, 4, 2, 8, 3, 5, 1, 6]	[7, 4, 3, 2, 6, 5, 1, 8]	[7, 4, 3, 6, 2, 5, 1, 8]	[7, 4, 6, 3, 8, 5, 1, 2]	[7, 4, 8, 2, 6, 5, 1, 3]	[7, 5, 3, 2, 6, 4, 1, 8]	[7, 5, 6, 3, 8, 4, 1, 2]
[7, 5, 6, 8, 3, 4, 1, 2]	[7, 5, 8, 2, 6, 4, 1, 3]	[7, 6, 3, 5, 4, 2, 1, 8]	[7, 6, 4, 3, 8, 2, 1, 5]	[7, 6, 4, 8, 3, 2, 1, 5]	[7, 6, 5, 8, 3, 2, 1, 4]	[7, 6, 8, 5, 4, 2, 1, 3]
[7, 8, 4, 2, 6, 3, 1, 5]	[7, 8, 5, 2, 6, 3, 1, 4]	[8, 1, 4, 2, 6, 7, 3, 5]	[8, 1, 6, 5, 4, 7, 3, 2]	[8, 2, 4, 1, 7, 6, 3, 5]	[8, 2, 5, 7, 1, 6, 3, 4]	[8, 2, 7, 4, 5, 6, 3, 1]
[8, 4, 1, 2, 6, 5, 3, 7]	[8, 4, 2, 1, 7, 5, 3, 6]	[8, 4, 6, 1, 7, 5, 3, 2]	[8, 5, 1, 2, 6, 4, 3, 7]	[8, 5, 2, 1, 7, 4, 3, 6]	[8, 5, 2, 7, 1, 4, 3, 6]	[8, 5, 6, 1, 7, 4, 3, 2]
[8, 5, 6, 7, 1, 4, 3, 2]	[8, 5, 7, 6, 2, 4, 3, 1]	[8, 6, 1, 5, 4, 2, 3, 7]	[8, 6, 4, 1, 7, 2, 3, 5]	[8, 6, 7, 4, 5, 2, 3, 1]	[8, 6, 7, 5, 4, 2, 3, 1]	[8, 7, 2, 4, 5, 1, 3, 6]
[8, 7, 2, 5, 4, 1, 3, 6]	[8, 7, 4, 2, 6, 1, 3, 5]	[8, 7, 4, 6, 2, 1, 3, 5]	[8, 7, 5, 2, 6, 1, 3, 4]	[8, 7, 6, 4, 5, 1, 3, 2]	[8, 7, 6, 5, 4, 1, 3, 2]	[8, 7, 6, 5, 4, 1, 3, 2]

Figure 6. Automorphisms set of [8,4,2] LDPC code

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 16, 12, 13, 14, 15, 11]	[1, 2, 3, 4, 5, 6, 7, 9, 8, 10, 11, 12, 13, 14, 15, 16]
[1, 2, 3, 4, 5, 6, 7, 9, 8, 10, 16, 12, 13, 14, 15, 11]	[1, 2, 3, 4, 8, 6, 7, 5, 9, 10, 11, 12, 13, 14, 15, 16]	[1, 2, 3, 4, 8, 6, 7, 5, 9, 10, 16, 12, 13, 14, 15, 11]
[1, 2, 3, 4, 8, 6, 7, 9, 5, 10, 11, 12, 13, 14, 15, 16]	[1, 2, 3, 4, 8, 6, 7, 9, 5, 10, 16, 12, 13, 14, 15, 11]	[1, 2, 3, 4, 9, 6, 7, 5, 8, 10, 11, 12, 13, 14, 15, 16]
[1, 2, 3, 4, 9, 6, 7, 5, 8, 10, 16, 12, 13, 14, 15, 11]	[1, 2, 3, 4, 9, 6, 7, 8, 5, 10, 11, 12, 13, 14, 15, 16]	[1, 2, 3, 4, 9, 6, 7, 8, 5, 10, 16, 12, 13, 14, 15, 11]

Figure 7. Automorphisms set of [16,8,4] LDPC code

To be mentioned, each combination of two automorphisms permutations is an automorphism permutation, if the set contains all generators of automorphisms group, then we can obtain the others automorphisms permutations easily. The Table 2 shows statistical measures of 32 runs of GA-based method for [8,4,2] LDPC code, which shows the efficiency of our method for finding an important automorphisms set, in some runs, we get an important number of automorphisms in few number generations (set of 160 Automorphisms permutations in 8 generations).

Table 2. The statistical measures

Mean	Median	Standard deviation	Best	Worst
151.09	152.5	11.09	160	104




5. CONCLUSION

In this paper, the genetic algorithm-based method has been proposed for finding an important automorphisms set of a given LDPC code, which can be used in improving their decoding algorithms (the hard decision algorithm and the soft decision algorithm). It showed good results for LDPC codes in the short block length regime. Our future work is to optimize our method and its genetic parameters, and combining it with the GSK algorithm in order to process long LDPC codes.




REFERENCES

- [1] H. Abdelfattah and D. Harzalla, "On the Automorphism Group of Some Classes of Systematic Codes," *Asian Journal of Mathematics and Computer Research*, vol. 13, no. 2, 2016.
- [2] Elsevier, "4 Finite fields," *North-Holland Mathematical Library*, vol. 16, pp. 93–124, 1977, doi:10.1016/S0924-6509(08)70529-4.
- [3] R. G. Gallager, "Low-density parity-check codes," *IRE Transactions on Information Theory*, vol. 8, no. 1, pp. 21–28, 1962, doi: 10.1109/TIT.1962.1057683.
- [4] D. J. C. MacKay and R. M. Neal, "Near Shannon limit performance of low density parity check codes," *Electronics Letters*, vol. 32, no. 18, 1996, pp. 1645–1646, 1996, doi: 10.1049/el.19961141.
- [5] Z. Tu and S. Zhang, "Overview of LDPC Codes," In *7th IEEE International Conference on Computer and Information Technology (CIT 2007)*, 2007, pp. 469–474, doi: 10.1109/CIT.2007.7.
- [6] J. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1992.
- [7] M. P. Cuéllar, J. Gómez-Torrecillas, F. J. Lobillo, and G. Navarro, "Genetic algorithms with permutation-based representation for computing the distance of linear codes," *Swarm and Evolutionary Computation*, vol. 60, 2021, doi: 10.1016/j.swevo.2020.100797.
- [8] C. H. Papadimitriou and K. Steiglitz, *Combinatorial optimization: algorithms and complexity*. Prentice Hall, USA, 1982.
- [9] A. J. Umbarkar and P. D. Sheth, "Crossover operators in genetic algorithms: a review," *ICTACT journal on soft computing*, vol. 6, no. 1, pp. 1083-1092, 2015, doi: 10.21917/ijsc.2015.0150.
- [10] M. Basmassi, L. Chentoufi, and J. Alami Chentoufi, "A novel greedy genetic algorithm to solve combinatorial optimization problem," *The International archives of the photogrammetry, remote sensing and spatial information sciences*, vol. 44, pp. 117– 120, 2020, doi: 10.5194/isprs-archives-XLIV-4-W3-2020-117-2020.
- [11] D. N. Mudaliar and N. K. Modi, "Applying m-Mutation Operator in Genetic Algorithm to Solve Permutation Problems," In *2019 IEEE International Conference on System, Computation, Automation and Networking (ICSCAN)*, 2019, pp. 1–5, doi: 10.1109/IC-SCAN.2019.8878867.
- [12] G. Rajappa, "Solving Combinatorial Optimization Problems Using Genetic Algorithms and Ant Colony Optimization," Ph. D. Dissertation, University of Tennessee, USA, 2012. [Online]. Available: https://trace.tennessee.edu/cgi/viewcontent.cgi?article=2657&context=utk_graddiss.
- [13] S. M. Elsayed, R. A. Sarker, and D. L. Essam, "A genetic algorithm for solving the CEC'2013 competition problems on real-parameter optimization," In *2013 IEEE Congress on Evolutionary Computation*, 2013, pp. 356–360, doi: 10.1109/CEC.2013.6557591.
- [14] K. Phiwhorm and K. R. Saikaew, "A Hybrid Genetic Algorithm with Multi-Parent Crossover in Fuzzy Rule-Based," In *International Journal of Machine Learning and Computing*, Oct. 2017, vol. 7, pp. 114–117, doi: 10.18178/ijmlc.2017.7.5.631.
- [15] L. Huliyanyskiy and I. Riasna, "Formalization and Classification of Combinatorial Optimization Problems," In *Optimization Methods and Applications*, Jan. 2017, pp. 239–250, doi: 10.1007/978-3-319-68640-011.
- [16] S. Yakovlev, O. Kartashov, and O. Yarovaya, "On Class of Genetic Algorithms in Optimization Problems on Combinatorial Configurations," In *2018 IEEE 13th International Scientific and Technical Conference on Computer Sciences and Information Technologies (CSIT)*, 2018, vol. 1, pp. 374–377, doi: 10.1109/STC-CSIT.2018.8526746.
- [17] A. Wagdy, A. Hadi, and A. Khater, "Gaining-sharing knowledge based algorithm for solving optimization problems: a novel nature-inspired algorithm" *International Journal of Machine Learning and Cybernetics*, vol. 11, Jul. 2020, doi: 10.1007/s13042-019-01053-x.
- [18] P. Agrawal, G. Talari, and A. Wagdy, "A novel binary gaining–sharing knowledge-based optimization algorithm for feature selection," *Neural Computing and Applications*, vol. 33, no. 11, pp. 1–20, Jun. 2021, doi: 10.1007/s00521-020-05375-8.
- [19] A. Wagdy, P. Agrawal, and G. Talari, "Chaotic gaining sharing knowledge-based optimization algorithm: an improved metaheuristic algorithm for feature selection," *Soft Computing*, pp. 1-24, May 2021, doi: 10.1007/s00500-021-05874-3.
- [20] A. Hassanat, K. Almohammadi, E. Alkafaween, E. Abunawas, A. Hammouri, and V. B. Prasath, "Choosing Mutation and Crossover Ratios for Genetic Algorithms—A Review with a New Dynamic Approach," *Information*, vol. 10, no. 12, 2019, doi: 10.3390/info10120390.
- [21] B. Koohestani, "A crossover operator for improving the efficiency of permutation-based genetic algorithms," *Expert Systems with Applications*, vol. 151, 2020, doi: 10.1016/j.eswa.2020.113381.
- [22] C. Liu and A. Kroll, "Performance impact of mutation operators of a subpopulation-based genetic algorithm for multi-robot task allocation problems," *SpringerPlus*, vol. 5, Aug. 2016, doi: 10.1186/s40064-016-3027-2.
- [23] S. Nouh, I. Chana, and M. Belkasmi, "Decoding of Block Codes by using Genetic Algorithms and Permutations Set," *International Journal of Communication Networks and Information Security (IJCNIS)*, vol. 5, no. 3, 2013, doi: 10.54039/ijcnis.v5i3.428.
- [24] M. Askali, A. Azouaoui, S. Nouh, and M. Belkasmi, "On the Computing of the Minimum Distance of Linear Block Codes by Heuristic Methods," 2013, *ArXiv abs/1303.4375*.
- [25] S. Ball, "A Course in Algebraic Error-Correcting Codes," in *Springer Nature*, 2020. [Online]. Available: <https://link.springer.com/book/10.1007/978-3-030-41153-4>.




BIOGRAPHIES OF AUTHORS

Bellfkih El Mehdi    was born on 20 July 1989 in El Jadida, Morocco. He received his license in Applied Mathematics and Master in Teaching and Training Professions in Mathematics from Ibn Tofail University in 2016 and 2018, respectively. Currently, He is pursuing his study in PhD in coding theory in Hassan II university. His research study is in error Correcting Codes. He can be contacted at email: elmehdi.bellfkih@gmail.com.






Said Nouh    is Professor at Faculty of sciences Ben M'Sik, Hassan II university, Casablanca, Morocco. He had PhD in computer sciences at National superior School of Computer Science and Systems Analysis (ENSIAS), Rabat, Morocco in 2014. His current research interests telecommuni-cations, Information and Coding Theory, Machine Learning, deep Learning and Data Sciences. He can be contacted at email: nouh_ensias@yahoo.fr.






Imrane Chems eddine Idrissi    was born on 10 April 1981 in Casablanca, Morocco. He received his Master in data science and big data from Mohammed V University in 2019. Currently, He is pursuing his study in PhD in machine learning and error correcting codes in Hassan II university. His research study is applying the machine learning techniques in error correcting code field. He can be contacted at email: imran.chems@gmail.com.






Abdelaziz Ettaoufik    is Professor in the department of mathematics and informatics at Faculty of sciences Ben M'Sik, Hassan II University, Morocco. His research field of interest includes data warehouse, cloud computing and optimisation. He can be contacted at email: aet-taoufik@gmail.com.



Khalid Louartiti    born in Taounate, Morocco. He received his PhD degree from Sidi Mohamed Ben Abdellah University, Fes, Morocco. Currently works as a Professor at National School of Applied Sciences (ENSA), Tetouan, Morocco. His research field of interest includes graph theory, modules, ideals, commutative algebra and Amalgamated algebra. He can be contacted at email: lokha2000@hotmail.com.



Jamal Mouline    born in Ouazzane, Morocco. He received his PhD degree from Provence University, France. Currently works as a Professor in the department of mathematics and informatics at Hassan II University, Morocco. His research field of interest includes fixed point theory and combinatorial theory. He can be contacted at email: mouline61@gmail.com.