Turk. J. Math. Comput. Sci.
14(1)(2022) 191–200
© MatDer
DOI : 10.47000/tjmcs.982426

Mutation-Based Algebraic Artificial Bee Colony Algorithm for Computing the Distance of Linear Codes

Adrian Korban [1], Serap Şahinkaya[2], Deniz Ustun[3,*]

[1] *Department of Mathematical and Physical Sciences, University of Chester, Chester, England.*
[2]*Department of of Natural and Mathematical Sciences, Faculty of Engineering, Tarsus University, 33400, Mersin, Turkey.*
[3]*Department of Computer Engineering, Faculty of Engineering, Tarsus University, 33400, Mersin, Turkey.*

Received: 13-08-2021 • Accepted: 11-01-2022

Abstract. Finding the minimum distance of linear codes is a non-deterministic polynomial-time-hard problem and different approaches are used in the literature to solve this problem. Although, some of the methods focus on finding the true distances by using exact algorithms, some of them focus on optimization algorithms to find the lower or upper bounds of the distance. In this study, we focus on the latter approach. We first give the swarm intelligence background of artificial bee colony algorithm, we explain the algebraic approach of such algorithm and call it the algebraic artificial bee colony algorithm (A-ABC). Moreover, we develop the A-ABC algorithm by integrating it with the algebraic differential mutation operator. We call the developed algorithm the mutation-based algebraic artificial bee colony algorithm (MBA-ABC). We apply both; the A-ABC and MBA-ABC algorithms to the problem of finding the minimum distance of linear codes. The achieved results indicate that the MBA-ABC algorithm has a superior performance when compared with the A-ABC algorithm when finding the minimum distance of Bose, Chaudhuri, and Hocquenghem (BCH) codes (a special type of linear codes).

2010 AMS Classification: 11T71, 14G50, 68P30, 78M50, 80M50, 90C27

Keywords: Minimum distance, minimum-weight codeword, optimisation, heuristic, artificial bee colony algorithm.

1. Introduction

Coding theory is one of the fundamental tools of Information theory. Codes are used to convert messages into a sequence of binary digits, then those messages are transmitted via communication channels. Although communications channels are safer these days than in the past, codes can still be disturbed during the transmission. Therefore, for a reliable communication, it is very important for codes to have the feature of determining and correcting the errors. The error correction capability of a code depends on the notion of the distance. Thus, for a given block length n and dimension k, the code is desired to have the minimum distance as large as possible. Finding the minimum distance of a linear code is the major task for determining the error correction capability of the code. However, it is not an easy task to compute the minimum distance of codes of large sizes. Vardy showed that, the computation of the minimum weight of a non-zero binary codeword is a non-deterministic polynomial-time (NP)-complete [28]. Although, it is possible to determine the minimum distance of linear codes, computation is getting harder when the size and block length of the code grow. There exist different approaches in the literature to determine the minimum distance of linear codes. Although some of them focus on finding the true distances by using the exact algorithms [7, 16, 20], some of them focus on the approximate methods to find the lower or upper bounds for the distance. For the latter approach, several

*Corresponding Author
Email addresses: adrian3@windowslive.com (A. Korban), serap@tarsus.edu.tr (S. Şahinkaya), denizustun@tarsus.edu.tr (D. Ustun)

heuristic search algorithms like a genetic algorithms simulated annealing, Metropolis algorithm, hill climbing, tabu search, ant colony optimization were applied to this problem [1,4,5,8,9,11,12,15].

Swarm based intelligence optimization algorithms have been a very popular method among many researchers for solving the complex optimization problems in recent years [27]. Although genetic algorithms have been used many times for the problem of finding the minimum distance of linear codes, this is not the case for the algorithms based on the swarm intelligence. As far as our knowledge, only the ant colony algorithm was used to optimize this problem in terms of swarm intelligence [9, 11]. The artificial bee colony (ABC) algorithm [17–19] is a newly defined swarm based intelligence optimization algorithm that is developed by the inspiration of honey bees foraging behaviours. The ABC algorithm, based on swarm intelligence, is very effective for solving the difficult optimization problems [25, 26]. Although, the ABC algorithm was designed for numerical problems and finding the minimum distance is a combinatorial problem, this algorithm can not be used directly for the considered problem. Therefore, we introduce the algebraic-based scheme, inspired by [23], for the ABC algorithm, which we call the algebraic artificial bee colony (A-ABC) algorithm. Moreover, we develop the A-ABC algorithm by integrating the algebraic differential mutation operator in the employed and onlooker bee phases of the A-ABC algorithm and we call this developed algorithm, the mutational-based algebraic artificial bee colony (MBA-ABC) algorithm. We show that the MBA-ABC algorithm outperforms the A-ABC algorithm . In order to control the effectiveness of the presented algorithm, Bose, Chaudhuri, and Hocquenghem (BCH) codes are used since these are the standard codes with well known minimum distance values [6, 22].

In most of the existing literature, researchers have attempted to solve the minimum distance problem by heuristic methods with the use of the codewords as a search space. But it is known from [14] that for a generator matrix G of an $[n, k]$ binary linear code C, I_iG is also generator matrix of C, where I_i is some $k \times k$ invertible matrices. By this fact, generator matrices have been started to be used in minimum distance problem and as far as our knowledge, [1, 12, 13] are the only papers in which the generator matrices are used as a search space. In [1], generator matrices were used instead of codewords and it was showed that the generator matrix space is a better search space when compared to the codeword space. In that paper, for a given generator matrix G of C, generator search space was considered as a set of of $k \times k$ invertible matrices. For any invertible matrix I_i the cost of I_i was defined as the least Hamming weight of the rows of I_iG. In [12], authors proved that for a given generator matrix G of an $[n, k]$ binary linear code C, there exists a permutation $x \in S_n$ such that the reduced row echelon form of GP_x , where P_x is the permutation matrix for a permutation x, satisfies the Hamming weight of some of its rows. Therefore, in that work, for a given generator matrix G of C, generator search space was considered as a set of permutations, where permutations encoded as arrays of size n and each array contains a column permutation of the initial generating matrix G. The cost of permutation x was defined as the least Hamming weight of the rows of the reduced row echelon form of GP_x. Also, in that paper, the authors used the algebraic crossover operator for the solution set. In our paper, for a given generator matrix G of a code C, we consider the permutation matrices P_x, where each of the permutation matrices is obtained by shuffling the columns of the identity matrices, as a search space. The cost of P_x is considered as the least Hamming weight of the rows of the reduced row echelon form of GP_x. Generator matrices of the BCH codes are taken from the database of the software MAGMA [10].

The rest of the paper is organised as follows. In Section 2, we give preliminary definitions and notions on linear codes and we give a brief history of the ABC algorithm. In Section 3, we explain the algebraic approach of the ABC algorithm and develop this algorithm by integrating it with the algebraic differential mutation operator. Then we implement these two algorithms for the considered problem. In Section 4, we tabulate and compare our results accordingly to the algorithms used.

2. Preliminaries

Some basic definitions and notions that will be used in later sections are recalled.

2.1. **Linear Codes.**

Definition 2.1. A binary linear code C is an k-dimensional vector subspace of the n-dimensional vector space $\mathbb{F}_2^n$ over the finite field $\mathbb{F}_2$.

The minimum distance d of a linear code C is determined by: $d = min_{i \neq j} d_H(\mathbf{c}_i, \mathbf{c}_j)$, where $d_H(\mathbf{c}_i, \mathbf{c}_j)$ denotes the Hamming distance between codewords $\mathbf{c}_i, \mathbf{c}_j \in C$. Hamming distance is defined as the number of positions that differs between two distinct codewords. The weight of a codeword $\mathbf{c} \in C$ equals to the number of non-zero entry in it, for a binary codeword it is just the number of ones in the codeword. A linear code C is given by parameters $[n, k, d]$, where

n is the length, k is the dimension and d is the minimum distance of the code. Any codeword $\mathbf{c} \in C$ can be obtained by a linear combination of k-basis codewords, that is, $\mathbf{c} = \boldsymbol{\alpha} G$, where G is $k \times n$ generator matrix and $\boldsymbol{\alpha}$ is an k-tuple vector which is also called as an information vector and $\mathbf{c}$ is an n-tuple vector, called, codeword.

For a linear code C, a non-zero codeword of minimum Hamming weight is called a minimum-weight codeword. It is clear by definitions that the minimum distance of a linear code equals to the minimum weight of a non-zero codeword in the code. A linear code can be presented by providing either a basis or a generator matrix whose rows form a basis for the code C.

Error detecting and correcting capability is a very important feature of a code and it is determined by the minimum distance of the code. More precisely, for a given $[n, k, d]$ linear code C, it can detect $d - 1$ errors and correct $\lfloor \frac{d-1}{2} \rfloor$ errors [21]. Therefore, for a given block of length n and dimension k, the code is desired to have the minimum distance as large as possible. One of the main problems of coding theory is the Minimum Weight Codeword Problem which determines the number of codewords of weight M or less, in an $[n, k]$ linear code C for a given integer M.

2.2. **Artificial Bee Colony (ABC) Algorithm.** The approach of swarm intelligence has been a very popular method among many researchers for solving the complex optimization problems in recent years [27]. The Artificial bee colony (ABC) algorithm [17–19] is a new optimization method that is developed by the inspiration of honey bees foraging behaviors. The ABC algorithm, based on swarm intelligence, is very effective for solving the difficult optimization problems [25, 26]. In ABC, the bee colony includes three types of bees that are the employed, onlooker and scout. Also, there are some assumptions in the processing of the algorithm. The number of the food source is equal to the half size of the colony. The food sources are represented as the possible solutions and each bee is assigned to one food source. Artificial bees find the optimal solutions by flying in a multidimensional search space. The employed bees are appointed to a particular food source and this depends on their experiences. Then, the onlooker bees get information about the food source by watching the dance of the employed bees within the hive and they select the food sources by this information. Also, they adjust their positions in the search space. Any bee consuming its food source becomes a scout bee and the scout bees perform a random search process for detecting a new food source. In the initial, the scout bees discover the positions of the food sources and then these resources are consumed by the employed and onlooker bees. In the ABC algorithm, a possible solution of the problem in the search space is represented by the position of a food source. There is a correspondence between the nectar quantity of a food source and the quality (fitness) of the associated solution. The pseudocode of the ABC algorithm is given in Algorithm 1.

Algorithm 1. Pseudocode of ABC Algorithm

```
Generate initial population x_i(i=1,2,...,SN)
Evaluate the fitness fit_i(x_i) of the population
Set iteration to 1
repeat
    for each Employed Bee do
        Generate a new solution v_i by using Equation (2.1)
        Compute its fitness value fit_i(v_i) by using Equation (2.2)
        Apply greedy selection process
    end for
    Compute probability value p_i for the solution x_i by using Equation (2.3)
    for each Onlooker Bee do
        Choose a solution x_i depending on p_i
        Generate a new solution v_i by using Equation (2.1)
        Compute its fitness value fit_i(v_i) by using Equation (2.2)
        Apply greedy selection process
    end for
    if There is an abandoned solution for the Scout Bee then
        Replace old solution with new solution produced randomly as initial phase
    end if
    Save the best solution so far
    iteration=iteration+1
until iteration=MaximumIteration
```

The ABC algorithm phases are given as follows:

Initialization Phase: Initial food source are determined by randomly chosen scouts.
REPEAT
Employed Bees Phase: Employed bees are sent to the food sources for the determination of nectar amounts.
Onlooker Bees Phase: Onlooker bees are appointed to food sources according to the calculated probability value of the sources.
Scout Bees Phase: Randomly chosen scout bee is send for discovering new food sources if a source is abandoned by an employed bee.
UNTIL(Cycle=Maximum Cycle Number)

After the initial phase, the employed, onlooker and scout phases are performed for a fixed number of iterations. An employed bee generates a new position (solution) by modification the old position. In the ABC algorithm, the equation for the new position is given as follows:

$$v_{i,j} = x_{i,j} + \phi_{i,j}(x_{i,j} - x_{k,j}), \tag{2.1}$$

where $k \in \{1, 2, \ldots, SN\}$ and $j \in \{1, 2, \ldots, D\}$. Here k and j are randomly generated and k is different from i. $\phi_{i,j}$ is also a random number that belong to $[-1, 1]$. Then, the profitability (fitness value) of the new position is evaluated. The fitness values of the solution, generated by employed bee, is computed for minimizing the problems as following:

$$fit_i(x_i) = \begin{cases} \frac{1}{1+f_i(x_i)} & if \quad f_i(x_i) \geq 0 \\ 1 + |f_i(x_i)| & if \quad f_i(x_i) < 0, \end{cases} \tag{2.2}$$

where $f_i(x_i)$ is the objective function value of the solution x_i.

When a new solution is generated, a process called greedy selection based on comparing the new solution to the old solution is used. It tries to select the best solution between the new solution and the old solution to enhance the possible solutions in the population. If the fitness value of the new solution is higher than previous one, the bee memorizes the new solution and the employed bee abandons the old solution. Otherwise, the bee tries to improve the old solution in the next iteration. After completing the employed bees phase, the onlooker bee phase is performed by considering the probability values of all the solutions. In the onlooker bee phase, the employed bee shares the nectar information of food source with the onlooker bees on the dance area within hive. An onlooker bee appraises the nectar information, obtained from all the employed bees, and then it selects a food source according to the amount of the nectar. It performs this selection process by using the probability value computed from the fitness values. After completing the selection phase, the onlooker bee improves the new solution achieved by the previous solutions as in the employed bee phase. The probability value p_i with which x_i is chosen by an onlooker bee can be calculated by using the expression given

$$p_i = \frac{fit_i(x_i)}{\sum_{i=1}^{SN} fit_n(x_i)} \tag{2.3}$$

where fit_i is the fitness value of the solution which is proportional to the nectar amount of the food source in the position i, and SN is the number of food sources which is equal to the number of the employed bees. In the scout bee phase, when the nectar of any food source is consumed, that source is abandoned by the bee and then it transforms into a scout bee for finding a new food source randomly. The fixed number of iteration, called the "limit", is a major control parameter of the ABC algorithm for abandonment.

3. An Algebraic Approach of the ABC Algorithm

Although using the standard operator in the employed and onlooker bee phases exhibit the high performance for continuous problems, the standard ABC algorithm is not suitable for combinatorial problems. Therefore, an algebraic approach which is inspired from the paper of [23] is proposed for the ABC algorithm. In this section, we introduce the algebraic ABC (A-ABC) algorithm and the mutation based algebraic ABC (MBA-ABC) algorithm. The parameters of the algorithms, which are the population size, scale factor F, maximum number of iteration and limit value are given in Table 1.

Parameters	A-ABC	MBA-ABC
Population Size	100	100
Scale Factor F	-	0.6
Maximum Iteration	1000	1000
Limit	50	50

TABLE 1. Parameters of Proposed Algorithms

Let us recall the definition of the permutation matrix which is used for the search space. An $n\times n$ permutation matrix is a square binary matrix that has exactly one entry of 1 in each row and each column and 0s elsewhere. For a given generator matrix G of an $[n,k]$ binary BCH code, the A-ABC and MBA-ABC algorithms produce a distributed initial population, consisting of the permutation matrices by shuffling the columns of $n\times n$ identity matrix.

3.1. **Algebraic Artificial Bee Colony (A-ABC) Algorithm.** It is well known that, the genetic algorithms have been successively used for finding the minimum distance of linear codes [12] and the standard differential evaluation (DE) algorithm developed by Storn and Price in 1997 is a population-based genetic search strategy [24]. Therefore, it is natural to consider the DE algorithm for computing the minimum distance of linear codes but the standard DE algorithm is not suitable for this combinatorial problem. Thus, we consider the algebraic DE (A-DE) algorithm that was introduced in [23]. In [23], an algebraic differential mutation was given for a finitely generated group G with a binary operation $\star$. For every population individual x_i, mutant y_i is generated as follows:

$$y_i \leftarrow x_1 \oplus F_i \odot (x_2 \ominus x_3),$$

where $F_i \in (0,1]$ is the differential evaluation (DE) scale factor and x_1, x_2, x_3, are three randomly chosen distinct population individuals, all different from x_i. The operators $\oplus, \ominus, \otimes$ are the algebraic operators defined as follows, $x \oplus y = x \star y$, $x \ominus y = y^{-1} \star x$ and the multiplication $z = F \odot x$ satisfies that $|z| = \lceil F.|x| \rceil$, if $F \leq 1$, the sequence of generators in a minimal decomposition of z is a prefix of the sequence of generators in a minimal decomposition of x, vice versa, when $F > 1$.

For our problem, each population individual P_{x_i} is an $n\times n$ permutation matrix. Group, consisting of $n\times n$ permutation matrices with a usual matrix multiplication as a group operator is considered. For each population individual P_{x_i} a mutant P_{y_i} is generated according to

$$P_{y_i} \leftarrow P_{x_1} \oplus F \odot (P_{x_2} \ominus P_{x_3}), \tag{3.1}$$

where $F = 0.6$ is the fixed DE scale factor, $P_{x_1}, P_{x_2}, P_{x_3}$ are three randomly chosen distinct population individuals.

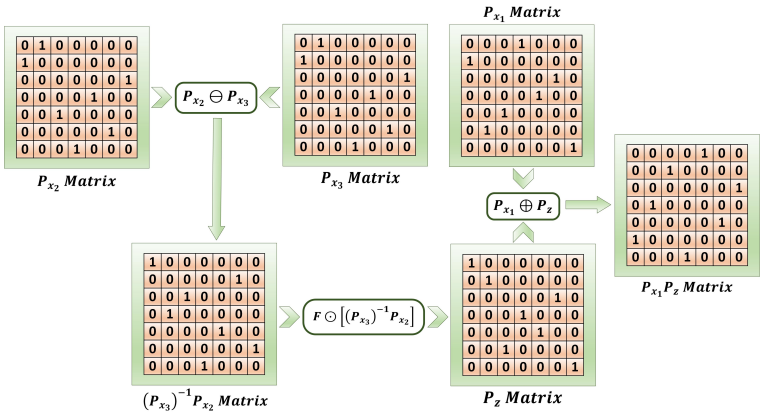

FIGURE 1. Algebraic differential mutation operator for permutation matrices

Algebraic operators $\oplus, \ominus, \otimes$ are defined as follows: $P_{x_1} \oplus P_z = P_{x_1}.P_z$, $P_{x_2} \ominus P_{x_3} = P_{x_3}^{-1}.P_{x_2}$, where "." is the usual matrix multiplication. It is noted that multiplication of the permutation matrices is a permutation matrix so permutation

matrices are obtained after applying the $\oplus$ and $\ominus$ operators. The magnitude of the permutation matrix is defined from the matrix $P_z = P_y^{-1}P_x$. More precisely, the magnitude $|P_z|$ is defined as a minimum number of the shuffles that are applied to the identity matrix for obtaining the matrix P_z. We simply calculate the number of shuffles from the following formula;

$$k = Tr(I_n - P_z) - 1 (\mathrm{mod}\ 2),$$

where I_n is an $n \times n$ identity matrix and Tr denotes the trace of a matrix. This definition makes sense since permutation matrices can be obtained by shuffling the columns of the $n \times n$ identity matrix. The multiplication $F \odot P_z$ is defined as follows: after obtaining the scaler number $k = \lceil F.|P_z| \rceil$, k columns of the $n \times n$ identity matrix is shuffled. The flowchart, given in Figure 1, illustrates the algebraic differential mutation for permutation matrices.

Unfortunately, calculations showed that the A-DE algorithm does not have a good performance for computing the minimum distance of linear codes. It is known that the ABC algorithm has a better performance than the DE algorithm for continuous problems [19]. It is natural to think that algebraic ABC algorithm can have a better performance than the algebraic DE algorithm for combinatorial problems. Inspiring from the algebraic approach of the DE algorithm we introduce the algebraic ABC (A-ABC) algorithm.

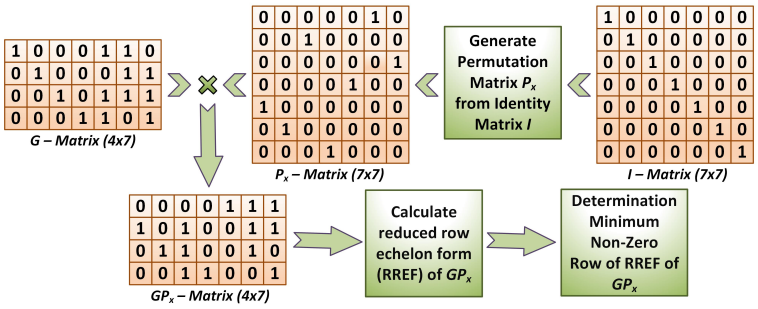

FIGURE 2. Generation process of the GP_x

The A-ABC algorithm has three phases like in the classical ABC algorithm given in previous section. In the A-ABC algorithm, each solution P_{x_i} is an $n \times n$ permutation matrix and the cost function f_i for each solution is calculated as the least Hamming weight of the rows of the reduced row echelon form of GP_{x_i}, where G is a generator matrix of the code. The flowchart, given in Figure 2), illustrates the generation process of GP_x and also the calculation of its cost.

In the initial phase, permutation matrices which represent possible solutions in the population are created randomly and then the cost values f_i are calculated for each solution. For each f_i the fitness value is calculated from Equation (3.2).

$$fit_i(P_{x_i}) = \frac{1}{1 + f_i(P_{x_i})} \tag{3.2}$$

where $f_i(P_{x_i})$ is the cost function of the solution P_{x_i}.

After the initial phase, the employed, onlooker and scout phases are performed for a fixed number of iterations. In the classical ABC algorithm, the equation for the new food source is given in Equation (2.1). Although multiplication of permutation matrices gives a permutation matrix, this property is not inherited by addition. That is, summation of two or more permutation matrices is not a permutation matrix. Therefore, the classical addition operator of Equation (2.1) can not be used for generating a new solution in the A-ABC algorithm. In this study, since the new food source is generated by the permutation matrices, Equation (2.1) should be rewritten from the algebraic perspective. The A-ABC algorithm is a permutation-based scheme of the standard ABC algorithm. The search operator in the standard ABC algorithm given in Equation (2.1) is converted to the algebraic operator as follows:

$$P_{v_i} \leftarrow P_{x_i} \oplus \phi_i \odot (P_{x_i} \ominus P_{x_k}), \tag{3.3}$$

where $\phi_i \in (0, 1]$ is a random number, P_{x_i} is the current solution and P_{x_k} is a randomly chosen distinct population individual different from P_{x_i}. The operators $\oplus, \ominus, \otimes$ are defined in the same way as in the A-DE algorithm. An

employed bee generates a new position (solution) by modification the old position using Equation (3.3) and then the profitability (fitness value) of the new position is evaluated by using Equation (3.2). If the fitness value of the new solution is higher than previous one, the bee memorizes this new solution and the employed bee abandons the old solution. Otherwise, the bee tries to improve the old solution in the next iteration. After completing the employed bees phase, the onlooker bee phase is performed by considering the probability values of all the solutions. Next, the onlooker bee phase is performed by considering the probability values of all the solutions. The probability value p_i with which P_{x_i} is chosen by an onlooker bee can be calculated by using the Equation (2.3). In the scout bee phase, when the nectar of any food source is consumed, that source is abandoned by the bee and than it transforms into a scout bee for finding a new food source at random, by the generation process given in Figure 2. The pseudocode of the A-ABC algorithm is given in Algorithm 2.

Algorithm 2. Pseudocode of A-ABC Algorithm

Generate initial population $P_{xi}(i=1,2,...,SN)$
Evaluate the fitness $fit_i(P_{xi})$ of the population by using Equation (3.2)
Set *iteration* to 1
repeat
    for each Employed Bee **do**
        Generate a new solution P_{vi} by using Equation (3.3)
        Compute its fitness value $fit_i(P_{vi})$ by using Equation (3.2)
        Apply greedy selection process
    end for
    Compute probability value p_i for the solution P_{xi} by using Equation (2.3)
    for each Onlooker Bee **do**
        Choose a solution P_{xi} depending on p_i
        Generate a new solution P_{vi} by using Equation (3.3)
        Compute its fitness value $fit_i(P_{vi})$ by using Equation (3.2)
        Apply greedy selection process
    end for
    if There is an abandoned solution for the Scout Bee **then**
        Replace old solution with new solution produced randomly as initial phase
    end if
    Save the best solution so far
    iteration=*iteration*+1
until *iteration*=MaximumIteration

3.2. **Mutation-Based Algebraic Artificial Bee Colony (MBA-ABC) Algorithm.** The performance of the standard optimization algorithms may not be sufficient to achieve a good solution for the same problems with the complex and difficult structure. In this case, the hybrid approaches have been developed and presented to the literature in order to increase the performance of the standard optimization algorithms. It is known that the hybrid ABC algorithms have better performances than the classical ABC algorithms [2, 3]. Therefore, it is natural to consider the same implication for the algebraic ABC algorithm. We develop an A-ABC algorithm by integrating the algebraic differential mutation operator from the A-DE algorithm to the employed and onlooker bee phases of the A-ABC algorithm and call this hybrid algorithm, the mutation-based algebraic artificial bee colony algorithm (MBA-ABC). More precisely, we use Equation (3.1) in place of Equation (3.3) in the employed and onlooker bee phases of the A-ABC algorithm. The MBA-ABC algorithm is illustrated in Figure 3 and the pseudocode of the MBA-ABC Algorithm is given in Algorithm 3.

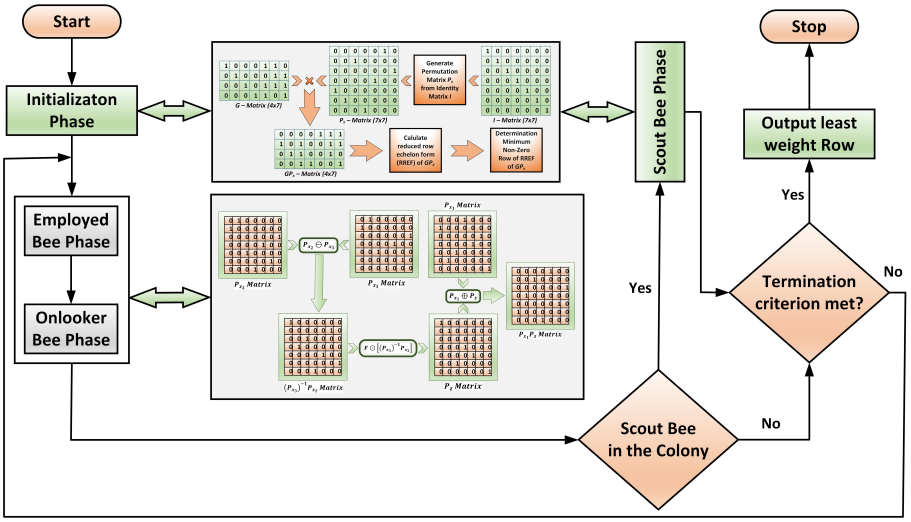

FIGURE 3. The Flowchart of MBA-ABC algorithm

Algorithm 3. Pseudocode of MBA-ABC Algorithm

Generate initial population $P_{xi}(i=1,2,...,SN)$
Evaluate the fitness $fit_i(P_{xi})$ of the population by using Equation (3.2)
Set *iteration* to 1
repeat
  for each Employed Bee **do**
    Generate a new solution P_{yi} by using Equation (3.1)
    Compute its fitness value $fit_i(P_{yi})$ by using Equation 3.2
    Apply greedy selection process
  end for
  Compute probability value p_i for the solution P_{xi} by using Equation (2.3)
  for each Onlooker Bee **do**
    Choose a solution P_{xi} depending on on p_i
    Generate a new solution P_{yi} by using Equation (3.1)
    Compute its fitness value $fit_i(P_{yi})$ by using Equation (3.2)
    Apply greedy selection process
  end for
  if There is an abandoned solution for the Scout Bee **then**
    Replace old solution with new solution produced randomly as initial phase
  end if
  Save the best solution so far
  iteration=*iteration*+1
until *iteration*=MaximumIteration

4. COMPARASION WITH PREVIOUS WORK

Two nature-inspired algebraic optimization algorithms are presented in this study. The proposed algorithms are run on a workstation with Intel Xeon 4.0 GHz processor and 64 GByte RAM. The parameters of the algorithms, which are the population size, scale factor F, maximum number of iteration and limit value are given in Table 1 and the achieved results by these algorithms are given in Table 2 for the BCH codes. In the literature, there have been some heuristic attempts to calculate the minimum distance of some specific BCH codes [1, 4, 5, 8, 9, 15]. The performance of the proposed optimization method is appraised and confirmed by a comparison with the GA-A, GA-B, GA, Hill Climbing,

Tabu Search, Ant Colony and Metropolis Algorithms given in Table 2. The algorithms are compared with the other algorithms to illustrate their superior performances. It can be obviously seen from Table 2 that the presented algorithms determine the exact minimum distance value for many BCH codes and they have an outstandingly better performance than the other optimization methods in the table. When the MBA-ABC algorithm is compared with the A-ABC algorithms, it is seen that the first one has much better performance. The differential mutation operator integrated to the employed and onlooker bee phases of the MBA-ABC algorithm, increases the diversity of the population randomly located in the search space. The diversity of population in the optimization algorithms affects the global and local search abilities of the nature-inspired optimization algorithms. In this study, the diversity of the MBA-ABC algorithm's population is increased by the differential mutation operator and in this way, the MBA-ABC is provided that it exhibits a higher performance than the A-ABC algorithms.

BCH Codes (n,k,d)	Askali's GA-A [4], [5]	Askali's GA-B [4], [5]	Wallis's GA [4], [15]	Hill Climbing [4], [5], [15]	Tabu Search [4], [5], [15], [8]	Ant Colony [9]	Metropolis Algorithm [1]	**A-ABC**	**MBA-ABC**
(127, 64, 21)	21	21	21	28	24	24	21	**21**	**21**
(127, 57, 23)	23	23	23	28	23	24	23	**23**	**23**
(127, 50, 27)	27	27	27	32	31	27	27	**27**	**27**
(255, 71, 59)	64	63	66	79	79	70	63	**61**	**61**
(255, 79, 55)	57	57	60	74	64	69	57	**55**	**55**
(255, 87, 53)	57	58	57	70	66	66	57	**53**	**53**
(255, 91, 51)	58	53	59	72	69	68	54	**51**	**51**
(255, 99, 47)	51	52	55	64	61	62	51	**47**	**47**
(255, 107, 45)	53	49	51	64	62	60	50	**45**	**45**
(255, 115, 43)	48	45	50	57	55	58	46	**43**	**43**
(511, 304, 51)	87	74	79	90	85	-	73	**66**	**64**
(511, 286, 55)	98	84	84	96	92	-	78	**76**	**71**
(511, 238, 75)	113	103	105	118	112	-	99	**95**	**95**
(511, 220, 79)	112	109	111	123	117	-	108	**104**	**101**
(511, 184, 91)	111	127	128	135	140	-	120	**120**	**119**
(511, 166, 95)	143	135	137	152	140	-	128	**131**	**120**
(511, 121, 117)	159	155	152	163	163	-	148	**148**	**135**
(511, 103, 123)	164	160	164	179	179	-	160	**155**	**147**
(511, 76, 171)	176	176	176	195	184	-	171	**171**	**171**
(511, 58, 183)	183	184	185	207	199	-	183	**183**	**183**

TABLE 2. Comparasions of Some Heuristics For Finding the Minimum Distance of Linear Codes

5. CONCLUSION

In this article, two optimization algorithms, the A-ABC and MBA-ABC algorithms are presented to compute the minimum distance of binary linear codes. The proposed algorithms are based on the algebraic differential mutation operator. The proposed methods are applied to twenty BCH codes with known minimum distance values. It is clearly seen from the Table 2 that the MBA-ABC algorithm presents a better performance when compared to the A-ABC, GA-A, GA-B, GA, Hill Climbing, Tabu Search, Ant Colony and Metropolis Algorithms. Almost in all the papers in the literature, the codewords are used as a search space to calculate the minimum distance of a given code. But, in the presented algorithms, generator matrices are used as a search space, just like in [1, 12]. It is illustrated that the performance of the MBA-ABC algorithm is effected positively by two factors: the use of generator matrices instead of codewords as a search space and the use of the algebraic differential mutation operator in place of the classical operator of the A-ABC algorithm.

ACKNOWLEDGEMENT

The authors thank to Tarsus University for providing workstations with high computation performance used in the optimization of the minimum distance problem.

CONFLICTS OF INTEREST

The authors declare that there are no conflicts of interest regarding the publication of this article.

Authors Contribution Statement

All authors have contributed sufficiently to the planning, execution, or analysis of this study to be included as authors. All authors have read and agreed to the published version of the manuscript.

References

[1] Ajitha Shenoy, K.B., Biswas, S., Kurur, P.P., *Efficacy of the metropolis algorithm for the minimum-weight codeword problem using codeword and generator search spaces*, IEEE Transactions on Evolutionary Computation, **24**(4)(2020), 664–678.

[2] Akdagli, A., Ustun, D., *Bandwidth enhancement of rectangular microstrip antenna with a rectangular slot by using a novel hybrid optimization method based on the ABC and DE algorithms*, Int J Numer Model., **31**(5)(2018).

[3] Akdagli, A., Ustun, D., *Design of bandnotched UWB antenna using a hybrid optimization based on ABC and DE algorithms*, AEU - International Journal of Electronics and Communications, **87**(2018), 10–21.

[4] Askali, M., Azouaoui, A., Nouh, S., Belkasmi, M., *On the computing of the minimum distance of linear block codes by heuristic methods*, Int. J. Commun. Netw. Syst. Sci., **5**(2012), 774–84.

[5] Askali, M., Nouh, S., Belkasmi, M., *An efficient method to find the minimum distance of linear block codes*, in Proc. IEEE Int. Conf. Multimedia Comput. Signal Process, Tangier, Morocco, (2012), 185–188.

[6] Augot, D., Charpin, P., Sendrier, N., *Studying the locator polynomial of minimum weight codewords of BCH codes*, IEEE Trans. Info. Theory, *38*(1992), 960–973.

[7] Betten, A., Braun, M., Fripertinger, H., Kerber, A., Kohnert , A. et al., Error Correcting Linear Codes, Algorithms and Computation in Mathematics, 18, Springer, 2006.

[8] Bland, J.A., Baylis, A.T., *A tabu search approach to the minimum distance of error-correcting codes*, Int. J. Electron., **79**(6)(1995), 829–837.

[9] Bland, J.A., *Local search optimisation applied to the minimum distance problem*, Adv. Eng. Informat., **21**(2007), 391–397.

[10] Bosma, W., Cannon, J., Playoust, C., *The Magma algebra system I: The user language*, J. Symbolic Comput., **24**(1997), 235–265.

[11] Bouzkraoui, H. , Azouaoui, A., Hadi, Y., *New ant colony optimization for searching the minimum distance for linear codes*, Advanced Communication Technologies and Networking (CommNet), International Conference on. IEEE, (2018).

[12] Cuellar, M.P., Gomez-Torrecillas, J. , Lobillo, F.J., Navarro, G., *Genetic algorithms with permutation-based representation for computing the distance of linear codes*, Swarm and Evolutionary Computation, **60**(2021), 100797.

[13] Gomez-Torrecillas, J., Lobillo, F.J., Navarro, G., *Minimum distance computation of linear codes via genetic algorithms with permutation encoding*, ACM Communications in Computer Algebra, **52**(3)(2019).

[14] Hogben, L., Handbook of Linear Algebra, Boca Raton, FL, USA, Champman and Hall, 2007.

[15] Houghten, S.K., Wallis, J.L., *A comparative study of search techniques applied to the minimum distance problem of BCH codes*, in Proc, 6th IASTED Int. Conf. Artif. Intell. Soft Comput., (2002), 164–169.

[16] Joundan, I., Nouh, S., Azouazi, M., Namir, A., *A new efficient way based on special stabilizer multiplier permutations to attack the hardness of the minimum weight search problem for large BCH codes*, Int. J. Electr. Comput.Eng., **9**(2019), 1232.

[17] Karaboga, D., *An idea based on honey bee swarm for numerical optimization*, Technical Report-TR06, Erciyes University, Kayseri, Turkey, 2005.

[18] Karaboga, D., Basturk, B., *A powerful and efficient algorithm for numerical function optimization: Artificial bee colony (ABC) algorithm*, J. Glob. Optim., **39**(2007), 459–471.

[19] Karaboga D., Basturk, B., *On the performance of artificial bee colony (ABC) algorithm,* Applied Soft Computing, **8**(2008), 687–697.

[20] Lisonek, P., Trummer, L., *Algorithms for the minimum weight of linear codes*, Adv. Math. Commun., **10**(2016), 195–207.

[21] Ling, S., Xing, C., Coding Theory: A First Course, United Kingdom, Cambridge University Press, 2004.

[22] MacWilliams, F.J., Sloane, N.J.A., The Theory of Error-Correcting Codes, Amsterdam, North-Holland, 1993.

[23] Santucci, V., Baioletti, M., Milani, A., *Algebraic differential evolution algorithm for the permutation flowshop scheduling problem with total flowtime criterion*, IEEE Transactions on Evolutionary Computation, **20**(5)(2016), 682–694.

[24] Storn, R., Price, K., *Differential evolution a simple and efficient heuristic for global optimization over continuous spaces*, Journal of Global Optimization, **11**(1997), 341–359.

[25] Toktas, A., Ustun, D., *A triple-objective optimization scheme using butterfly-integrated ABC algorithm for design of multi-layer RAM*, IEEE Transactions on Antennas and Propagation, **68**(7)(2020), 5602–5612.

[26] Toktas, A., Ustun, D., Erdogan, N., *Pioneer Pareto artificial bee colony algorithm for three-dimensional objective space optimization of composite-based layered radar absorber*, Applied Soft Computing, **96**(2020), 1–11.

[27] Ustun, D., *An enhanced adaptive butterfly optimization algorithm rigorously verified on engineering problems and implemented to ISAR image motion compensation*, Engineering Computations, **37**(9)(2020), 3543–3566.

[28] Vardy, A., *The intractability of computing the minimum distance of a code*, IEEE Transactions on Information Theory, **43**(6)(1997), 1757–1766.