Journal of Soft Computing and Information Technology (JSCIT)
Babol Noshirvani University of Technology, Babol, Iran
Journal Homepage: www.jscit.nit.ac.ir
Volume 9, Number 1, Spring 2020, pp. 28-35
Received: 12/01/2018; Revised: 01/31/2020; Accepted: 02/13/2020

# EGA: An Enhanced Genetic Algorithm for Numerical Functions Optimization

Kiumars Ghazipour[1], Asadollah Shahbahrami[2*]
1-Department of Computer Engineering, Faculty of Engineering, University of Guilan, Rasht, Iran.
2-Department of Computer Engineering, Faculty of Engineering, University of Guilan, Rasht, Iran.
[1] ghazipour.k@gmail.com, [2*] shahbahrami@guilan.ac.ir

Corresponding author's address: Asadollah Shahbahrami, Department of Computer Engineering, Faculty of Engineering, University of Guilan, Rasht, Iran.

**Abstract-** Optimization is the process of making something as good or effective as possible. Optimization problems are used over many fields such as economics, science, industry and engineering. The growing use of optimization makes it essential for researchers in every branch of science and technology. To solve optimization problems many algorithms have been introduced, while achieving a higher quality of results in terms of accuracy and robustness is still an issue. Metaheuristics are widely recognized as efficient approaches for many hard optimization problems. In this study, to achieve a higher quality of results in numerical functions optimization, two new operators named N-digit lock search (NLS) and Two-Math crossover are introduced to enhance the genetic algorithm (GA) as a widely used metaheuristic. The NLS operator is inspired by the N-digit combination lock pattern and enhances the exploitative behavior of the GA by calibrating the current best solution and the relatively new Two-Math crossover operator combines both two-point and arithmetic crossover techniques to guide the overall search process better. The proposed enhanced genetic algorithm (EGA) is tested over 33 benchmark mathematical functions and the results are compared to some population-based, particle swarm optimization (PSO2011) and artificial bee colony (ABC) algorithms, and single-solution based, simulated annealing (SA), pattern search (PS), and vortex search (VS). A problem-based test is performed to compare the performance of the algorithms, which results shows the proposed EGA outperforms all other algorithms, SA, PS, VS, PSO2011 and ABC. In addition, it surprisingly finds the global best points for almost all 33 test functions with a constant value for 2 out of 3 EGA operators.

**Keywords**- Metaheuristics, Genetic algorithm, Function optimization, Global optimization.

## I. INTRODUCTION

Hard optimization problems are problems that cannot be solved to optimality, or any guaranteed bound, by any exact, deterministic method within a ''reasonable'' time limit. These problems can be divided into several categories. To find acceptable solutions for these types of problems, we can use metaheuristics. A metaheuristic is an algorithm designed to solve approximately a wide range of hard optimization problems without having to adapt to each problem deeply [1].

Metaheuristics are mainly classified and studied under two major categories, single-solution based and population-based [1, 2]. Single-solution based metaheuristics, are based on a single solution and include local search-based metaheuristics such as simulated annealing (SA)[3, 4], pattern search (PS) [5], and random search (RS) [6]. In population-based metaheuristics, first, a number of solutions are created and then updated continuously until the termination condition is met. Population-based metaheuristics are generally studied under two major groups: evolutionary algorithms and swarm-based algorithms. Fig. 1 depicts a visual categorization of these metaheuristics.

Although many metaheuristic algorithms have been introduced so far, works to provide a metaheuristic algorithm to get higher accuracy of results for the problems in hand continues in the research field [7-9], and achieving higher accuracy of results is still an issue.

Among other metaheuristics, a genetic algorithm (GA) [10]

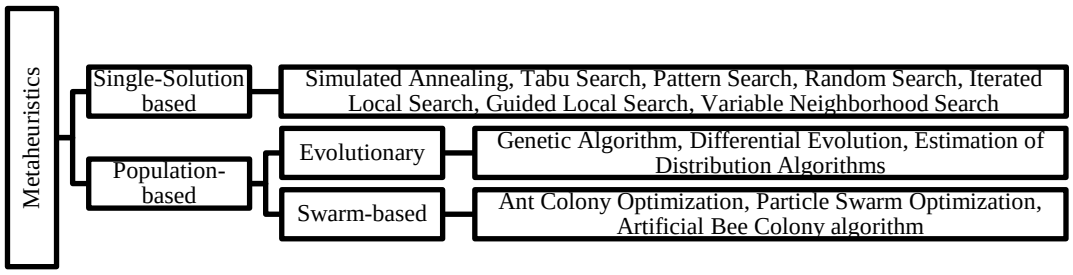


**Fig. 1. Visual categorization of metaheuristics mentioned in this paper.**

is arguably the most well-known and mostly used metaheuristic. The power of GAs comes from the fact that the technique is robust and can deal successfully with a wide range of problem areas including those which are difficult for other methods to solve. However, in some hard to optimize problems, GA usually finds near-global optimum but after some generations cannot offer further improvement even if a large number of generations are produced. Finally, at the end of the execution of GA, there is no guaranty to find the global best or at least the best possible solution using the currently found ones.

In this study we propose an enhanced genetic algorithm (EGA) which utilizes two new operators, namely, N-digit lock search (NLS), and a relatively new hybrid crossover operator named Two-Math crossover to solve bound-constrained global optimization problems. Here, the proposed NLS and Two-Math operators are applied to the standard continuous genetic algorithm as operators and results show that the proposed EGA considerably improves the performance of the search process and finds almost all the global best points for test functions. The proposed EGA is tested over the 33 well-known benchmark functions, and the results were compared to SA, PS, PSO2011[11], ABC[12], and VS[13] algorithms, where EGA was found to outperform all these algorithms.

The remaining part of the paper is organized as follows. Section 2 demonstrates the proposed EGA algorithm. Section 3 covers the experimental results and discussion. Finally, section 4 concludes the work.

## II. THE PROPOSED EGA

The proposed EGA is comprised of GA and two new operators introduced in the present work, namely the NLS and the Two-Math crossover. After initializing the population, using the roulette wheel selection method, the fittest parents will be selected to undergo the mating process. The mating process takes place by applying Two-Math crossover to form two high-quality offspring. The Two-Math crossover consists of two phases: first, by spreading good gens using two-point crossover, and second, by applying the arithmetic crossover with the probability of $\rho_{\text{arithmetic}}$ to the same offspring to adjust them to diverse the population and find new promising solutions in the continuous search space. Afterward, some offspring will undergo the mutation process with the probability of $p_m$. Then, the NLS operator works on the best solution found so far "Sbest" and tries to tune it to reach the best fitness available for Sbest. Finally, the tuned Sbest is inserted E time into the new population to fulfill elitism. E is a parameter into adjusting elitism which could be set from 1 to N where N is the population size. We had determined the value of E by parameter tuning to find the best degree of elitism that works best by EGA. Our elitism strategy takes place by inserting Sbest E times into the population, instead of selecting and inserting E best individuals. Using high values for E leads the population to stuck in a local optimum and/or behave randomly. Assuming E=1 is a typical elitism strategy that is commonly used by elitism-based GAs. We found E=3 a compromise point where both faster convergence and local optima avoidance are obtained. In the middle stages of EGA, thanks to the roulette wheel selection method, the population will be full of the fittest solutions found so far, which are likely very similar to the Sbest and so, faster convergence to either local or global optimum point is achieved. Here if Sbest is global optima, it is done, and if the Sbest is local optima, then because of population-based approach and variation operators (mostly the mutation), new variants of Sbest will be generated. Resulted in Sbest variations (after applying the NLS operator) are either better or worse than Sbest itself. Here, by inserting better Sbest variations into the population and repeating the evolution process, the total solutions will be guided towards the global optimum point. Fig. 2. describes the proposed EGA.

### *A. Methodology*

#### *1) Initial population and chromosome representation*

Given an initial population of N chromosomes, the initial values for each gen is generated by uniformly distributed random real values as below:

$$x_i(k) \leftarrow U[x_{min}(k) \dots x_{max}(k)] \,. \quad (1)$$

$$for\ i \in [1 \dots N] and\ k \in [1 \dots n] \,.$$

Where $x$ is a vector of individuals of size N; U is the uniform random real number generator, and $x_{min}$ and $x_{max}$ are lower and upper bounds for generated numbers. In the present study, the real coded representation method is adopted, in which each continuous variable holds a real number [14].

#### *2) Elitist strategy and selection method*

In the proposed EGA, a new elitism strategy named elitism of degree E is used, where E is the number of times

```
Parents ← {randomly generated population}
While (maximum number of iterations is not reached)
  Calculate the fitness of each parent in the population
  Elite ← Best parent
  Children ← 0
  While | Children | < | Parents |
    Use fitnesses to probabilistically select a pair of parents for mating using roulette wheel selection method
    Mate the parents to create children c1 and c2 using Two-Math crossover
    Children ← Children U {c1, c2}
  Loop
  Randomly mutate some of the children
  Apply NLS to the Elite individual
  Parents ← Children U Elite E Times  //insert elite individual (Sbest) E times into the population
  Parents ← Best N Parents
Next generation
```

**Fig. 2. The proposed EGA pseudo code**

the elite individual "Sbest" is inserted into the new population in each iteration. Inserting the E elite individuals into the new population takes place by replacing them with E random individuals which already exist in the population. For the selection process we adopted the roulette wheel method [15, 16] since the roulette wheel can ensure a faster convergence to either local or global optimum point.

*3) Two-Math crossover method*

In the present work, a relatively new hybrid crossover named Two-Math is introduced by combining the two-point and arithmetic crossover methods. These methods are utilized as a single crossover operator. In this Two-Math crossover as it can be understood from the name, we first apply two-point crossover to spread good gens through chromosomes and produce two offspring by the mating process. Then arithmetic crossover which is essentially designed to use in continuous GA, is applied to the same two produced offspring with the probability of $\rho_{arithmetic}$, where we found $\rho_{arithmetic} = 0.01$ by parameter tuning. Fig. 3. describes the Two-Math crossover method.

```
//Do two-point crossover
Mate the parents p1, p2 to create children c1 and c2 using
two-point crossover
//Do arithmetic crossover with probability of ρarithmetic
ρarithmetic ← 0.01
if (RandomDoubleBetween(0, 1) < ρarithmetic){
  α ← RandomDoubleBetween(0, 1);
  j ← random crossover point //Set arithmetic crossover point
  for (; j < GenSize; j++){
    c1.Gen[j] ← (p1.Gen [j] + p2.Gen [j]) * α;
    c2. Gen[j] ← (p1.Gen [j] + p2.Gen [j]) * (1 - α);}
}
```

**Fig. 3. Two-Math crossover pseudo code**

*4) Mutation method*

In the present work we used a uniform mutation [17]. The formulation can be written as:

$$r \leftarrow U[0..1]$$

$$x_i(k) \leftarrow \begin{cases} x_i(k). & if\ r \geq \rho \\ U[x_{min}(k).\ x_{max}(k)]. & if\ r < \rho \end{cases} \quad (2)$$

$$for\ i \in [1..N] and\ k \in [1..n].$$

*5) Stopping criteria of EGA*

The EGA is terminated after a 500000 prespecified maximum number of generations. This number of iterations is also applied in other algorithms which are used for performance comparison.

*6) The NLS operator*

The NLS operator is inspired by the N-digit combinational lock pattern and is used to improve the exploitative behavior of GA. Fig. 4. demonstrates a typical 10-digit combinational lock. In decimal representation, every digit of any number consists of digits ranging from 0 to 9. Considering the optimization of the continuous function, the same rule applies to the real numbers which are used as the value of the optimization functions variables. Like a combinational lock, in NLS the goal is to find the best digits for an N-digit solution of a numerical optimization function. In NLS, instead of the brute force search that happens while guessing the combinational lock password, a priority numerical search is used. It begins from calibrating the most significant digit of Sbest to less significant one using a vector of fixed values of length N that we call it the calibration vector (CV). The search process keeps calibrating Sbest in a way that fitness value for Sbest continually improves, until no improvement is available to Sbest. Fig. 5. Depicts a sample CV of length 10. The CV initialization could be varied according to the type of target problems and the amount of precision required. The NLS operator starts with Sbest, then attempts to find a better solution by making either an incremental or decremental change to the Sbest. Changes are made by means of adding or subtracting the whole CV values with Sbest starting from CV[0] to CV[N] in which better fitness is gained. If the change produced a better solution, another incremental or decremental change is made to the Sbest, and so on, until no further improvements can be achieved. As a result, rapid convergence to the best possible

**Fig. 4. A typical 10-digit combinational lock pad**

improvement available for Sbest is achieved. Fig. 6. shows the NLS pseudo code.

| No. | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| Value | 4.0 | 2.0 | 1.0 | 0.1 | 0.01 | 0.001 | 0.0001 | 0.00001 | 0.000001 | 0.0000001 |

**Fig. 5. Sample calibration vector of length 10**

**Inputs**: Initialized Calibration Vector "CV" (i.e. Fig. 5.)
Best solution found so far "$S_{best}$"
Chromosome Size "CS"
i = 0;
S = $S_{best}$
**Repeat**
Set *direction* (either to "+" or "-") in which improvement is available to the S
**Repeat**
Starting from CV[0] to CV[N], apply whole CV to S(i) either by adding or subtracting depending on the current *direction*
**While** (improvement is available for S(i)) //so fitness value for whole S will be improved
i++;
**While** i < CS
$S_{best}$ = S
**Output**: $S_{best}$ as the best solution achieved

**Fig. 6. The NLS operator pseudo code**

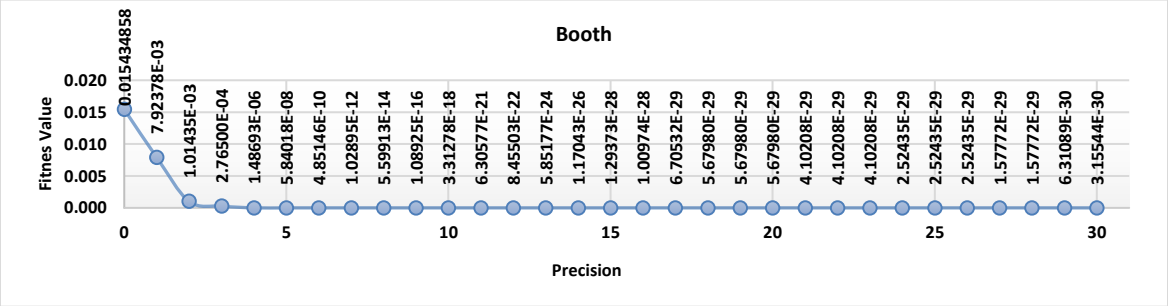


**Fig. 7. Running the EGA for max of 100 iterations over 2-domensional Booth function using CV length ranging from 0 to 30, which 0 means to not apply NLS operator.**

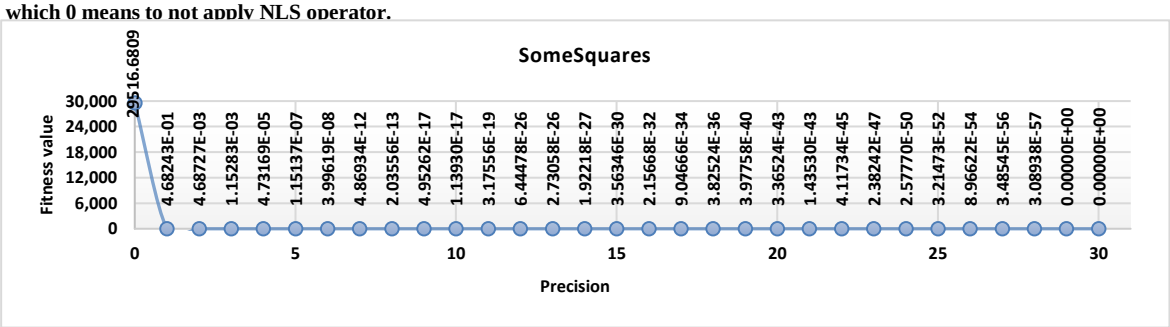


**Fig. 8. Running the EGA for max of 100 iterations over 30-domensional SumSquares function using CV length ranging from 0 to 30, which 0 means to not apply NLS operator.**

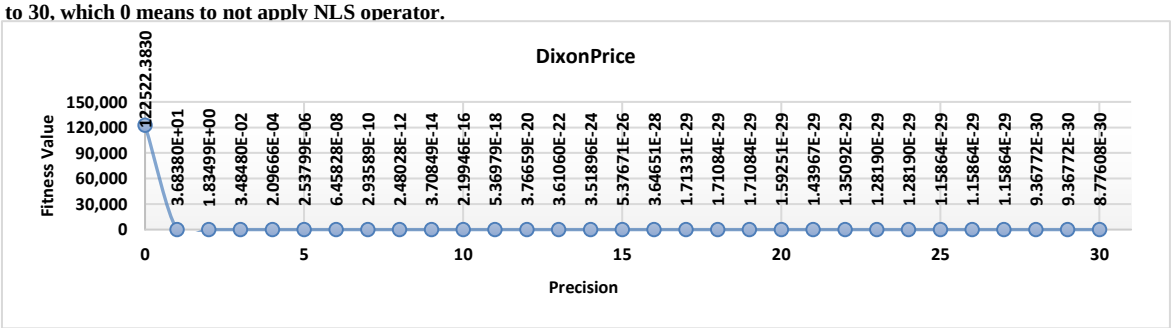


**Fig. 9. Running the EGA for max of 100 iterations over 30-domensional DixonPrice function using CV length ranging from 0 to 30, which 0 means to not apply NLS operator.**

Figs 7, 8, and 9 display fitness values for the 2-dimensional booth, and 30-dimensional SumSquares and DixonPrice test functions after 100 iterations respectively. Precision values represent the CV length (ranging from 0 to 30) utilized by the NLS operator. Since the precision value of "0", means running the simple GA without using the NLS operator, dramatic improvement can be seen using the CV length of only "1" over the max of 100 iterations. Furthermore, using the same number of iterations, results from applying CV of length 30 is outstanding, so that Booth, SumSquares and DixonPrice fitness values range from (0.015434858 to 3.15544E-30), (29516.6809 to 0.0) and (122522.3830 to 8.77608E-30) respectively.

**Table 1**
Statistical results of 30 runs obtained by SA, PS, PSO2011, ABC, VS and EGA algorithms (values < $10^{-16}$ are considered as 0). Where best values found by algorithms are global optima, they marked with "*". C: Characteristics, U: Unimodal, M: Multimodal, S: Separable, N: Non-Separable

| Test bed | C | Min. | | SA | PS | PSO2011 | ABC | VS | EGA |
|---|---|---|---|---|---|---|---|---|---|
| **Stepint** | **US** | **0** | Mean | 1.866666667 | 0 | 0 | 0 | 0 | 0 |
| | | | StdDev | 1.136641554 | 0 | 0 | 0 | 0 | 0 |
| | | | **Best** | **0*** | **0*** | **0*** | **0*** | **0*** | **0*** |
| **Step** | **US** | **0** | Mean | 0 | 0 | 0.066666667 | 0 | 0.2 | 0 |
| | | | StdDev | 0 | 0 | 0.253708132 | 0 | 0.406838102 | 0 |
| | | | **Best** | **0*** | **0*** | **0*** | **0*** | **0*** | **0*** |
| **Sphere** | **US** | **0** | Mean | 0 | 0 | 0 | 2.75098E-16 | 0 | 0 |
| | | | StdDev | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | **Best** | **0*** | **0*** | **0*** | 2.23487E-16 | **0*** | **0*** |
| **SumSquares** | **US** | **0** | Mean | 0 | 0 | 0 | 2.75098E-16 | 0 | 0 |
| | | | StdDev | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | **Best** | **0*** | **0*** | **0*** | 1.85594E-16 | **0*** | **0*** |
| **Quartic** | **US** | **0** | Mean | 0.4028326 | 0.049370406 | 1.64098E-05 | 0.013732963 | 0.000145026 | 7.38312E-05 |
| | | | StdDev | 0.301544881 | 0.046578461 | 5.56581E-06 | 0.002379448 | 7.30549E-05 | 1.069E-04 |
| | | | **Best** | 0.001414536 | 1.61333E-05 | 7.13993E-06 | 0.008413424 | 5.54996E-05 | **5.81378E-06** |
| **Beale** | **UN** | **0** | Mean | 0.000430475 | 0 | 0 | 6.37598E-16 | 0 | 0 |
| | | | StdDev | 0.000943865 | 0 | 0 | 3.58687E-16 | 0 | 0 |
| | | | **Best** | 2.51078E-08 | **0*** | **0*** | **0*** | **0*** | **0*** |
| **Easom** | **UN** | **-1** | Mean | -0.028827505 | -8.11022E-05 | -1 | -1 | -1 | -1 |
| | | | StdDev | 0.157894721 | 0 | 0 | 0 | 0 | 0 |
| | | | **Best** | -0.864825008 | -8.11022E-05 | **-1*** | **-1*** | **-1*** | **-1*** |
| **Matyas** | **UN** | **0** | Mean | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | StdDev | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | **Best** | **0*** | **0*** | **0*** | **0*** | **0*** | **0*** |
| **Colville** | **UN** | **0** | Mean | 1.83377047 | 0.002199995 | 0 | 0.00576453 | 0 | 0 |
| | | | StdDev | 2.351638954 | 0 | 0 | 0.003966867 | 0 | 0 |
| | | | **Best** | 0.000971812 | 0.002199995 | **0*** | 0.000383073 | **0*** | **0*** |
| **Trid6** | **UN** | **-50** | Mean | -49.84789091 | -50 | -50 | -50 | -50 | -50 |
| | | | StdDev | 0.150775917 | 0 | 3.61345E-14 | 4.94748E-14 | 2.96215E-14 | 1.67649E-12 |
| | | | **Best** | -49.98701123 | **-50*** | **-50*** | **-50*** | **-50*** | **-50*** |
| **Trid10** | **UN** | **-210** | Mean | 209.5023223 | -209.9954224 | -210 | -210 | -210 | -210 |
| | | | StdDev | 0.230476381 | 0 | 2.30778E-13 | 9.62204E-12 | 6.19774E-13 | 5.70687E-11 |
| | | | **Best** | -209.8801988 | -209.9954224 | **-210*** | **-210*** | **-210*** | **-210*** |
| **Zakharov** | **UN** | **0** | Mean | 0 | 0 | 0 | 7.56674E-14 | 0 | 0 |
| | | | StdDev | 0 | 0 | 0 | 3.76382E-14 | 0 | 0 |
| | | | **Best** | **0*** | **0*** | **0*** | 2.31887E-14 | **0*** | **0*** |
| **Powell** | **UN** | **0** | Mean | 0 | 0 | 2.04664E-07 | 9.09913E-05 | 1.43967E-05 | 0 |
| | | | StdDev | 0 | 0 | 1.21051E-08 | 1.42475E-05 | 2.27742E-06 | 0 |
| | | | **Best** | **0*** | **0*** | 1.72679E-07 | .23427E-05 | 5.71959E-06 | **0*** |
| **Schwefel 2.22** | **UN** | **0** | Mean | 0 | 0 | 1.094284383 | 8.51365E-16 | 0 | 0 |
| | | | StdDev | 0 | 0 | 0.870781136 | 0 | 0 | 0 |
| | | | **Best** | **0*** | **0*** | 0.107097937 | 8.51365E-16 | **0*** | **0*** |
| **Schwefel 1.2** | **UN** | **0** | Mean | 0 | 0 | 0 | 0.000760232 | 0 | 0 |
| | | | StdDev | 0 | 0 | 0 | 0.000440926 | 0 | 0 |
| | | | **Best** | **0*** | **0*** | **0*** | 0.00027179 | **0*** | **0*** |
| **Rosenbrock** | **UN** | **0** | Mean | 0.224618742 | 9.84185348 | 0.930212233 | 0.003535257 | 0.367860114 | 0.930212233 |
| | | | StdDev | 0.097171414 | 0 | 1.714978077 | 0.003314818 | 1.130879848 | 1.686152863 |
| | | | **Best** | 0.082077849 | 9.84185348 | **0*** | 7.08757E-05 | 9.42587E-05 | **0*** |
| **Dixon-Price** | **UN** | **0** | Mean | 0.990721802 | 0.666666667 | 0.666666667 | 1.91607E-15 | 0.666666667 | 0 |
| | | | StdDev | 0.029412712 | 0 | 4.38309E-16 | 2.55403E-16 | 7.68909E-16 | 0 |
| | | | **Best** | 0.871516993 | 0.666666667 | 0.666666667 | 1.1447E-15 | 0.666666667 | **0*** |
| **Foxholes** | **MS** | **0.998003838** | Mean | 5.5682975 | 0.998003838 | 34.26621987 | 0.998003933 | 0.998003838 | 0.998003838 |
| | | | StdDev | 4.367922182 | 4.51681E-16 | 126.6004794 | 4.33771E-07 | 0 | 0 |
| | | | **Best** | **0.998003838*** | **0.998003838*** | **0.998003838*** | **0.998003838*** | **0.998003838*** | **0.998003838*** |
| **Branin** | **MS** | **0.397887358** | Mean | 0.398269177 | 0.397887358 | 0.397887358 | 0.397887358 | 0.397887358 | 0.397887358 |
| | | | StdDev | 0.001624387 | 0 | 0 | 0 | 0 | 5.99931E-16 |
| | | | **Best** | 0.397887361 | **0.397887358*** | **0.397887358*** | **0.397887358*** | **0.397887358*** | **0.397887358*** |
| **Bohachevsky1** | **MS** | **0** | Mean | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | StdDev | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | **Best** | **0*** | **0*** | **0*** | **0*** | **0*** | **0*** |
| **Booth** | **MS** | **0** | Mean | 5.28496E-05 | 0 | 0 | 0 | 0 | 0 |
| | | | StdDev | 7.35674E-05 | 0 | 0 | 0 | 0 | 0 |
| | | | **Best** | 7.35674E-05 | **0*** | **0*** | **0*** | **0*** | **0*** |
| **Rastrigin** | **MS** | **0** | Mean | 0 | 0 | 26.11016129 | 0 | 57.60799224 | 0 |
| | | | StdDev | 0 | 0 | 5.686650032 | 0 | 13.94980276 | 0 |
| | | | **Best** | **0*** | **0*** | 16.91429893 | **0*** | 13.94980276 | **0*** |
| **Schwefel** | **MS** | **-12569.48662** | Mean | -1891.275468 | -3686.285205 | -8316.185447 | -12569.48662 | -11283.05416 | -12569.48662 |
| | | | StdDev | 137.3913021 | 2.77513E-12 | 463.9606712 | 1.85009E-12 | 352.1869262 | 1.81899E-12 |
| | | | **Best** | -2188.304761 | -3686.285205 | -9466.201047 | **-12569.48662*** | -11799.62928 | **-12569.48662*** |
| **Michalewicz2** | **MS** | **-1.80130341** | Mean | 1.792778285 | -1.80130341 | -1.80130341 | -1.80130341 | -1.80130341 | -1.80130341 |
| | | | StdDev | 0.043874926 | 1.35504E-15 | 9.03362E-16 | 9.03362E-16 | 9.03362E-16 | 6.66134E-16 |
| | | | **Best** | -1.801296643 | **-1.80130341*** | **-1.80130341*** | **-1.80130341*** | **-1.80130341*** | **-1.80130341*** |
| **Michalewicz5** | **MS** | **-4.687658179** | Mean | -3.670604734 | -4.495893207 | -4.67700874 | -4.687658179 | -4.670953055 | -4.681228926 |
| | | | StdDev | 0.496257736 | 2.71009E-15 | 0.036487971 | 2.60778E-15 | 0.020809276 | 0.034622588 |
| | | | **Best** | -4.684023442 | -4.495893207 | **-4.687658179*** | **-4.687658179*** | **-4.687658179*** | **-4.687658179*** |
| **Michalewicz10** | **MS** | **-9.660151716** | Mean | -6.060491565 | -8.461507306 | -9.204154798 | -9.660151716 | -8.793361668 | -9.660151716 |
| | | | StdDev | 0.504024688 | 5.42017E-15 | 0.298287637 | 0 | 0.382153549 | 4.91851E-15 |
| | | | **Best** | -6.880235805 | -8.461507306 | **-9.660151716*** | **-9.660151716*** | -9.410563187 | **-9.660151716*** |
| **Schaffer** | **MN** | **0** | Mean | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | StdDev | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | **Best** | **0*** | **0*** | **0*** | **0*** | **0*** | **0*** |
| **Six Hump Camel Back** | **MN** | **-1.031628453** | Mean | -1.031621639 | -1.031628453 | -1.031628453 | -1.031628453 | -1.031628453 | -1.031628453 |
| | | | StdDev | 2.1595E-05 | 4.51681E-16 | 6.71219E-16 | 6.77522E-16 | 6.77522E-16 | 0 |
| | | | **Best** | -1.031628448 | **-1.031628453*** | **-1.031628453*** | **-1.031628453*** | **-1.031628453*** | **-1.031628453*** |
| **Bohachevsky2** | **MN** | **0** | Mean | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | StdDev | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | **Best** | **0*** | **0*** | **0*** | **0*** | **0*** | **0*** |
| **Bohachevsky3** | **MN** | **0** | Mean | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | StdDev | 0 | 0 | 0 | 0 | 0 | 0 |
| | | | **Best** | **0*** | **0*** | **0*** | **0*** | **0*** | **0*** |
| **Shubert** | **MN** | **-186.7309088** | Mean | -186.7309087 | -123.5767709 | -186.7309088 | -186.7309088 | -186.7309088 | -186.7309088 |
| | | | StdDev | 5.76173E-07 | 0 | 4.49449E-13 | 1.18015E-14 | 1.18015E-14 | 2.84217E-14 |
| | | | **Best** | **-186.7309088*** | -123.5767709 | **-186.7309088*** | **-186.7309088*** | **-186.7309088*** | **-186.7309088*** |
| **GoldStein-Price** | **MN** | **3** | Mean | 3.000000254 | 30 | 3 | 3 | 3 | 3 |
| | | | StdDev | 4.36073E-07 | 1.08403E-14 | 1.22871E-15 | 1.7916E-15 | 1.44961E-15 | 2.61436E-14 |
| | | | **Best** | **3*** | 30 | **3*** | **3*** | **3*** | **3*** |
| **Kowalik** | **MN** | **0.000307486** | Mean | 0.002635099 | 0.00031966 | 0.000307486 | 0.000319345 | 0.000307486 | 0.000307486 |
| | | | StdDev | 0.001644496 | 0 | 0 | 5.4385E-06 | 0 | 0 |
| | | | **Best** | 0.000780214 | 0.00031966 | **0.000307486*** | 0.00030894 | **0.000307486*** | **0.000307486*** |

## III. EXPERIMENTAL RESULTS

The proposed EGA algorithm is tested over 33 benchmark functions, that are obtained from the study performed by Doğan and Ölmez [13]. They compared the performance of the VS algorithm to the SA, PS, PSO2011 and ABC algorithms. We have compared the performance of the proposed approach to the mentioned algorithms using the same benchmark. SA and PS are two well-known single-solution based algorithms, and PSO2011 is an extension of the standard PSO algorithm.

For evaluation, the overall performances of the algorithms are studied for a constant number of iterations. After a certain number of iterations, the algorithms are evaluated according to the mean and the best fitness values found for each benchmark function.

**Table 2**
Problem-based comparison of the proposed EGA algorithm.

| Problem type | EGA vs. SA | EGA vs. PS | EGA vs. PSO2011 | EGA vs. ABC | EGA vs. VS |
|---|---|---|---|---|---|
| US | 2/3/**0** | 1/4/**0** | 1/3/**1** | 3/2/**0** | 2/3/**0** |
| UN | 7/5/**0** | 5/7/**0** | 8/4/**0** | 8/4/**0** | 3/9/**0** |
| MS | 7/2/**0** | 3/6/**0** | 5/4/**0** | 0/9/**0** | 4/5/**0** |
| MN | 4/3/**0** | 3/4/**0** | 0/7/**0** | 1/6/**0** | 1/6/**0** |
| Total (+/=/-) | 20/13/**0** | 12/21/**0** | 14/18/**1** | 12/21/**0** | 10/23/**0** |

### A. Algorithm settings

Population-based metaheuristics (ABC, PSO2011, EGA) are selected to have a population size of 50. For the proposed EGA algorithm, crossover and CV parameters are set to the fixed value of 0.9 and 14 respectively, and mutation rates are set as (Kowalik: 1.0), (Quartic-Easom-Schwefel-Michalewicz5-Michalewicz10-Bohachevsky2-Bohachevsky3-Shubert: 0.8), (Rastrigin- Schaffer: 0.1), (the rest of functions: 0.5). The number of neighborhood solutions of the VS algorithm is set to 50. The SA algorithm always performs with a single solution, and the PS algorithm creates its own neighbor vectors (pattern). The acceleration coefficients (c1 and c2) of the PSO2011 algorithm are both set to 1.8, and the inertia coefficient is set to 0.6, as in [18]. The limit value for the ABC algorithm is determined as limit = SN / D, where SN represents the number of food sources and D represents the dimension. The maximum number of iterations is selected as 500,000 to evaluate the overall performances of the algorithms.

### B. Overall performances of the algorithms

The proposed EGA algorithm is compared to the SA, PS, PSO2011, ABC and VS algorithms, using the 33 benchmark functions given in Table 1. For each algorithm, 30 different runs are performed, and the mean and the best fitness values are recorded. The maximum number of iterations is selected to be 500,000, as mentioned previously. For the SA and PS algorithms, the MATLAB® Global Optimization Toolbox is used, and the other algorithms are also coded in MATLAB®, except EGA which is coded in C#.Net. For each algorithm, all of the functions are run in parallel using an Intel® Corei5 CPU 8 GB RAM workstation. Experimental results are presented in Table 1.

Most of the modern software development tools use an arithmetic precision of $10^{-16}$ in the double-precision mode. An arithmetic precision value that is higher than necessary, makes it difficult to compare the local search abilities of the algorithms [19]. For this purpose, during the pair-wise comparison, resulting values below $10^{-16}$ are considered as 0. From Table 1, it can be shown that the EGA algorithm outperforms the SA, PS, PSO2011, ABC and VS algorithms. The SA algorithm performs a pure random search over the search space for which obtained results become meaningful. The PS algorithm is also a single-solution based algorithm that performs poorly compared to the EGA algorithm. The ABC algorithm is a powerful swarm-based algorithm that is used successfully for the solution of many types of optimization problems. For a number of functions, the ABC algorithm fails to exceed the $10^{-16}$ limit. Once the algorithm converges to a near-optimal point, the excellent local search ability of the EGA algorithm can be seen, which helps the algorithm to further improve the solution.

In Table 2, a problem-based (US, UN, MS and MN) comparison of the algorithms is also provided. Each cell in the Table 2 shows the total count of the three cases (+ ➔ Win / = ➔ Equal / - ➔ Loss). From Table 2, it can be shown that, for MN (multimodal non-separable) functions, the proposed EGA algorithm performs better than the other algorithms. For UN (unimodal non-separable) functions the proposed EGA algorithm again performs better than the others. Thus, from these results it can be inferred that the EGA algorithm performs better than other algorithms. For MS (multimodal separable) functions the proposed EGA algorithm again outperforms the other algorithms except for the ABC algorithm. Finally, for US (unimodal separable) functions the proposed EGA algorithm performs completely better than other algorithms while being competitive with the PSO2011 algorithm.

### C. Analyzing the effect of proposed operators

**Table 3**
Comparing the effect of each proposed operator independently. For an accurate comparison, exact values are displayed.
GA1: GA with two-point crossover.
GA2: GA with proposed Two-Math crossover.
GA3: GA with two-point crossover and proposed NLS operator.
EGA: Proposed algorithm.

| Test bed | Min. | | GA1 | GA2 | GA3 | EGA |
|---|---|---|---|---|---|---|
| Beale | 0 | Mean | 1.25927E-07 | 1.00663E-07 | 5.15487E-25 | 4.79682E-25 |
| | | StdDev | 8.28221E-08 | 5.27296E-08 | 1.53119E-25 | 1.37906E-25 |
| | | Best | 1.52144E-08 | 3.08371E-08 | 2.68522E-25 | 2.21837E-25* |
| Booth | 0 | Mean | 4.33821E-09 | 2.86155E-09 | 2.97156E-26 | 7.42323E-27 |
| | | StdDev | 5.51052E-09 | 3.61045E-09 | 2.32237E-26 | 1.91435E-26 |
| | | Best | 1.96679E-11 | 5.35202E-11 | 5.50625E-27 | 7.88861E-31* |
| Colville | 0 | Mean | 1.89115E-01 | 2.13100E-01 | 4.17146E-22 | 3.74920E-22 |
| | | StdDev | 0.155654696 | 0.167983809 | 1.82244E-22 | 1.81417E-22 |
| | | Best | 1.56804E-03 | 2.30417E-04 | 1.24546E-22 | 8.79868E-23* |
| Kowalik | 0.000307486 | Mean | 0.000593268 | 0.000485504 | 0.000307486 | 0.000307486 |
| | | StdDev | 0.000103839 | 9.93076E-05 | 6.64809E-17 | 4.91747E-17 |
| | | Best | 0.000406713 | 0.000320809 | 0.000307486 | 0.000307486* |

| | | | | | |
|---|---|---|---|---|---|
| Rosenbrock | 0 | Mean | 1852863.9560 | 28.712710919 | 1.461762080 | 0.930212233 |
| | | StdDev | 811776.2431 | 0.527735046 | 1.921131749 | 1.686152863 |
| | | Best | 410660.3884 | 2.72733E+01 | 8.04386E-22 | 6.27580E-22* |

**Table 4**
Comparing the advantages and disadvantages of algorithms used in the current work.
N= Number of iterations, P= Population size, CS: Chromosome Size, CVL= Calibration Vector Length
SB= Single Solution-Based, PB= Population-Based

| | SA | PS | VS | PSO2011 | ABC | EGA |
|---|---|---|---|---|---|---|
| Accuracy | 53% | 64% | 79% | 82% | 67% | 97% |
| Computational Complexity | O(N) | O(N) | O(N) | O(N*P) | O(N*P) | O(N*CS*CVL) |
| Computational approach | SB | SB | SB | PB | PB | PB |

In Table 3, we analyzed the effect of each operator introduced in our work independently and compared them with standard GA and proposed EGA to measure improvements obtained by each operator. To achieve this, different versions of GA have run on five test-beds for a maximum of 500,000 iterations. According to Table 3, improvements obtained using proposed operators are clear, and they become clearer, when the hardness of test function grows higher, like Rosenbrock function. Assuming GA1 as a base algorithm, GA2 performs better (Kowalik (GA1: 0.000406713 ➔ GA2: 0.000320809), Rosenbrock (GA1: 410660.3884 ➔ GA2: 2.72733E+01)) or at least competitive (Beale (GA1: 1.52144E-08 ➔ GA2: 3.08371E-08), Booth (GA1: 1.96679E-11 ➔ GA2: 5.35202E-11), Colville (GA1: 1.56804E-03 ➔ GA2: 2.30417E-04)) with GA1 which indicates better performance of proposed Two-Math crossover in comparison with two-point crossover. GA3 is far better than both GA1 & GA2 by holding a large edge in all test cases (i.e. Colville (GA2: 2.30417E-04 ➔ GA3: 1.24546E-22), Rosenbrock (GA2: 2.72733E+01 ➔ GA3: 8.04386E-22)) and thus, the impressive performance of NLS operator can be seen. Finally, the last column displays results obtained by the proposed EGA, where both Two-Math and NLS operators are used. As results show in Table 3, EGA outperforms other GA versions used in the experiment. Comparing EGA with GA3, again the performance of proposed Two-Math crossover is clear where slightly better results obtained by EGA (Booth (GA3: 5.50625E-27 ➔ EGA: 7.88861E-31), Colville (GA3: 1.24546E-22 ➔ EGA: 8.79868E-23)).

### *D. Summarized comparison of discussed algorithms*

Table 4 compares algorithms used in the current work in terms of accuracy, computational complexity and computational approach, while they share nearly similar implementation difficulty. Since metaheuristics are stochastic algorithms and meanwhile, the most time-consuming part of a metaheuristic is the fitness function evaluation phase[20]. Therefore, the number of fitness function evaluations is usually used as a performance criterion in metaheuristic algorithms. Considering the fitness function evaluation metric, the lower computational complexity of PS, SA, and VS is because of their single-solution based approach, where fewer function evaluations are required. PSO2011, ABC and EGA algorithms are population-based approaches and their higher computational complexity is meaningful. In Table 4, based on results provided in Table 1, the accuracy of algorithms is represented by a fraction of times the algorithm successfully reached the global best point for benchmark functions where formulation can be written as:

$$accuracy = \frac{total\ count\ of\ times\ the\ algorithm\ found\ global\ best}{total\ number\ of\ test\ functions} \quad (3)$$

Using the formula (3), fractions for algorithms are as SA($\frac{17}{33}$), PS($\frac{21}{33}$), VS($\frac{26}{33}$), PSO2011($\frac{27}{33}$), ABC($\frac{22}{33}$), EGA($\frac{32}{33}$). As can be understood from Table 4, the proposed EGA algorithm noticeably has the highest accuracy of 97% among the other algorithms in cost of higher computational complexity, while the second-best algorithm is PSO2011 with an accuracy of 82% which has considerable difference compared with EGA.

## IV. CONCLUSIONS

This paper introduced the EGA that utilizes two new operators introduced in this paper, namely the NLS and Two-Math, which is and may be used for other bound-constraint numerical function optimization problems. The NLS operator utilizes a CV of length N that helps to rapidly find the best possible state and accurate optimum point for the existing individuals and thus effectively guides the overall search process toward the global optimum point. The NLS operator is quite simple and does not require any additional parameters and has rapid convergence behavior. Furthermore, the Two-Math crossover operator is used to effectively explore the continuous search space by first spreading good gens through produced offspring and then applying the mathematical crossover technique with the probability of $\rho_{arithmetic}$, to the same produced offspring. The EGA algorithm is tested over a large set of 33 benchmark functions that comprise unimodal, multimodal, separable and non-separable problems of different dimensions. The results are compared to both single-solution based metaheuristics (SA, PS and VS) and population-based metaheuristics (PSO2011, ABC and GA); the results revealed that besides its simplicity, the proposed EGA algorithm is also highly competitive when compared to the performance of the other algorithms. The proposed algorithm is quite simple and uses fixed values for crossover rate and CV parameters as represented in algorithm settings. Accuracy of the proposed EGA algorithm makes it the right candidate for the solution of real-life optimization problems. In future studies, the proposed EGA algorithm will be improved to handle constraint optimization problems. The

EGA algorithm will also be applied to some real-life optimizations including engineering optimization problems.

## V. REFERENCES


[1] N. Razmjooy, V. V. Estrela, H. J. Loschi, and W. Fanfan, "A comprehensive survey of new meta-heuristic algorithms," Recent Advances in Hybrid Metaheuristics for Data Clustering, Wiley Publishing, 2019.

[2] J. Silberholz, B. Golden, S. Gupta, and X. Wang, "Computational Comparison of Metaheuristics," in Handbook of Metaheuristics, Springer, 2019, pp. 581-604.

[3] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," Science, vol. 220, no. 4598, pp. 671-680, 1983.

[4] W. Zhang, A. Maleki, M. A. Rosen, and J. Liu, "Optimization with a simulated annealing algorithm of a hybrid system for renewable energy including battery and hydrogen storage," Energy, vol. 163, pp. 191-207, 2018.

[5] R. Hooke and T. A. Jeeves, "Direct Search Solution of Numerical and Statistical Problems," Journal of the ACM (JACM), vol. 8, pp. 212-229, 1961.

[6] L. Rastrigin, "The convergence of the random search method in the extremal control of a many parameter system," Automaton & Remote Control, vol. 24, pp. 1337-1342, 1963.

[7] M. M. Motevali, A. M. Shanghooshabad, R. Z. Aram, and H. Keshavarz, "WHO: A New Evolutionary Algorithm Bio-Inspired by Wildebeests with a Case Study on Bank Customer Segmentation," International Journal of Pattern Recognition and Artificial Intelligence, vol. 33, p. 1959017, 2019.

[8] S. Harifi, M. Khalilian, J. Mohammadzadeh, and S. Ebrahimnejad, "Emperor Penguins Colony: a new metaheuristic algorithm for optimization," Evolutionary Intelligence, vol. 12, pp. 211-226, 2019.

[9] P. Pijarski and P. Kacejko, "A new metaheuristic optimization method: the algorithm of the innovative gunner (AIG)," Journal of Engineering Optimization, vol. 51, no. 12, pp. 1-20, 2019.

[10] S. Mirjalili, "Genetic algorithm," in Evolutionary Algorithms and Neural Networks Theory and Applications, ed: Springer, 2019, pp. 43-55.

[11] S. D. Mohanty, T. Anderson, M. Birattari, M. Dorigo, S. Boixo, T. F. Rønnow, et al., "Standard particle swarm optimisation 2011 at CEC-2013: A baseline for future PSO improvements," in Swarm Intelligence Methods for Statistical Regression. vol. 1, ed: IEEE Wiley, 2019, pp. xi-xiii.

[12] D. Karaboga and B. Basturk, "Artificial bee colony (ABC) optimization algorithm for solving constrained optimization problems," in International fuzzy systems association world congress, 2007, pp. 789-798.

[13] B. Doğan and T. Ölmez, "A new metaheuristic for numerical function optimization: Vortex Search algorithm," Information Sciences, vol. 293, pp. 125-145, 2015.

[14] F. Herrera, M. Lozano, and J. L. Verdegay, "Tackling real-coded genetic algorithms: Operators and tools for behavioural analysis," Artificial intelligence review, vol. 12, pp. 265-319, 1998.

[15] D. E. Goldberg and K. Deb, "A comparative analysis of selection schemes used in genetic algorithms," in Foundations of genetic algorithms. vol. 1, ed: Elsevier, 1991, pp. 69-93.

[16] J. Zhong, X. Hu, J. Zhang, and M. Gu, "Comparison of performance between different selection strategies on simple genetic algorithms," in International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce, pp. 1115-1121, 2005.

[17] T. Bäck, D. B. Fogel, and Z. Michalewicz, Evolutionary computation 1: Basic algorithms and operators: CRC press, 2018.

[18] D. Karaboga and B. Akay, "A comparative study of artificial bee colony algorithm," Applied mathematics and computation, vol. 214, pp. 108-132, 2009.

[19] P. Civicioglu, "Backtracking search optimization algorithm for numerical optimization problems," Applied Mathematics and Computation, vol. 219, pp. 8121-8144, 2013.

[20] A. Cano, A. Zafra, and S. Ventura, "Speeding up the evaluation phase of GP classification algorithms on GPUs," Soft Computing, vol. 16, pp. 187-202, 2012.

مجله علمی پژوهشی رایانش نرم و فناوری اطلاعات
دانشگاه صنعتی نوشیروانی بابل
صفحه مجله: jscit.nit.ac.ir
جلد ۹، شماره ۱، بهار ۱۳۹۹، صفحه ۲۸-۳۵
دریافت: ۱۳۹۷/۰۹/۱۰، بازنگری: ۱۳۹۸/۱۱/۱۱، پذیرش: ۱۳۹۸/۱۱/۲۴

# ارائه یک الگوریتم ژنتیک بهبود یافته برای بهینه سازی توابع عددی

کیومرث قاضی پور[۱]، اسدالله شاه بهرامی *[۲]
۱- گروه مهندسی کامپیوتر، دانشکده فنی و مهندسی دانشگاه گیلان، رشت، ایران.
*۲- گروه مهندسی کامپیوتر، دانشکده فنی و مهندسی دانشگاه گیلان، رشت، ایران.
[1]ghazipour.k@gmail.com, [2*]shahbahrami@guilan.ac.ir

* نشانی نویسنده مسئول: اسدالله شاه بهرامی، رشت، دانشگاه گیلان، دانشکده فنی و مهندسی، گروه مهندسی کامپیوتر.

**چکیده: بهینه‌سازی فرآیندی است که مسائل را تا آنجائیکه امکان داشته باشد، به بهترین وجه ممکن کارآمدی در می‌آورد. مسائل بهینه‌سازی در زمینه‌های مختلفی از قبیل اقتصاد، علوم، صنایع و مهندسی مورد استفاده قرار می‌گیرد. رشد استفاده از الگوریتم‌های بهینه‌سازی بحدی رسیده است که پژوهشگران در زمینه‌های مختلف دیگر از جمله فناوری خواهان استفاده از این الگوریتم‌ها نیز هستند. برای بهینه‌سازی مسائل، الگوریتم‌های زیادی معرفی شده‌اند ولی بدست آوردن نتایج با کیفیت و دقت بالا هنوز از چالشهای پیش رو است. الگوریتم‌های فراابتکاری در مسائل بهینه‌سازی سخت، شناخته شده هستند. در این مقاله برای بهبود کارایی الگوریتم ژنتیک (GA) بعنوان یک الگوریتم فراابتکاری، برای بدست آوردن نتایجی با کیفیت بالاتر در بهینه‌سازی توابع عددی دو عملگر جدید بنامهای جستجوی قفل N رقمی (NLS) و تلاقی دو نقطه حسابی ارائه شده است. عملگر NLS از الگوی قفل ترکیبی N رقمی الهام گرفته شده است که رفتار استثماری الگوریتم ژنتیک را با تنظیم بهترین راه حل جاری بهبود می‌دهد و عملگر دوم با تکنیکهای تلاقی دو نقطه و حسابی در جهت هدایت جستجوی کلی بهتر، ترکیب و پردازش می کند. الگوریتم ژنتیک بهبود داده شده (EGA) روی ۳۳ تابع ریاضی بعنوان توابع محک‌زن مورد تست و ارزیابی قرار گرفت. نتایج بدست آمده با برخی از الگوریتم های مبتنی بر جمعیت مانند بهینه‌سازی ازدحام ذرات و الگوریتم کلونی زنبور عسل (ABC) و الگوریتم های مبتنی بر یک جواب مانند الگوریتم شبیه‌سازی تبرید (SA) و جستجوی الگو (PS) و جستجوی گردابی (VA) مورد مقایسه قرار گرفت. نتایج نشان داد که کارایی عملکرد الگوریتم EGA نسبت به بقیه الگوریتم‌های مورد مقایسه بهتر است. بعلاوه نتایج نشان داد که الگوریتم EGA بهترین نقاط بهینه سراسری را تقریبا برای تمامی ۳۳ تابع مورد تست با استفاده از مقادیر ثابت برای دو عملگر از سه عملگر موجود پیدا می کند.**

**واژه‌های کلیدی:** الگوریتم‌های فراابتکاری، الگوریتم ژنتیک، بهینه‌سازی توابع، بهینه‌سازی سراسری.