STATISTICS, OPTIMIZATION AND INFORMATION COMPUTING
Stat., Optim. Inf. Comput., Vol. **12**, March 2024, pp 446–462.
Published online in International Academic Press (www.IAPress.org)

An Enhanced Genetic Algorithm Using Directional-Based Crossover and Normal Mutation for Global Optimization Problems

Ahmed M. Abdelkhalek [1,*], Ammar Mohammed[2], Mahmoud A. Attia [3], Niveen Badra[1]

[1]*Department of Physics and Engineering Mathematics, Faculty of Engineering, Ain Shams University, Cairo, Egypt*
[2]*Department of Computer Science, Faculty of Graduate Studies for Statistical Research, Cairo University, Cairo, Egypt*
[3]*Department of Electrical Power and Mechanics, Faculty of Engineering, Ain shams University, Cairo, Egypt*

Abstract Global optimization has been employed in many practical modeling processes. Using gradient methods to solve optimization problems may be computationally inefficient and time-consuming, particularly when convexity or differentiability is not guaranteed. On the other hand, nature-inspired techniques offer an effective gradient-free approach for solving complex, non-convex, or non-differentiable problems. Genetic algorithms are one of the most effective and widely used nature-inspired techniques. However, canonical genetic algorithms do not always guarantee convergence to the optimum point owing to the stochastic nature of the genetic operators, and typically require more work to ensure convergence and increase performance. Improving the genetic operators remains an open issue and usually involves a trade-off between the speed of convergence and searchability. In this study, we propose an enhanced genetic algorithm that relies on directional-based crossover and normal mutation operators to increase the speed of convergence while preserving searchability. The proposed algorithm is evaluated using a set of 40 typical benchmark functions in two dimensions. In addition, to examine its performance at higher dimensions, 16 functions from the test set were tested at 10 and 100 dimensions. The evaluation results of the proposed algorithm are compared to the outcomes of three modern optimization algorithms, namely (Whale optimization algorithm, Teacher-Learner based algorithm, and Covariance matrix adaptation evolution strategy). The results revealed that the proposed algorithm outperformed the conventional algorithms at lower dimensions in all test functions and showed a relatively better performance than the other algorithms at higher dimensions.

Keywords Directional-based crossover, Global Optimization, Genetic Algorithm, Normal Mutation

AMS 2010 subject classifications 90C26, 65K10

DOI:10.19139/soic-2310-5070-1796

1. Introduction

Modeling and analysis of many practical applications (e.g., finance, health, and engineering) involve an optimization process to search for the best design parameters that optimize the measure of model efficiency (e.g., losses, and gains) [1, 2, 3, 4], many of which are described by the global optimization problem:

$$min\ f(X),\ \ s.t.\ X \in \Omega \tag{1}$$

Where $X = \{x_1, x_2, ..., x_i\} \in \Omega$ is a design vector, the search space $\Omega \subset \mathbb{R}^d$ is a hyper-interval defined by the upper and lower bounds of each dimension and $f : \mathbb{R}^d \to \mathbb{R}$ is a real-valued objective function. Although gradient-based methods [5, 6] offer a direct and efficient way to optimize (1), they require further advanced computations when convexity or differentiability of $f(.)$ is not guaranteed [7, 8], in addition to the increasing complexity emerging when the search space gets larger [9, 10].

*Correspondence to: Ahmed M. Abdelkhalek (Email: g18013180@eng.asu.edu.eg). Department of Physics and Engineering Mathematics, Faculty of Engineering, Ain Shams University. Cairo, Egypt (11517).

ISSN 2310-5070 (online) ISSN 2311-004X (print)

1.1. Metaheuristic optimization

Recently, metaheuristic optimization techniques have proven to be efficient in many applications especially with imperfect information about the objective function, in addition to their preferable properties (e.g., gradient-free and easy implementation) [11, 12]. Various types of metaheuristic algorithms have been introduced through the past few decades which can be categorized according to their search strategy in nine groups (physics, social, music, swarm, chemistry, biology, sports, math and hybrid-based)[13, 14, 15] Metaheuristic algorithms are also classified according to the number of candidate solutions involved in the search process. Some techniques are called ”population-based” which, in contrast to ”single-point- based” techniques, conduct optimization using a set of points/individuals called ”population” sampled from within the search space. A major benefit of having many points distributed over the search space is that most of the features of the objective function are captured and processed during iterations, leading to powerful exploration ability[12].
Genetic Algorithm GA is an important stochastic population-based technique inspired by natural evolution and was first introduced by Holland [16]. GAs have proven to be highly efficient in many practical applications such as resource allocation [17, 18], deep learning and neural networks [19, 20], and control systems [21, 22]. Moreover, GAs are superior to other local search algorithms in effectively searching large spaces effectively [23]. The GA iteratively applies stochastic evolution processes (selection-crossover-mutation), known as genetic operators, to a population of individuals, which are evaluated by a fitness function. The fitter individuals in each population have a higher chance of passing their characteristics to the next population [24], and the iterations end when a stopping condition is met. Unfortunately, because genetic operators are stochastic, the search process would be totally random if they were left to operate without any control, resulting in a slower convergence speed (or perhaps no convergence at all). Therefore, a control scheme should guide genetic operators to create convergence pressure and improve population characteristics. However, if the control scheme is very restrictive, the search process converges quickly before reaching the global optimum, and the algorithm falls into a local optimum. This case is called premature convergence [25, 26].

1.2. Genetic Algorithm.

The searching process of metaheuristic optimization algorithms comprises, in general, two main tools (local and global) search tools. The local search tool is resposible for exploiting neighboring area of the solution candidates for better solutions, whereas the global search tool explores the search space for new promising areas. The evolution process of GA applies local and global search through three steps (selection, crossover and mutation). Researchers sometimes utilize an extra step called a replacement operator, which is periodically applied to renew stagnant population and help the search process avoid getting trapped in a local optimum [27, 28].
Formally, let $P(t)$ be the population at iteration t. To create the next population $P(t+1)$, GA applies the following steps:

1. Selection is the process of choosing individuals from the current population $P(t)$ to form parents that participate in the crossover. In the canonical GA, for the population of N individuals, a selection operator will form N pairs of parents that will produce n children of the next population. However, there may be variations in the numbers of parents, children, or both.
2. 2) Crossover is the second genetic operator in which paired parents recombine their characteristics to form new children. Note that in canonical GA, not all mating must be reproduced by crossover; there is a probability $0 \leq Px \leq 1$ for a crossover to occur.
3. 3) Mutation is the third operator, which is very important because, if new individuals are generated by crossover only, many features that have not been represented in the initial population might never appear, and there is a high chance that the algorithm loses diversity. Therefore, mutation performs spontaneous changes in individuals after crossover to add more genetic diversity to the algorithm by enforcing mutated individuals to be in new areas of the search space. In addition, there is a probability of $0 \leq Pm \leq 1$ for an individual to undergo a mutation.
4. Although this is not one of the canonical GA operators, many researchers have used this step because the diversity introduced by crossover and mutation operators may not be sufficient to enhance the global

search ability, especially in the case of multi-extrema functions. Hence, the replacement operator is applied periodically to enrich the genetic diversity by preserving the best k individuals and randomly regenerating the remaining $N - k$ [28].

Finally, a new population $P(t + 1)$ with better characteristics is formed. This process continues until a certain stopping condition is satisfied.

1.3. *A brief review of existing methods*

Several studies have proposed modifications to genetic operators to improve solution quality, accelerate convergence speed, or both. For instance, the authors in [29] proposed grouping selection GS to improve the selection process because traditional selection methods are computationally insufficient [30, 31]. However, after several iterations, the proposed algorithm suffers from a loss of genetic diversity. Other works have introduced a uniform crossover UX, a directed crossover DX operator, and arithmetic, average, and bound crossover [32, 33]. However, these operators are computationally expensive and have a slower convergence speed. Similarly, several studies adopted improved mutation operators. For example, the authors in [34, 35, 36] proposed non-uniform, polynomial, group, and random transfer-vector-based mutation operators. Another perspective of research suggested to combine multible crossover operators to strengthen local search ability, see for instance [37]. Although these efforts have increased the performance of the algorithm, they are either problem-specific and cannot be generalized or computationally expensive [27].
Based on the survey, improving genetic operators entails a trade-off between the search capability, computational cost, and convergence speed.

1.4. *Contributions.*

In this paper, we propose an enhanced genetic algorithm EGA based on directional-based crossover and normal mutation to address global optimization problems that can compete with current techniques in low and high dimensions, with lower computation complexity and relatively faster convergence time.

1.5. *Organization.*

The rest of this paper is organized as follows: Section 2 explains the procedure of the proposed enhanced algorithm, with a detailed explanation of each step. Section 3 provides the experimental results and comparative performance profile analysis of the proposed algorithm tested on 40 standard optimization benchmark functions in two dimensions, 16 of the 40 tested in 10 and 100 dimensions and compared with three selected population-based optimization techniques (Whale optimization algorithm WOA [38], Teaching-Learning based algorithm TLBO [39], and Covariance Matrix Adaptation evolution strategy CMA-ES [40]). Section 4 presents the conclusions of this study.

2. The Proposed Enhanced Genetic Algorithm.

The motivation behind this work is to enhance the algorithm's local and global search abilities while maintaining computation efforts at an acceptable level. To achieve this, the proposed algorithm mainly utilizes the updated directional-based crossover UDX and the updated normal mutation UNM. These operators are improved versions of those introduced in [27], which showed promising performance; however, the search process slows as the dimensions increase. The adopted GS method forces all individuals to participate in the crossover, so that all features captured in the initial population are processed during iterations. The updated operators allow for the most recent fittest individuals attained so far to guide the search process. Finally, a replacement step is periodically applied to preserve the best individuals and reinitialize the remaining individuals to prevent population from stagnating. The introduced mutation and replacement steps are implemented to increase genetic diversity and enhance global search ability while maintaining the speed of convergence achieved so far. Algorithm 1 presents the steps of the proposed EGA, followed by a detailed description of each step.

Algorithm 1 EGA steps

```
Require: $(N, d, f(\mathbf{X}))$                                   ▷ input parameters
  step 1: $P(0) \leftarrow$ initial population $N, d$
  while stopping criteria = false do
      step 2: $(p_a(t), p_b(t)) \leftarrow$ group selection$P(t)$
      step 3: $P(t) \leftarrow$ UDX($p_a(t), p_b(t)$)
      step 4: $P(t) \leftarrow$ UNM($P(t)$)
      step 5: $pr(t) \leftarrow$ replacement                         ▷ when necessary
      $P(t+1) \leftarrow$ new population
  end while
  return $X^*$, $f^*$                ▷ $X^*$ is the fittest individual in the final population
```

- **step 1.** *Initial Population:* Uniform sampling is the most common method for constructing the initial population, and this step can be improved using space-filling sampling techniques such as Latin hypercube sampling [41], quasi-random sequence [42], or stratified sampling [43]. However, these techniques add computational burden or may be infeasible at higher dimensions. Thus, the initial population in this study is generated uniformly from within the hyper-interval $[x^l, x^u]^d$, where x^l and x^u are the lower and upper bounds of the decision variable x_i, respectively.
- **step 2.** *Selection operator:* The core concept of the proposed algorithm is to guide the search process by fitter individuals in each population. Therefore, sorting and defining fitter and worse individuals are reasonable. The adopted GS takes a population $P(t) = \{X_1, X_2, ..., X_N\}$, and sorts it in ascending order according to the fitness values so that the first individual in the sorted population is the fittest. That is,

$$\widehat{P}(t) = \left\{X^1, X^2, ..., X^N\right\} \tag{2}$$

Where $f(X^1) \leq f(X^2) \leq ... \leq f(X^N)$. Then, $\widehat{P}(t)$ is divided into two groups $p_a(t)$ and $p_b(t)$, each of size $\frac{N}{2}$, i.e.

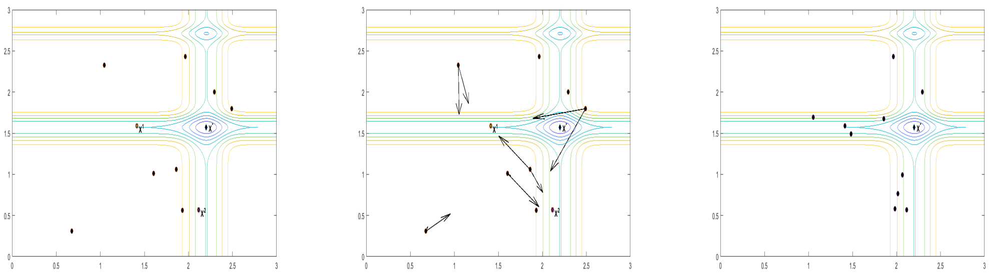
$$p_a(t) = \left\{X^1, .., X^{N/2}\right\}, \ p_b(t) = \left\{X^{N/2+1}, .., X^N\right\} \tag{3}$$

so that set $p_a(t)$ has fitter individuals than set $p_b(t)$. Pairing is performed between an individual from $p_b(t)$ and one from $p_a(t)$ to produce an offspring individual. Note that (3) enforces all individuals in the crossover process, that is, the crossover probability $Pr_x = 1$. Moreover, the selection scheme is directly based on the fitness value without the need to calculate it repeatedly. This simplicity makes it preferable in terms of computational expense.
- **step 3.** *Crossover operator:* In this study, we propose a UDX operator that continuously updates the position of the fittest individual. Note that all paired individuals outputting from (3) have one individual from the fitter set $p_a(t)$ and the other from the set $p_b(t)$. Let $(X_a(t), X_b(t))$ be paired parents and $X^1(t), X^2(t)$ be the fittest two individuals at iteration t; then, $\vec{v} = X_a(t) - X_b(t)$ is supposed to point at the descent direction. Thus, UDX produces two offspring individuals from each pair as follows:

$$\hat{X}_i(t+1) = X_a(t) + r_1 * \vec{v} + \underbrace{r_2 * \vec{v_1}}_{re-correction}, \tag{4a}$$

$$\hat{X}_{i+1}(t+1) = X_b(t) + r_1 * \vec{v} + \underbrace{r_2 * \vec{v_2}}_{re-correction} \tag{4b}$$

where $\vec{v_1} = X^{1:recent}(t) - X_a(t)$ and $\vec{v_2} = X^{2:recent}(t) - X_b(t)$ are re-correction directions and $r_1, r_2 \sim U(0,1)$. The first parts in (4a) and (4b) guide the search toward the region of the better individual $X_a(t)$, and the second part has a re-correction effect on the two fittest individuals $X^{1:recent}(t)$ and $X^{2:recent}(t)$.



(a) individuals before crossover (b) created directions (c) result offsprings

Figure 1. One iteration of UDX for a population of size 10 for the search space of the Michalewicz Function. (a) individuals before crossover, (b) created directions, (c) result offspring.

The name (recent) refers to the most recent fittest individual attained so far if any paired parents succeeded in creating an individual better than the fittest two in the current population. This step continuously updates the direction of the re-correction effect during the process, thereby accelerating the convergence speed. The reason for using the two fittest individuals in the re-correction part is to add diversity to the local search operator.

The crossover operation produces N individuals, which are compared with the original N parents and the best N of both constitutes offspring population. Using this scheme, there is a high chance of generating a direction that points toward the fittest individual or a similar direction, which increases the probability of creating better individuals and improves searchability and convergence speed. Fig. 1 illustrates a population of size 10 undergoing a crossover operation in the search space of the Michalewicz function. It shows 20 possible directions (two for each pair of parents), which will, in turn, spot the locations of the children. Increasing N increases the direction possibilities, thereby enriching the search diversity.

- **step 4.** *Mutation operator:* This operator is applied directly after crossover. Note that although UDX maintains a balance between convergence speed and diversity, there is no guarantee that this balance can last during all iterations. Additionally, a completely random mutation operator decreases the convergence speed of the algorithm. Therefore, the UNM operator was proposed to control the mutation process.

  Let $X^m = (x_1^m, x_2^m, ..., x_d^m)$ be the individual to be mutated at population t, and $X^{1:recent}$ be the most recent fittest individual in the current population $P(t)$ as before. A mutated individual $\widehat{X}$ is generated using (5a). Mutated individuals have a high chance of being in the vicinity of the original individual and will not decrease the speed of convergence.

$$\widehat{x}_i \sim N(\mu_i, \sigma_i), \;\; i = 1, 2, ..., d \tag{5a}$$

$$\mu_i = x_i^m \tag{5b}$$

$$\sigma = \frac{\left| x_i^{1:recent} - x_i^{N/2+1} \right|}{6} \tag{5c}$$

  According to the adaptive nature of equation (5c), in the beginning, there is a large difference between the fittest individual and all others in the current population. Consequently, σ is large, and the mutated individuals are spread over the search space, enhancing global search-ability. However, after several iterations, this difference decreases, and σ decreases as well. Thus, the mutated individuals spread closer to the fittest element, enhancing the local search-ability.

- **step 5.** *Replacement:* According to the mutation operator presented in subsequent populations, all individuals will continue to follow the fittest one in the population. All other individuals become closer as the standard deviation of UNM becomes smaller, eventually causing the search process to stagnate. This requires additional action to help the algorithm to escape from this situation. Therefore, a replacement operation

Table 1. Parameters settings of the EGA

Pr_x	Pr_m	N			t_r	k
		d=2	d=10	d=100		
1	0.5	60	100	200	5	$N/4$

Table 2. Maximum FEs for each dimension

d	max FEs 1	max FEs 2	max FEs 3
2	1000	5000	10000
10	5000	10000	15000
100	10000	15000	20000

is necessary. Generally, replacement is applied periodically to every t_r population by preserving the best k individuals and randomly regenerating the remaining $N - k$ across the entire search space. According to the results in [44], the performance of the algorithm increases as the change in population size increases. This step mostly simulates the idea of variable population size, but it also directly affects the population diversity.

3. Experimental Results.

3.1. Algorithm Setup.

In the traditional GA, not all individuals participate in the crossover operation, and there is a chance Pr_x, typically $(0.5 \sim 0.9)$, for an individual to undergo a crossover operation. However, one major disadvantage of such setting is that some promising individuals not selected from the initial population may not appear at all as the evolution process continues. Therefore, the adopted selection operator is beneficial because all individuals participate in the crossover operation, that is, the crossover probability $Pr_x = 1$, allowing the exploitation of the characteristics of the search space as much as possible. Conversely, not all individuals suffer mutation, and there is a chance Pr_m for each individual to undergo UNM. In our experiments, we set Pr_m to 0.5. The population size is set to 60, 100, and 200 for dimensions 2, 10, and 100, respectively. The number of elitist individuals to be preserved in each iteration is $N/4$, and the replacement operation is performed every 5^{th} population by replacing $3N/4$ individuals. Table 1 summarizes the parameter settings of the algorithm.

3.2. Experiment Setup.

To examine the effectiveness of the proposed algorithm, we tested and compared it with three modern optimization techniques (WOA, TLBO, and CMA-ES). These algorithms were tested on a set of 40 standard benchmark functions at $d = 2$ and 16 functions from the set at $d = 10$ and $d = 100$. The initial popualtion must be the same for all algorithms, so the random number population seed is set to (5) in each run. The steps of the experiment are as follows:

1. Run the algorithms on the 40 test functions at d = 2 and 16 test functions at d = 10, d = 100 (Table 3).
2. Replicate each algorithm 30 times on each function.
3. Terminate the iterations of each algorithm after the maximum number of function evaluations FEs specified in Table 2 is reached.
4. Keep track of the best objective values during iterations.
5. Perform performance profile analysis.

To evaluate our experiment, we used two performance analysis methods: the modified performance profile method (MPPM) [45], which measures how well the algorithms can estimate the global optimum compared to one another for different levels of the FEs budget, and the stochastic performance profile method (PPMS) described in [46], which compares algorithms in terms of sample means and sample variances.

Table 3. Test functions and dimensions

no.	function name	dimension tested	no.	function name	dimension tested
$f_1(x)$	Modified Ackley's Function	2,10,100	$f_{21}(x)$	Holder Table Function	2
$f_2(x)$	Ackley's Function	2,10,100	$f_{22}(x)$	Levy function Function	2,10,100
$f_3(x)$	Adjiman Function	2	$f_{23}(x)$	Levy N.13 Function	2
$f_4(x)$	Alpine N.1 Function	2,10,100	$f_{24}(x)$	Matyas Function	2
$f_5(x)$	Alpine N.2 function	2,10,100	$f_{25}(x)$	Michalewicz Function	2
$f_6(x)$	Booth Function	2,10,100	$f_{26}(x)$	Perm, 0, d, β function	2,10,100
$f_7(x)$	Bohachevsky function	2	$f_{27}(x)$	Perm, d, β function	2,10,100
$f_8(x)$	Bird Function	2	$f_{28}(x)$	Powell Function	2,10,100
$f_9(x)$	Biggs Exp2 function	2	$f_{29}(x)$	Rastrigin's Function	2,10,100
$f_{10}(x)$	Beale Function	2	$f_{30}(x)$	Rosenbrock Function	2,10,100
$f_{11}(x)$	Bartels Conn Function	2	$f_{31}(x)$	Schaffer N.2 Function	2
$f_{12}(x)$	Branin N.2 Function	2	$f_{32}(x)$	Schaffer N.4 Function	2
$f_{13}(x)$	Branin N.1 Function	2	$f_{33}(x)$	Schwefel Function	2,10,100
$f_{14}(x)$	cross-in-tray Function	2	$f_{34}(x)$	Six-hump camel Function	2
$f_{15}(x)$	Dixon-Price Function	2,10,100	$f_{35}(x)$	Shubert Function	2
$f_{16}(x)$	Drop-wave Function	2	$f_{36}(x)$	Styblinski-Tang Function	2,10,100
$f_{17}(x)$	Easom Function	2	$f_{37}(x)$	Trefethen Function	2
$f_{18}(x)$	Egg-holder Function	2	$f_{38}(x)$	Tripod Function	2
$f_{19}(x)$	Goldstein-Price Function	2	$f_{39}(x)$	Wheeler ridge Function	2
$f_{20}(x)$	Griewank Function	2,10,100	$f_{40}(x)$	Zakharov Function	2,10,100

3.3. Performance Analysis.

Both the MPPM and PPMS methods are extensions of the performance profile method (PPM) in [47]. The original PPM compares different derivative-free optimization algorithms by generating CDFs to display raw data from experiments in terms of the computational cost parameter $t_{p,s}$, which is the number of FEs required for algorithm s on problem p to find a solution x that meets the convergence condition:

$$\frac{f(x_0) - f(x)}{f(x_0) - f(x_L)} \geq 1 - \epsilon \tag{6}$$

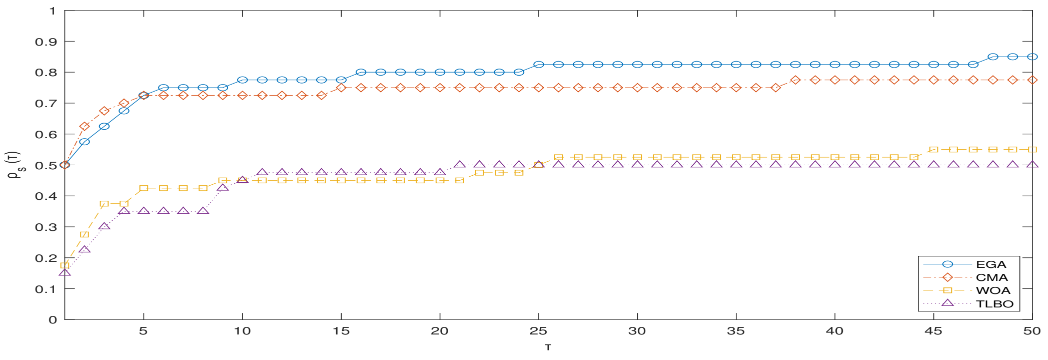
where $\epsilon > 0$ is a given error (typically $10^{-i}, i = 1, 2, 3, ...$). $f(x_0)$ is the best function value in the initial population (it must be the same for all compared algorithms), and $f(x_L)$ is the best function value found by any algorithm within a given FEs budget. However, in MPPM, $t_{p,s}$ is replaced by a function value improvement measure scaled to the best function value f^* found by algorithm s on problem p; that is:

$$m_{p,s} = \frac{\widehat{f}_{p,s} - f^*}{f_w - f^*} \tag{7}$$

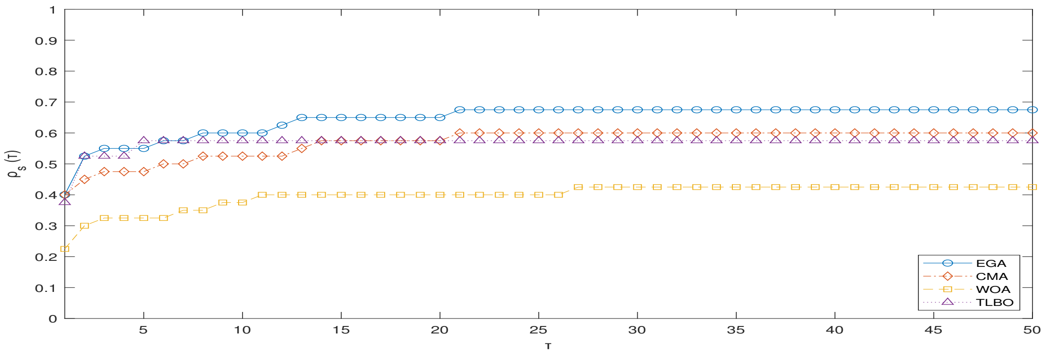
where $\widehat{f}$ is the average function value obtained during a given level of FEs over the 30 replications, f_w is the worst function value found by any algorithm for problem p.

Conversely, in PPMS, the parameter $t_{p,s}$ is used, but the original raw data in 6 is replaced by two stochastic measures, the average $\mu_{p,s}$ and variance $\sigma^2_{p,s}$, of the best function values during a given level of FEs of a given problem p by a given algorithm s. These two measures consider the stochastic nature of the algorithms by introducing a comparison based on the general confidence bounds.

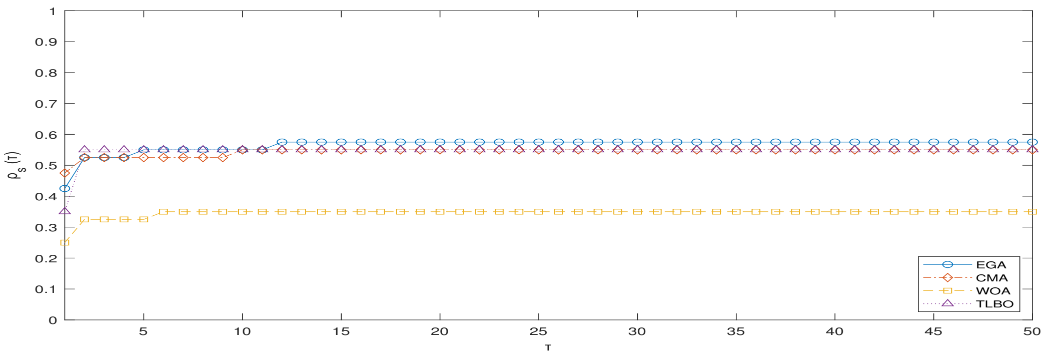
3.3.1. Modified Performance Profile Method Each curve in the plots is non-decreasing and indicates the performance of each algorithm. The vertical axis $\rho_s(T)$ represents the percentage of all problems for which algorithm s has a performance ratio within a factor T. That is, for $T = 1$, $\rho_s(1)$ is the probability that an algorithm will first find the best solution on average. The final value of any curve depends on how well a particular algorithm



(a)



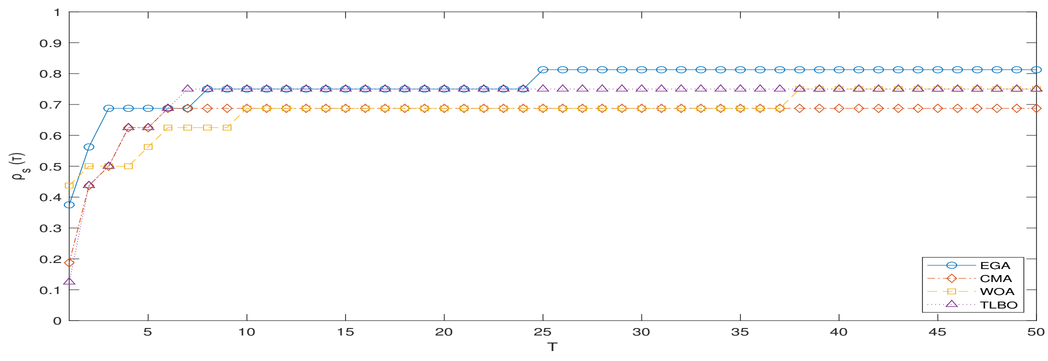
(b)



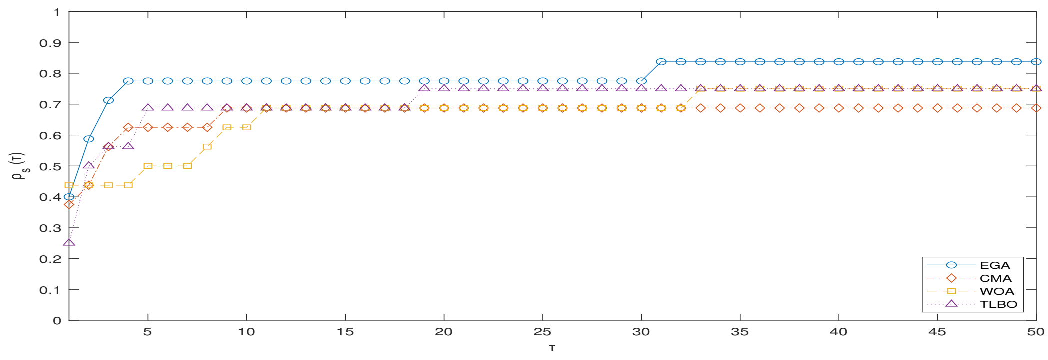
(c)

Figure 2. Performance Profile on Average of Best Objective Function Values for d=2 with (a) max Fe 1000. (b) max Fe 5000. (c) max Fe 10000.

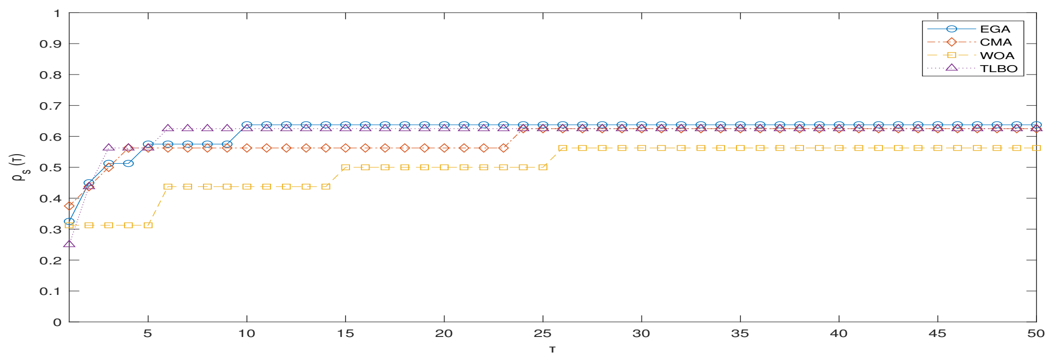
estimates the best solution among other algorithms. In other words, if the value of a curve is less than one at the last value of T, it indicates that on some problems, the corresponding algorithm became stuck in some local minima, and its best solutions were dominated by the best solutions found by other algorithms at multiples of T on the horizontal axis. Fig. 2 shows the performance profile of the algorithms at dimension $d = 2$ with three maximum FEs levels. When the maximum FEs was 1000, as shown in Fig. 2.(a), the EGA and CMA outperformed all other



(a)



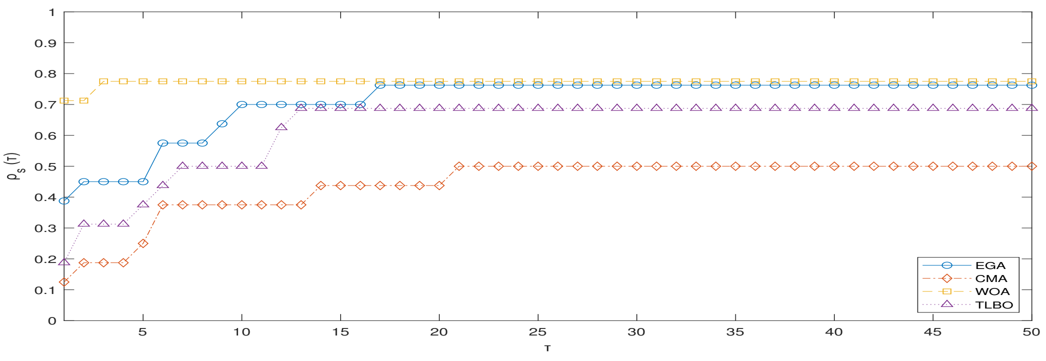
(b)



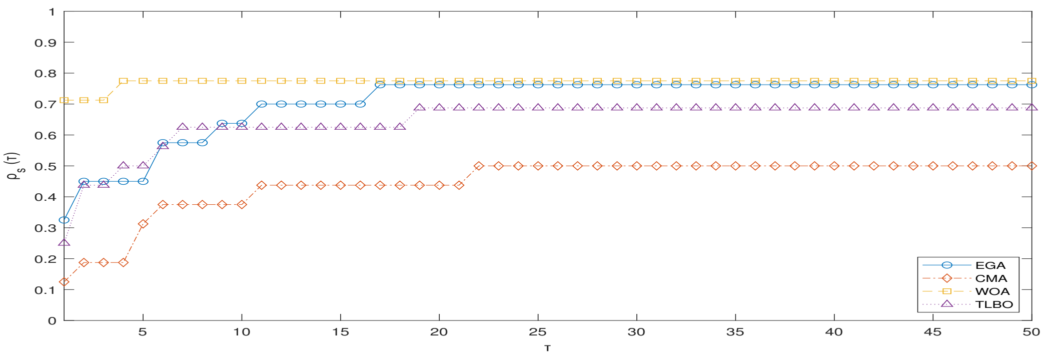
(c)

Figure 3. Performance Profile on Average of Best Objective Function Values for d=10 with (a) max Fe 5000. (b) max Fe 10000. (c) max Fe 15000.

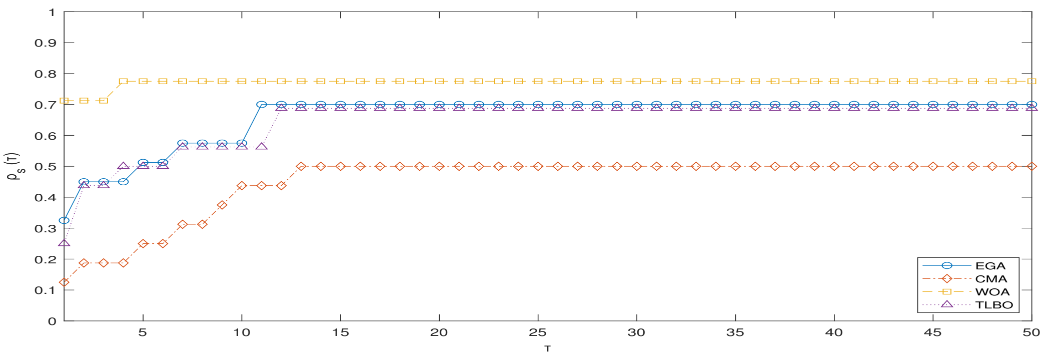
algorithms over the entire T range. When $T = 1$, EGA and CMA found the best solutions in approximately 50% of the total test functions, whereas at $T = 50$, EGA found the best solution in over 80% of the total test functions and was approximately 4% better than CMA.
At higher levels of maximum FEs, fig.2.(b) and 2.(c), WOA showed the worst performance among the algorithms, whereas EGA, CMA, and TLBO showed competitive performances against each other at almost all values of T.



(a)



(b)

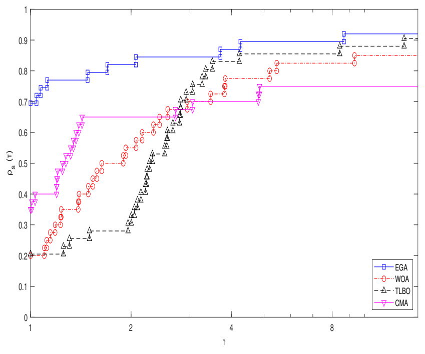


(c)

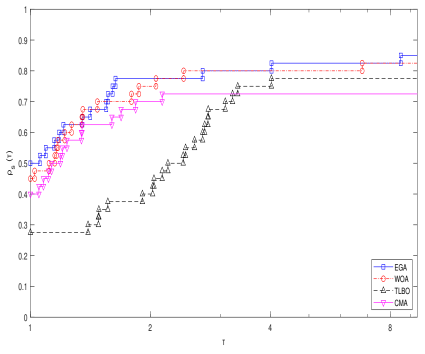
Figure 4. Performance Profile on Average of Best Objective Function Values for d=100 with (a) max Fe 10000. (b) max Fe 15000. (c) max Fe 20000.

In Fig.3, the performances of all algorithms for dimension $d = 10$ are relatively close to each other when the maximum FEs was 5000. However, the probability $\rho_s(1)$ of finding the best solution in the first place is twice that of WOA and TLBO, and EGA outperformed the other algorithms for $T > 15$. This situation is not the same when the level of FEs was 10000 and 15000. Fig. 3 shows that the probability of finding the best solution in the first place of the EGA degrades as the maximum FEs increases. However, the EGA succeeded in finding the best solution at approximately 70% and 60% when FEs were 10000 and 15000, respectively.

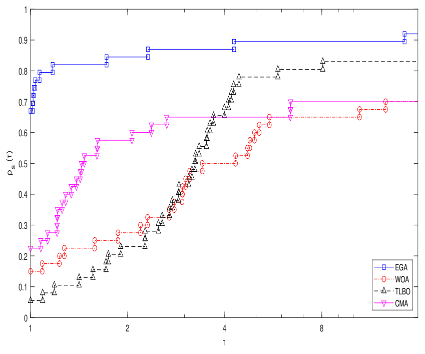
In higher dimensions, as shown in Fig. 4, the WOA dominates all other algorithms at the three levels of maximum FEs. The proposed EGA came in second place at the three levels with probability $\rho_s(1)$ of approximately $30\% - 40\%$ and succeeded in estimating the best solution for approximately $70\% - 80\%$ of test functions for $T > 10$, competing with TLBO, whereas CMA showed the worst performance among them.
The above discussion implies that the EGA performs well at low and moderate dimensions and has very promising performance at higher dimensions, particularly in applications where the maximum FEs budget is limited to lower values.



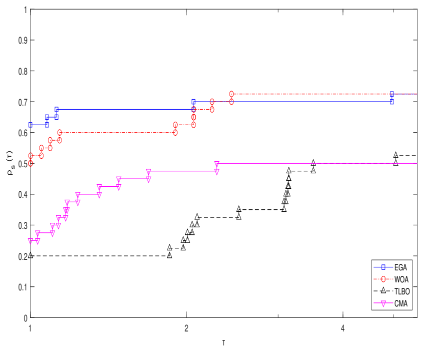
(a)



(b)



(c)



(d)

Figure 5. Stochastic Performance Profile for d=2. (a) sample means, $\epsilon = 10^{-3}$ (b) confidence bounds, $\epsilon = 10^{-3}$ (c) sample means, $\epsilon = 10^{-5}$ (d) confidence bounds, $\epsilon = 10^{-5}$

3.3.2. Stochastic Performance Profile Analysis. For the best function values found by algorithm s on problem p during a given FEs level for a given convergence error of ϵ, CDFs are generated for the sample means $\mu_{p,s}$ and variances $\sigma^2_{p,s}$. In this experiment, the level of the maximum FEs is set to 10000 and the convergence error to 10^{-3} and 10^{-5}. The plot of PPMS for each set of FEs and convergence error consists of two graphs; one is the CDFs for sample means that show the performance curve of each algorithm. The second graph shows the CDFs for the upper confidence bound of the winner algorithm from the first graph against the lower confidence bounds for the other algorithms to show whether there is a significant difference between the algorithms or not.
Fig.5 shows the profile plot of the sample means and confidence bounds for all algorithms in dimension $d = 2$ for a convergence error $\epsilon = 10^{-3}$ and 10^{-5}. Clearly, the proposed EGA performs better than the other algorithms, and

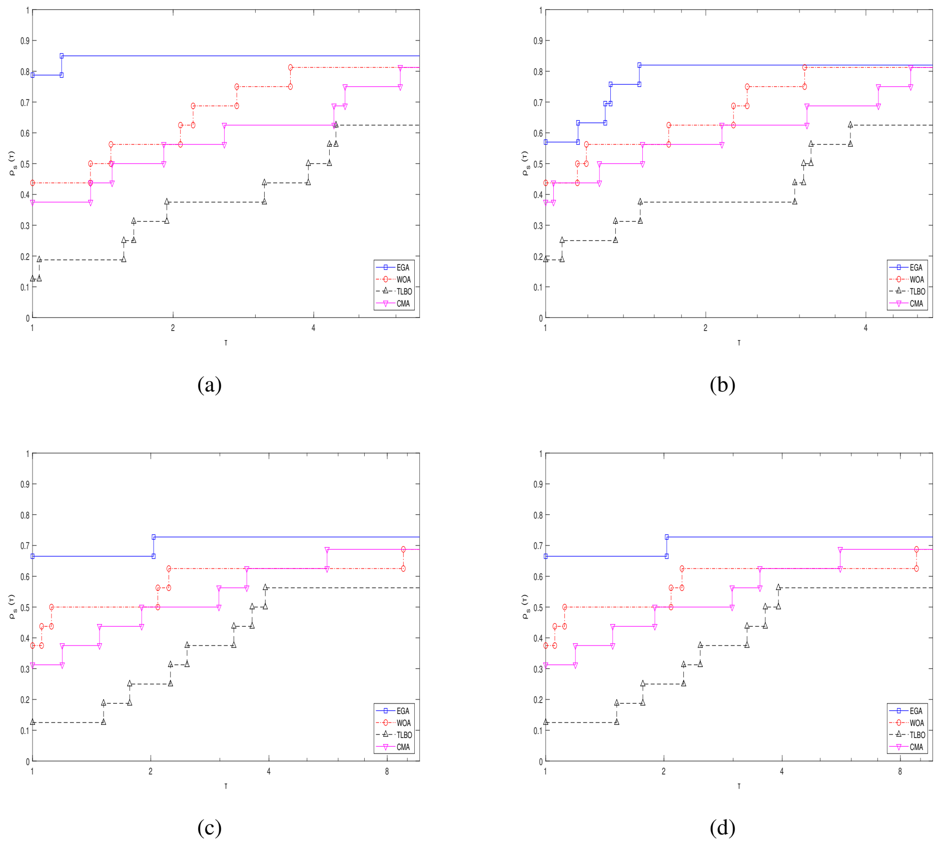
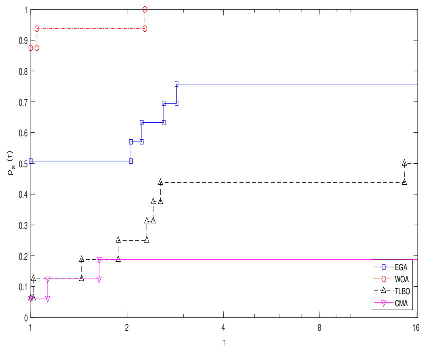


Figure 6. Stochastic Performance Profile for d=10. (a) sample means, $\epsilon = 10^{-3}$ (b) confidence bounds, $\epsilon = 10^{-3}$ (c) sample means, $\epsilon = 10^{-5}$ (d) confidence bounds, $\epsilon = 10^{-5}$

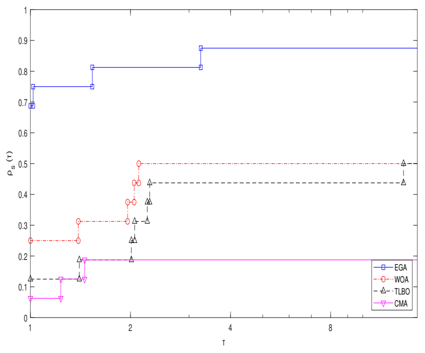
the profile plot of the confidence bounds shows that the EGA performance is significantly better than that of the other algorithms. A similar situation is illustrated in Fig.6. For dimension $d = 10$, EGA still performs better than the other algorithms in terms of the average best function values, and its upper confidence bound is better than the lower bounds of the other algorithms.
In Fig.7, the WOA indicates a better performance on the average best function values, which is consistent with the results of the previous section. However, Fig. 7(b) and (d) show that the lower confidence bounds of the EGA are better than the upper bounds of the rest. This result implies that there is no significant difference in the performance between them. Overall, the PPMS analysis results support the conclusion of the previous section; that is, the proposed EGA is much better in lower and moderate dimensions and performs quite well in higher dimensions.

4. Conclusion.

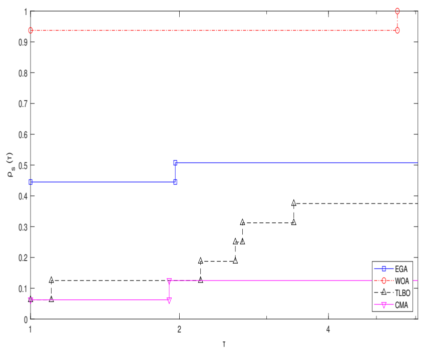
Whereas gradient-based methods show degrading efficiency in solving global optimization problems in cases where convexity or differentiability is not guaranteed, population-based techniques offer an efficient solution in the same situation. Genetic algorithm is a population-based technique inspired by nature and is widely used in global optimization problems owing to its good performance and simple implementation. Because of the stochastic nature of genetic operators, more control should be applied to guide the search process. However, this control



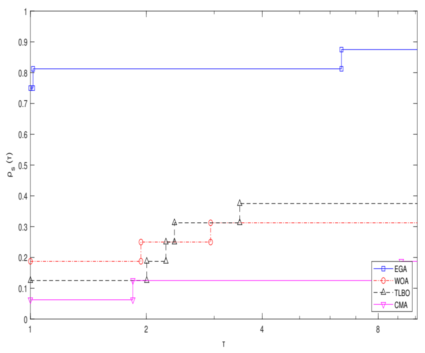
(a)



(b)



(c)



(d)

Figure 7. Stochastic Performance Profile for d=100. (a) sample means, $\epsilon = 10^{-3}$ (b) confidence bounds, $\epsilon = 10^{-3}$ (c) sample means, $\epsilon = 10^{-5}$ (d) confidence bounds, $\epsilon = 10^{-5}$

scheme entails a trade-off between the speed of convergence and search diversity. Therefore, to improve the GAs' performance, we propose an enhanced algorithm that depends on updated-directional-based crossover and updated normal mutation that continuously updates the location of the most recent fittest individual to accelerate the convergence speed without losing search diversity. The proposed algorithm is tested against three selected modern population-based optimization techniques (whale optimization algorithm, Teacher-Learner based optimization and covariance matrix adaptation evolution strategy). The performance profile analysis shows that EGA can outperform other algorithms at low dimensions and has a better performance, on average, at relatively high dimensions. In the future, the proposed EGA could be utilized in practical optimization problems, such as deep learning or control systems, and compared with existing solutions to examine its proficiency.

Appendix A: Matlab code for the algorithm

```
function [best,best_fit] =Ega(objective_function,max_FEs, Vars_upper,
   Vars_lower)
%% objective_function.. set to one of the test suit functions
%% max_FEs.. maximum function evaluations, set to values according to table
   2
```

```
%% Vars_upper, Vars_lower..are the upper and lower limits for each variable
   respectively
% ---- algorithm settings-------
Prm=.. probability of muataion
NP=..population size
D=.. problem dimension
tr=..replacement frequency
n=..number of elitest individuals
%..set to values in table 1

%--initial population--
pop = Vars_lower + (Vars_upper-Vars_lower)*rand(NP, D);
fit = objective_function(pop);
FEs=0; gen=1;
            for i = 1 : NP
                FEs = FEs + 1;
                if FEs > Max_FEs; break; end
            end
            [fit,index]=sort(fit,'ascend');
            pop= pop(index,:);
            elite=pop(1:n,:); elite_fit=fit(1:n);
while FEs < Max_FEs
pop_a=pop(randperm(NP/2),:);
pop_b=pop(NP/2 +1:end,:);
new_pop=[]; new_fit=[];
%--crossover---
for i=1:NP/2
    v = pop_a(i,:) - pop_b(i,:);
    v1= elite(1,:) - pop_a(i,:);
    v2= elite(2,:) - pop_b(i,:);
    child1= pop_a(i,:) + rand(1,D).*v + rand(1,D).*v1;
    child2= pop_b(i,:) + rand(1,D).*v + rand(1,D).*v2;
    [child1,child2]=bound(child1,child2);
    child_fit=objective_function([child1;child2]);
    FEs=FEs+2;
    if any(child_fit<elite_fit(1,2))
       [update_fit,id] = sort([elite_fit(1,2) child_fit]);
       dummy = [elite(1:2,:);child1;child2];
       update_elite= dummy(id,:);
       elite(1:2,:)= dummy(1:2,:);
    end
    if FEs > Max_FEs; break; end
    new_pop=[new_pop;child1;child];
    new_fit=[new_fit child_fit];
end
[new_fit,index]=sort(new_fit);
new_pop=sortrows(new_pop,index);

%---mutation---
for i=1:NP
```

```
    if rand < Prm
       C = abs(elite(1,:)-new_pop(NP/2 +1,:));
       mu = new_pop(i,:);
       new_pop(i,:) = mvnrnd(mu,C);
       new_pop(i,:)=bound(new_pop(i,:));
       new_fit(i)=objective_function(new_pop(i,:));
       FEs=FEs+1;
       if FEs > Max_FEs; break; end
       if new_fit(i)<elite_fit(1)
          elite_fit(1)=new_fit(i);
          elite(1,:)=new_pop(i,:);
       end
    end
end

%---replacement----
if mod(gen,tr)==0
   replace_pop= Vars_lower + (Vars_upper-Vars_lower).*rand(NP-n,D);
   replace_fit = objective_function(replace_pop);
   FEs=FEs+ NP -n;
   if FEs > Max_FEs; break; end
   new_pop = [elite;replace_pop];
   new_fit = [elite_fit replace_fit];
else
   [new_fit,index]=sort([elite_fit new_fit]);
   new_pop=sortrows([elite;new_pop],index);
   gen=gen+1;
end
  pop = new_pop(1:NP,:);
  fit = new_fit(1:NP);
end
  [best_fit,imin] = min(fit);
  best = pop(imin,:);
end

function x = bound(x,pop)

   xl = repmat(Vars_lower, 1, D);
   xu = repmat(Vars_upper,1,D);
   pos = x < xl;
   x(pos)=(pop(pos)+xl(pos))/2;
   pos = x > xu;
   x(pos)=(pop(pos)+xu(pos))/2;
end
```

REFERENCES

1. Jasbir Singh Arora. Introduction to optimum design fourth edition, 2017.
2. Michael D Intriligator. *Mathematical optimization and economic theory*. SIAM, 2002.

3. Abdur Rais and Ana Viana. Operations research in healthcare: a survey. *International transactions in operational research*, 18(1):1–31, 2011.
4. Ashok D Belegundu and Tirupathi R Chandrupatla. *Optimization concepts and applications in engineering*. Cambridge University Press, 2019.
5. Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.
6. Ersan Yazan and M Fatih Talu. Comparison of the stochastic gradient descent based optimization techniques. In *2017 International Artificial Intelligence and Data Processing Symposium (IDAP)*, pages 1–5. IEEE, 2017.
7. AM Bagirov, N Hoseini Monjezi, and S Taheri. An augmented subgradient method for minimizing nonsmooth dc functions. *Computational Optimization and Applications*, pages 1–28, 2021.
8. James V Burke, Frank E Curtis, Adrian S Lewis, Michael L Overton, and Lucas EA Simões. Gradient sampling methods for nonsmooth optimization. *Numerical nonsmooth optimization: State of the art algorithms*, pages 201–225, 2020.
9. Crina Grosan and Ajith Abraham. A novel global optimization technique for high dimensional functions. *International Journal of Intelligent Systems*, 24(4):421–440, 2009.
10. Christodoulos A Floudas and Panos M Pardalos. Recent advances in global optimization. 2014.
11. Mykel J Kochenderfer and Tim A Wheeler. *Algorithms for optimization*. Mit Press, 2019.
12. Zahra Beheshti and Siti Mariyam Hj Shamsuddin. A review of population-based meta-heuristic algorithms. *Int. J. Adv. Soft Comput. Appl*, 5(1):1–35, 2013.
13. Sinem Akyol and Bilal Alatas. Plant intelligence based metaheuristic optimization algorithms. *Artificial Intelligence Review*, 47(4):417–462, 2017.
14. Bilal Alatas and Harun Bingol. Comparative assessment of light-based intelligent search and optimization algorithms. *Light & Engineering*, 28(6), 2020.
15. Harun Bingol and Bilal Alatas. Chaos based optics inspired optimization algorithms as global solution search approach. *Chaos, Solitons & Fractals*, 141:110434, 2020.
16. John Henry Holland et al. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.
17. Fan-Hsun Tseng, Xiaofei Wang, Li-Der Chou, Han-Chieh Chao, and Victor CM Leung. Dynamic resource prediction and allocation for cloud data center using the multiobjective genetic algorithm. *IEEE Systems Journal*, 12(2):1688–1699, 2017.
18. Carlos Guerrero, Isaac Lera, and Carlos Juiz. Genetic algorithm for multi-objective optimization of container allocation in cloud architecture. *Journal of Grid Computing*, 16(1):113–135, 2018.
19. Hongqiang Li, Danyang Yuan, Xiangdong Ma, Dianyin Cui, and Lu Cao. Genetic algorithm for the optimization of features and neural networks in ecg signals classification. *Scientific reports*, 7(1):1–12, 2017.
20. Amit Kumar Gupta, Sharath Chandra Guntuku, Raghuram Karthik Desu, and Aditya Balu. Optimisation of turning parameters by integrating genetic algorithm with support vector regression and artificial neural networks. *The International Journal of Advanced Manufacturing Technology*, 77(1-4):331–339, 2015.
21. Arash Rikhtegar, Mohammad Pooyan, and Mohammad Taghi Manzuri-Shalmani. Genetic algorithm-optimised structure of convolutional neural network for face recognition applications. *IET Computer Vision*, 10(6):559–566, 2016.
22. Tengku Ahmad Faris Ku Yusoff, Mohd Farid Atan, Nazeri Abdul Rahman, Shanti Faridah Salleh, and Noraziah Abdul Wahab. Optimization of pid tuning using genetic algorithm. *Journal of Applied Science & Process Engineering*, 2(2), 2015.
23. Qifeng Lin, Wei Liu, Hongxin Peng, and Yuxing Chen. Efficient genetic algorithm for high-dimensional function optimization. In *2013 Ninth International Conference on Computational Intelligence and Security*, pages 255–259. IEEE, 2013.
24. Oliver Kramer. Genetic algorithms. In *Genetic algorithm essentials*, pages 11–19. Springer, 2017.
25. Ahmad Hassanat, Khalid Almohammadi, Esra'a Alkafaween, Eman Abunawas, Awni Hammouri, and VB Surya Prasath. Choosing mutation and crossover ratios for genetic algorithms—a review with a new dynamic approach. *Information*, 10(12):390, 2019.
26. Matej Črepinšek, Shih-Hsi Liu, and Marjan Mernik. Exploration and exploitation in evolutionary algorithms: A survey. *ACM computing surveys (CSUR)*, 45(3):1–33, 2013.
27. Yingying Song, Fulin Wang, and Xinxin Chen. An improved genetic algorithm for numerical function optimization. *Applied Intelligence*, 49(5):1880–1902, 2019.
28. Olympia Roeva, Stefka Fidanova, and Marcin Paprzycki. Population size influence on the genetic and ant algorithms performance in case of cultivation process modeling. In *Recent Advances in Computational Optimization: Results of the Workshop on Computational Optimization WCO 2013*, pages 107–120. Springer, 2015.
29. Yao-Chen Chuang, Chyi-Tsong Chen, and Chyi Hwang. A simple and efficient real-coded genetic algorithm for constrained optimization. *Applied Soft Computing*, 38:87–105, 2016.
30. RO Oladele and JS Sadiku. Genetic algorithm performance with different selection methods in solving multi-objective network design problem. *International Journal of Computer Applications*, 70(12), 2013.
31. Anupriya Shukla, Hari Mohan Pandey, and Deepti Mehrotra. Comparative review of selection techniques in genetic algorithm. In *2015 international conference on futuristic trends on computational analysis and knowledge management (ABLAZE)*, pages 515–519. IEEE, 2015.
32. Hsin-Chuan Kuo and Ching-Hai Lin. A directed genetic algorithm for global optimization. *Applied Mathematics and Computation*, 219(14):7348–7364, 2013.
33. Farah Ayiesya Zainuddin, Md Fahmi Abd Samad, and Durian Tunggal. A review of crossover methods and problem representation of genetic algorithm in recent engineering applications. *International Journal of Advanced Science and Technology*, 29(6s):759–769, 2020.
34. Na Sun and Yong Lu. A self-adaptive genetic algorithm with improved mutation mode based on measurement of population diversity. *Neural Computing and Applications*, 31:1435–1443, 2019.
35. Driss Saadaoui, Mustapha Elyaqouti, Khalid Assalaou, Souad Lidaighbi, et al. Parameters optimization of solar pv cell/module using genetic algorithm based on non-uniform mutation. *Energy Conversion and Management: X*, 12:100129, 2021.

36. Zbigniew Michalewicz, Thomas Logan, and Swarnalatha Swaminathan. Evolutionary operators for continuous convex parameter spaces. In *Proceedings of the 3rd Annual conference on Evolutionary Programming*, pages 84–97. World Scientific, 1994.
37. Monique Simplicio Viana, Orides Morandin Junior, and Rodrigo Colnago Contreras. A modified genetic algorithm with local search strategies and multi-crossover operator for job shop scheduling problem. *Sensors*, 20(18):5440, 2020.
38. Seyedali Mirjalili and Andrew Lewis. The whale optimization algorithm. *Advances in engineering software*, 95:51–67, 2016.
39. R Venkata Rao, Vimal J Savsani, and DP Vakharia. Teaching–learning-based optimization: an optimization method for continuous non-linear large scale problems. *Information sciences*, 183(1):1–15, 2012.
40. Nikolaus Hansen and Anne Auger. Cma-es: evolution strategies and covariance matrix adaptation. In *Proceedings of the 13th annual conference companion on Genetic and evolutionary computation*, pages 991–1010, 2011.
41. Felipe AC Viana. A tutorial on latin hypercube design of experiments. *Quality and reliability engineering international*, 32(5):1975–1985, 2016.
42. Mario A Navarro, Diego Oliva, Alfonso Ramos-Michel, Bernardo Morales-Castañeda, Daniel Zaldívar, and Alberto Luque-Chang. A review of the use of quasi-random number generators to initialize the population in meta-heuristic algorithms. *Archives of Computational Methods in Engineering*, 29(7):5149–5184, 2022.
43. V Roshan Joseph. Space-filling designs for computer experiments: A review. *Quality Engineering*, 28(1):28–35, 2016.
44. Vlasis K Koumousis and Christos P Katsaras. A saw-tooth genetic algorithm combining the effects of variable population size and reinitialization to enhance performance. *IEEE Transactions on Evolutionary Computation*, 10(1):19–28, 2006.
45. M Montaz Ali, Charoenchai Khompatraporn, and Zelda B Zabinsky. A numerical evaluation of several stochastic algorithms on selected continuous global optimization test problems. *Journal of global optimization*, 31(4):635–672, 2005.
46. Qunfeng Liu, Wei-Neng Chen, Jeremiah D Deng, Tianlong Gu, Huaxiang Zhang, Zhengtao Yu, and Jun Zhang. Benchmarking stochastic algorithms for global optimization problems by visualizing confidence intervals. *IEEE Transactions on Cybernetics*, 47(9):2924–2937, 2017.
47. Elizabeth D Dolan and Jorge J Moré. Benchmarking optimization software with performance profiles. *Mathematical programming*, 91(2):201–213, 2002.

.