Applied Soft Computing 13 (2013) 600–614

Contents lists available at SciVerse ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

Adaptive directed mutation for real-coded genetic algorithms

Ping-Hung Tang [a], Ming-Hseng Tseng [a,b,*]

[a] School of Medical Informatics, Chung-Shan Medical University, Taiwan, ROC
[b] Information Technology Office, Chung Shan Medical University Hospital, Taiwan, ROC

ARTICLE INFO

Article history:
Received 20 February 2012
Received in revised form 12 June 2012
Accepted 9 August 2012
Available online 23 August 2012

Keywords:
Real-coded genetic algorithm
Function optimization
Adaptive directed mutation

ABSTRACT

Adaptive directed mutation (ADM) operator, a novel, simple, and efficient real-coded genetic algorithm (RCGA) is proposed and then employed to solve complex function optimization problems. The suggested ADM operator enhances the abilities of GAs in searching global optima as well as in speeding convergence by integrating the local directional search strategy and the adaptive random search strategies. Using 41 benchmark global optimization test functions, the performance of the new algorithm is compared with five conventional mutation operators and then with six genetic algorithms (GAs) reported in literature. Results indicate that the proposed ADM-RCGA is fast, accurate, and reliable, and outperforms all the other GAs considered in the present study.

1. Introduction

Many real-life applications can be modeled as nonlinear optimization problems and, often, their global optimal solution is sought [1]. A typical nonlinear global optimization problem follows the form

$$\frac{\text{maximize}}{\text{minimize}}\, f(\boldsymbol{x} = x_1, x_2, \ldots x_N) \text{ subject to } \boldsymbol{x} \in \Omega \quad (1)$$

where $\boldsymbol{x}$ is a continuous variable vector with search space $\Omega \subseteq R^N$, and $f(\boldsymbol{x})$ is a continuous real-valued function having N variables. The domain Ω is defined within the upper and lower limits of each dimension. The problem is to find the global optimal solution $\boldsymbol{x}^*$ with its corresponding global optimal function value $f(\boldsymbol{x}^*)$. Two major classes of optimization techniques for solving general nonlinear optimization problems can be found in the literature, namely, gradient-based optimizers and evolutionary algorithm optimizers [1,2].

All gradient-based optimizers (also called deterministic optimizers) are point-by-point algorithms and are therefore local optimization techniques in nature. Gradient-based optimization techniques start the search procedure with an initial guess solution. If this guess solution does not come close enough to the global optimal solution, the gradient-based optimization techniques are likely to be trapped in the local optimal solution. In practice, finding such a suitable starting solution is the major difficulty when trying to optimize automatically. Gradient-based optimizers with about 20 variables are usually impractical [2] because as the number of variables increases, so does the number of evaluations. Most of them are designed to solve a particular class of optimization problems with few variables.

In other words, all evolutionary algorithm optimizers work with random sets of potential solutions—they are stochastic searching algorithms and therefore global optimization methods. Evolutionary algorithm optimizers generally scale well to solve higher dimensional optimization problems by comparing with gradient-based optimizers. Evolutionary algorithms consist of three population-based heuristic methodologies: genetic algorithms (GAs), evolutionary programming, and evolutionary strategies. GAs are perhaps the most popular evolutionary algorithms [3].

In traditional GA implementations [4,5], the decision variables were encoded as binary strings, namely, binary coded genetic algorithm (BCGA). The performance of BCGA has been satisfactory on small- and moderate-size problems requiring less precision in the solution, but BCGA entails huge computational time and memory [6] for high-dimensional problems that call for greater precision. To improve these drawbacks when applying BCGA to multidimensional and high-precision numerical problems, the decision variables can be encoded as real numbers, namely, real-coded genetic algorithm (RCGA), which has become increasingly popular [1,7]. The superiority of RCGA to BCGA has been established for continuous optimization problems [8] and medical data mining [9].

The performance of GAs relies on efficient search operators to guide the system toward global optima. One problem afflicting GAs is premature convergence. To mitigate or even avoid trapping into the local optima, the mutation operator provides a mechanism to explore new solutions and maintains the diversity of the population

* Corresponding author at: School of Medical Informatics, Chung-Shan Medical University, Taiwan, ROC.
E-mail address: mht@csmu.edu.tw (M.-H. Tseng).

1568-4946/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.asoc.2012.08.035

in GAs search, but it does so at the cost of slowing down the learning process. In GAs literature, relatively less effort has been put into designing a new mutation operator for RCGAs [1]. The step size and search direction are major factors that determine the performance of mutation operator [10]. The present study seeks to propose a novel, simple, and efficient RCGA based on the adaptive directed mutation (ADM) operator, and whose performance is demonstrated on a set of complex function optimization problems.

The remainder of this paper is organized as follows: Section 2 gives a brief review of the mutation operator in RCGAs. Section 3 provides a detailed description of the proposed methodology. The set of benchmark problems, the compared algorithms, and the experimental results are reported in Section 4. Finally, Section 5 presents a number of conclusions from the present study.

2. Review of literature on mutation operator

In general, a typical RCGA involves three main operators—selection, crossover, and mutation—to evolve the fitness of a population of guesses over a sequence of generations toward convergence at the global optimum. The method can be viewed as an evolutionary process. The mutation operation is used to change the offspring genes. Mutation is a key operator to increase the diversity of the population, hence enabling GAs to explore promising areas of the search space [10]. For common mutation operations, the random mutation (RM), uniform mutation, non-uniform mutation (NUM), polynomial mutation (PLM), and Gaussian mutation can be found [1,11].

Research effort has recently been spent to improve GAs performance by using different mutation techniques. Following the concept of induced mutation in biological systems, Bhandari et al. [12] first used directed mutation technique to improve BCGAs. Based on gradient or extrapolation, the directed mutation deterministically introduces a new point in the population guided by the information acquired in the previous generations. Zhou and Li [13] proposed a directed variation technique for mutation operator to adjust some individuals by using the feedback information from the current population. Berry and Vamplew [14] suggested a co-evolutionary technique where each component of a solution vector is added one extra bit to determine the direction of mutation by using the feedback information from the current population. Temby et al. [15] introduced a directed mutation based on momentum, where each component of an individual is attached a standard Gaussian mutation and the current momentum to mutate that component. Korejo et al. [10] proposed a directed mutation operator to improve the directed variation technique [13], in which the statistics information regarding the fitness and distribution of individuals over intervals of each dimension is calculated according to the current population and is used to guide the mutation of an individual toward the neighboring interval that has the best statistics result in each dimension.

Srinivas and Patnaik [16] described an adaptive BCGA for multimodal function optimization. In this adaptive GA, the probabilities of crossover and mutation are varied depending on the fitness values of the solutions. High-fitness solutions are protected while solutions with sub-average fitness are totally disrupted. According to the information of population evolutions in the impact of changes on fitness level, Chen and Liao [17] suggested an adaptive mutation operator to appropriate adjustment searching policies in RCGAs by using the simulation of gradient or counter-gradient direction. Tseng and Liao [18] proposed two adaptive strategies to improve the evolutionary efficiency of GAs. One strategy is to change crossover operators, which can randomly substitute the current crossover operator for another crossover operator at any time. The other strategy is to implement an adaptive adjustment of the crossover and mutation rates to increase the level of genetic diversity and guide the system toward global optimum if the system sinks into the local optimum. A state-of-the-art method for adaptive mutation is Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [19]. CMA-ES outperforms many other parametric optimization algorithms, as witnessed in the 2005 CEC algorithm contest, and is recommended by experts [20].

Ling and Leung [7] suggested the wavelet mutation, which is based on wavelet theorem. Deep and Thakur [1] designed the power mutation (PM) operator for RCGAs based on power distribution. By applying the underlying biological and mathematical idea to the generic framework of RCGAs, Vafaee and Nelson [21] proposed an adaptive mutation method based on the frequency of the best chromosomes' genes.

3. Methodology

3.1. The proposed ADM

The objective of the present study is to introduce a new mutation operator, namely, ADM, and to evaluate its performance against other mutation operators existing in literature. The ADM operator was designed to avoid both concentration of each chromosome caused by a crossover operator and an unsystematic search of the system due to RM. The ADM operator will introduce a new solution in the population. The new searching point is guided by the solutions obtained earlier based on the adaptive direction of gradient, hence its name. The direction of gradient is derived from the evolution of fitness value for each individual. The definition of $\Delta f(t-1)$ and $\Delta f(t)$ are variations of the fitness value for each chromosome $\boldsymbol{x}$ in the three consecutive generations ($t-2$, $t-1$, and t):

$$\Delta f(t) = f(\boldsymbol{x}(t)) - f(\boldsymbol{x}(t-1)) \tag{2}$$

$$\Delta f(t-1) = f(\boldsymbol{x}(t-1)) - f(\boldsymbol{x}(t-2)) \tag{3}$$

where $\boldsymbol{x} = \{x_1, x_2, \ldots x_k, \ldots x_N\}$is a chromosome, $f(\boldsymbol{x}(t))$ is the fitness value of chromosome $\boldsymbol{x}$ at the t generation. The variations of the k-dimensional gene for chromosome $\boldsymbol{x}$ in the three consecutive generations ($t-2$, $t-1$, and t) are defined as

$$\Delta x_k(t) = x_k(t) - x_k(t-1) \tag{4}$$

$$\Delta x_k(t-1) = x_k(t-1) - x_k(t-2) \tag{5}$$

Combining (2) with (5), the new solution of x_k will be iteratively updated as

$$x_k(t+1) = x_k(t) + (\Delta f(t),\ \Delta f(t-1),\ \Delta x_k(t),\ \Delta x_k(t-1), x_k(t), x_k^{UB}, x_k^{LB}) \cdot p_m \tag{6}$$

where x_k^{UB} and x_k^{LB} are the upper bound and lower bound of x_k, respectively. p_m is the adaptive probability of mutation [16]. It makes the bad chromosomes undergo a more substantial change in the population, and it can be expressed as follows:

$$p_m = \begin{cases} 0.5 \cdot \dfrac{f_{\max}(t) - f(x(t))}{f_{\max}(t) - \bar{f}(t)}, & \text{if} f(x(t)) \geq \bar{f}(t) \\ 0.5, & \text{if} f(x(t)) < \bar{f}(t) \end{cases} \tag{7}$$

where $f_{\max}(t)$ is the maximum fitness value of the population, $\bar{f}(t)$ is the average fitness value of population.

In Eq. (6), the function of $g(\cdot)$ can be termed as an acceleration function which controls the directed mutation. In the present study, four guided strategies were proposed based on nine different evolution trends for any chromosomes. These are "directional small-scale mutation," "random small-scale mutation," "random medium-scale mutation," and "random large-scale mutation." The

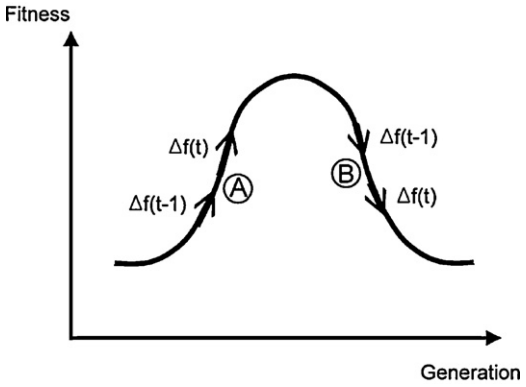

Fig. 1. Evolutionary trends of fitness value for $\Delta f(t-1)\cdot\Delta f(t)>0$.

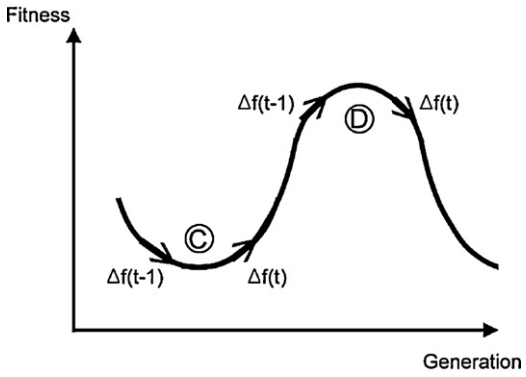

Fig. 2. Evolutionary trends of fitness value for $\Delta f(t-1)\cdot\Delta f(t)<0$.

nine different evolution trends for any chromosomes can be categorized into the following three conditions:

$$\text{Condition}(1):\ \Delta f(t-1)\cdot\Delta f(t)>0$$

The chromosome in the population sustains convergence or divergence of its fitness value to the global maximum as shown in Fig. 1. For the trend of continuous convergence as shown in case A, we increase the amount of mutation to accelerate the improvement of the fitness value. In contrast with case A, we move the solution in the opposite direction to suppress the divergent evolution of system for case B. Table 1 summarizes the three strategies of ADM for the case of $\Delta f(t-1)\cdot\Delta f(t)>0$.

For the case of $\Delta x_k(t-1)\cdot\Delta x_k(t)>0$, the new solution $x_k(t+1)$ of the k-dimensional gene in chromosome $\boldsymbol{x}$ is designed to follow the same or opposite direction of the original solution $x_k(t)$ to enhance the possibility of system convergence. The new point $x_k(t+1)$ can be generated as

$$x_k(t+1)=x_k(t)+\text{sign}(\Delta f(t))\cdot\Delta x_k(t)\cdot p_m \tag{8}$$

where $\text{sign}(z)=1$, if $z>0$; $\text{sign}(z)=-1$, if $z<0$; and $\text{sign}(z)=0$, if $z=0$.

For the case of $\Delta x_k(t-1)\cdot\Delta x_k(t)<0$, it means that the fitness value of the chromosome continues convergence (case A) or divergence (case B) even though the k-dimensional gene changes in different directions during the recent three generations. We used a small amount of mutation in the random direction because it is difficult to evaluate the effect between the k-dimensional gene and its corresponding fitness value. The new solution $x_k(t+1)$ can be defined as

$$x_k(t+1)=x_k(t)+|\Delta x_k(t)|\cdot r_s\cdot p_m \tag{9}$$

in which r_s is a uniformly distributed random number between -1 and 1.

In addition, when $\Delta x_k(t-1)\cdot\Delta x_k(t)=0$, the variation of the k-dimensional gene is zero at least one time during the recent three generations. We employed a medium amount of mutation in the random direction to mitigate or even avoid trapping local optima for this situation. The new solution $x_k(t+1)$ can be mutated as

$$x_k(t+1)=x_k(t)+x_k(t)\cdot r_s\cdot p_m \tag{10}$$

$$\text{Condition}(2):\ \Delta f(t-1)\cdot\Delta f(t)<0$$

Fig. 2 shows that the system is likely moving toward convergence (case C) and divergence (case D) when $\Delta f(t-1)\cdot\Delta f(t)<0$; it means that the chromosome solution is just over the region of hill or valley. Table 2 lists the three proposed strategies of ADM for the case of $\Delta f(t-1)\cdot\Delta f(t)<0$.

For case C, $\Delta f(t-1)>0$ and $\Delta f(t)<0$, the system may start to converge. To continue testing whether the search direction can improve the fitness value, the directional small amount of mutation will be used for the chromosomes that are better than the average of the fitness value in population. To take into account the system diversity and to improve the fitness value of the chromosome when $f(x(t))<\bar{f}(t)$, the strategies of random small-scale mutation [as Eq. (9)] and random medium-scale mutation [as Eq. (10)] were also employed for the cases of $\Delta x_k(t-1)\cdot\Delta x_k(t)<0$ and $\Delta x_k(t-1)\cdot\Delta x_k(t)=0$, respectively.

$$\text{Condition}(3):\ \Delta f(t-1)\cdot\Delta f(t)=0$$

Fig. 3 shows three evolution trends (cases E, F, and G) for the condition of $\Delta f(t-1)\cdot\Delta f(t)=0$. The corresponding strategies of ADM are listed in Table 3. For case G, $\Delta f(t-1)=0$ and $\Delta f(t)=0$, there are no variations of the fitness value, thus resulting in very low system diversity. We implemented a large amount of mutation to aid the original individual to escape this smooth region and avoid trapping local optima. The large-scale mutation is defined as

$$x_k(t+1)=\begin{cases}x_k(t)+(x_k^{UB}-x_k(t))\cdot r_s\cdot p_m, & \text{if } r<0.5\\ x_k(t)+(x_k(t)-x_k^{LB})\cdot r_s\cdot p_m, & \text{if } r\geq 0.5\end{cases} \tag{11}$$

where r is a uniformly distributed random number between 0 and 1.

For the case of $\Delta f(t-1)=0$ and $\Delta f(t)\neq 0$ (cases E), the strategy of the directional small-scale mutation [as Eq. (8)] was used to improve system convergence. On the other side, the strategy of random small-scale mutation [as Eq. (9)] was employed to increase

Table 1
Mutation strategies for $\Delta f(t-1)\cdot\Delta f(t)>0$.

Condition (1): $\Delta f(t-1)\cdot\Delta f(t)>0$	Mutation strategy
$\Delta x_k(t-1)\cdot\Delta x_k(t)>0$	Directional small scale
$\Delta x_k(t-1)\cdot\Delta x_k(t)<0$	Random small scale
$\Delta x_k(t-1)\cdot\Delta x_k(t)=0$	Random medium scale

Table 2
Mutation strategies for $\Delta f(t-1)\cdot\Delta f(t)<0$.

Condition (2): $\Delta f(t-1)\cdot\Delta f(t)>0$	Mutation strategy
$f(x(t))\geq\bar{f}(t)$	Directional small scale
$f(x(t))<\bar{f}(t)$ and $\Delta x_k(t-1)\cdot\Delta x_k(t)<0$	Random small scale
$f(x(t))<\bar{f}(t)$ and $\Delta x_k(t-1)\cdot\Delta x_k(t)=0$	Random medium scale

Table 3
Mutation strategies for $\Delta f(t-1)\cdot\Delta f(t)=0$.

Condition (3): $\Delta f(t-1)\cdot\Delta f(t)=0$	Mutation strategy
$\Delta f(t-1)=0$ and $\Delta f(t)\neq 0$	Directional small scale
$\Delta f(t-1)\neq 0$ and $\Delta f(t)=0$	Random small scale
$\Delta f(t-1)=0$ and $\Delta f(t)=0$	Random large scale

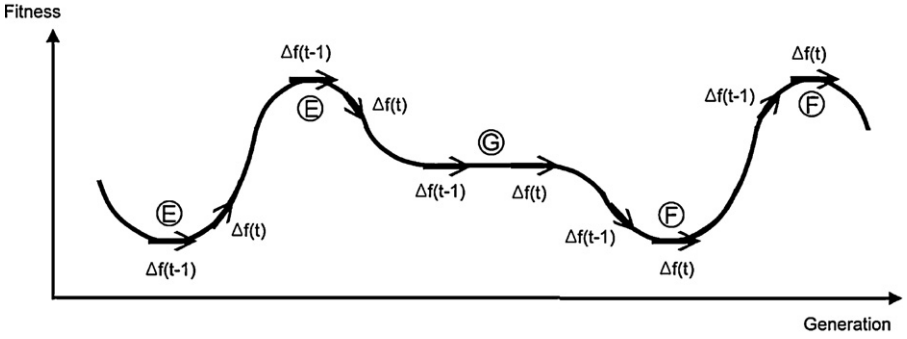

Fig. 3. Evolutionary trends of fitness value for $\Delta f(t-1)\cdot\Delta f(t)=0$.

system diversity for the case of $\Delta f(t-1) \neq 0$ and $\Delta f(t)=0$ (cases F) [as Eq. (10)].

3.2. The proposed ADM-RCGA

This section describes the real-coded genetic evolution process based on the proposed mutation operator ADM for function optimization. GA is basically an iterative population-based search technique that works on the concept of probability. Genetic operators, including selection, crossover, and mutation, are the kernels of GAs. Their primary job is to produce new combinations of parameters in line with the fitness function of each parameter's combination to achieve the purpose of evolution.

First, an initial population of chromosomes, where each gene is denoted by a real number, *P*, is randomly created from the lower and upper bounds of each decision variable. Second, each chromosome is evaluated by a defined fitness function; in this process, higher fitness-function values represent better chromosomes. Third, to improve the performance of convergence and to avoid reducing the level of genetic diversity within the population during the evolution process, a scaling method is needed whereby particularly good strings can be stopped from running away with the population in the earlier stages, while a degree of selection pressure can still be maintained in the final stages [5]. In the present study, a linear fitness scaling mechanism is applied. Fourth, some of the chromosomes are selected to undergo genetic operations for selection through stochastic universal sampling (SUS) [22]. Selection is a process by which pressure is applied on a population in a manner similar to that of natural selection found in biological systems. The chromosomes with high fitness values are chosen for reproduction, whereas poor-performance chromosomes may not be chosen at all. Fifth, genetic operation of crossover is performed. Crossover allows information to be exchanged in a way similar to that used by a natural organism undergoing sexual replication. Crossover operates by swapping a corresponding segment of a genetic representation of the parents and extends the search for new solutions in far-reaching directions. The crossover operator occurs with only some probability, which is called the *crossover rate* (p_c). A variant blend crossover (BLX-α) [23] incorporated with a crossover mask was implemented in the present study. Sixth, the mutation operation follows after the crossover operation. Mutation is used to randomly choose a member of the population and to change one randomly chosen aspect in its string representation. Although selection and crossover produce many new strings, they do not introduce any new information into the population at the gene level. The mutation process ensures that the probability of reaching any point in the search space is never zero. Mutation occurs with a certain level of probability, called the *mutation rate* (p_m). In the present study, we propose a new mutation operator, namely, ADM. Seventh, an elitism strategy is applied after performing the mutation operator. An elitism strategy may increase the speed with which a super individual dominates a population. For many applications, the search speed can be greatly improved by not losing the best, or elite, member between generations [5]. This study therefore presents a multi-elites mechanism based on probability p_e. The strategy not only keeps the best individual in the population, but also permits some secondary individuals to survive. After the processes of selection, crossover, mutation, and elitism have been applied to the initial population, a new population will have formed following the replacement step. After replacement, the new population will be evaluated based on its fitness in the next evolution. This process of selection, crossover, mutation, elitism, and replacement is continued until a fixed number of generations is reached or some form of convergence criterion is met. Fig. 4 demonstrates the working cycle of the proposed ADM-RCGA process. In the present study, these evolution processes involve programming using a Microsoft.NET framework. The details of the remaining operators are given below.

3.3. The remaining operators used in the present study

In this section, we define the selection, crossover, and other mutation operators employed in the present study, for example, SUS, BLX-α with crossover mask, RM, PLM, NUM, multi-non-uniform mutation (MNUM), and PM operators.

3.3.1. Stochastic universal sampling (SUS)

SUS is an elaborately named variation of roulette wheel selection. Instead of the single selection pointer used in roulette wheel methods, SUS employs a single random number to provide a

```
Procedure ADM-RCGA
{
    t = 0;
    Randomly generate initial population P(t);
    repeat{
        Evaluate P(t) to obtain its fitness;
        Scale the fitness values of P(t);
        while (not done) {
            Select two parents p1 and p2from P(t) based on theirfitness;
            Perform crossover for p1, p2 to produce c1 and c2 based onprobability pc;
            Mutate c1 or c2using ADM based on probability pm;
            Put c1 and c2 into P(t+1);
        }
        Put the elite members of P(t) into P(t+1) based onprobability pe;
        Generate the new generation P(t+1) to replace P(t);
        t = t + 1;
    }until (termination is met)
}
```

Fig. 4. The working cycle of ADM-RCGA.

Table 4
Summary of simulation conditions.

Experiment	Name of GA	Benchmark functions	Dimension	Population size	Maximum generations	Number of runs
Exp. 1	ADM–RCGA RM–RCGA PLM–RCGA NUM–RCGA MNUM–RCGA PM–RCGA	Benchmark functions I	30	300	30,000	30
Exp. 2	ADM-RCGA CMA-ES [19] HYK-GA [21]	Benchmark functions II	2, 10, 20, 30	100	40,000	100
Exp. 3	ADM-RCGA LX-PM [1]	Benchmark functions I	30	300	5000	30
Exp. 4	ADM-RCGA IEA [28] BOA [29] OGA [30]	Benchmark functions III	10, 100	30	12,000	30

starting position and the first selected individual. The selection process then proceeds to advance all the way around the wheel in equal-sized steps, in which the step size is determined by the number of individuals to be selected. SUS ensures that the observed selection frequencies of each individual are in line with the expected frequencies, thus achieving minimum spread. Standard roulette wheel selection does not make this guarantee. In addition, any individual can be selected entirely based on its position in the population; SUS has zero bias. For these reasons, SUS has become one of the most widely used selection algorithms in current GAs [24].

3.3.2. Blend crossover (BLX-α) with crossover mask

BLX-α [23] is defined as a combination of two selected parents $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$. It creates the children solutions lying in the range of $(\min(\boldsymbol{p}_1, \boldsymbol{p}_2) - \alpha|\boldsymbol{p}_1 - \boldsymbol{p}_2|, \max(\boldsymbol{p}_1, \boldsymbol{p}_2) + \alpha|\boldsymbol{p}_1 - \boldsymbol{p}_2|)$, where the constant α is to be selected, so that the children solutions do not come out of the range. In the present study, α is set to 0.25. The resulting offspring $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ are determined as follows:

$$\left\{ \begin{array}{l} \boldsymbol{c}_1 = \max(\boldsymbol{p}_1, \boldsymbol{p}_2) + \alpha|\boldsymbol{p}_1 - \boldsymbol{p}_2| \cdot r_s \\ \boldsymbol{c}_2 = \min(\boldsymbol{p}_1, \boldsymbol{p}_2) - \alpha|\boldsymbol{p}_1 - \boldsymbol{p}_2| \cdot r_s \end{array} \right\} \tag{12}$$

To select which variable would be mating in the two selected individuals, we employ crossover mask before performing the BLX-α operator. A binary string, called a crossover mask and equal in length to the solution strings, is randomly generated. If the crossover mask contains 1 on a specific bit position, then the offspring's value on that position will be calculated based on BLX-α from the two selected parents.

3.3.3. Random mutation (RM)

RM is also called uniform mutation. The mutated solution is obtained from the original solution using the rule given below [11].

$$x_k(t+1) = x_k(t) + \Delta(r - 0.5) \tag{13}$$

where r is uniform random numbers between 0 and 1, and Δ is the maximum value of perturbation defined by the user. In the present study, Δ is designed as follows:

$$\Delta = \max[2(x_k(t) - x_k^{LB}), 2(x_k^{UB} - x_k(t))] \tag{14}$$

3.3.4. Non-uniform mutation (NUM)

NUM is an operation with a fine-tuning capability. Its action depends on the generation number of the population [25]. From an original point $x_k(t)$, the muted point $x_k(t+1)$ is created as follows:

$$x_k(t+1) = \left\{ \begin{array}{ll} x_k(t) + h(t, x_k^{UB} - x_k(t)), & \text{if } r < 0.5 \\ x_k(t) + h(t, x_k(t) - x_k^{LB}), & \text{if } r \geq 0.5 \end{array} \right\} \tag{15}$$

The function h given below takes value in the interval $[0, y]$.

$$h(t, y) = y(1 - r^{(1-(t/t_{\max}))^b}) \tag{16}$$

where $t_{\max}$ is the maximum generation number of population, and b is a system parameter that determines the strength of the mutation operator. In the initial generations, NUM tends to search the space uniformly; in the later generations, it tends to search the space locally [1].

3.3.5. Multi-non-uniform mutation (MNUM)

A mutation operator increases the genetic diversity of a candidate individual. MNUM [26] was employed in the present study because it can perform uniform search with local fine-tuning and increase the capability to exploit a search space. The operator can be expressed by

$$x_k(t+1) = \left\{ \begin{array}{ll} x_k(t) + (x_k^{UB} - x_k(t)) \cdot A(t), & \text{if } r < 0.5 \\ x_k(t) + (x_k(t) - x_k^{LB}) \cdot A(t), & \text{if } r \geq 0.5 \end{array} \right\} \tag{17}$$

$$A(t) = \left[r_1 \left(1 - \frac{t}{t_{\max}} \right) \right]^b \tag{18}$$

Both r and r_1 are uniform random numbers between 0 and 1, and b is the shape parameter. In this study, parameter b is set to 2 in Eqs. (16) and (18).

3.3.6. Polynomial mutation (PLM)

Deb and Goyal [27] proposed a mutation operator based on polynomial distribution. The mutated solution is determined from the original solution as follows:

$$x_k(t+1) = x_k(t) + \bar{\delta} \cdot \delta_{\max} \tag{19}$$

$$\bar{\delta} = \left\{ \begin{array}{ll} (2r)^{(1/q+1)} - 1, & \text{if } r < 0.5 \\ 1 - [2(1-r)]^{(1/q+1)} & \text{if } r \geq 0.5 \end{array} \right\} \tag{20}$$

where q is a positive real number, r is a uniformly distributed random number between 0 and 1, and $\delta_{\max}$ is the user-defined maximum value of perturbation allowed between the original and

Table 5
Benchmark functions I. [1].

Test functions	x_i domain	Optimum
$f_1 = -20\exp\left(-0.02\sqrt{\frac{1}{N}\sum_{i=1}^{N}x_i^2}\right) - \exp\left(\frac{1}{N}\sum_{i=1}^{N}\cos(2\pi x_i)\right) + 20 + e$	$[-30, 30]$	0 (min)
$f_2 = 0.1\sum_{i=1}^{N}\cos(5\pi x_i) - \sum_{i=1}^{N}x_i^2$	$[-1, 1]$	$0.1N$ (max)
$f_3 = \exp\left(-0.5\sum_{i=1}^{N}x_i^2\right)$	$[-1, 1]$	1 (max)
$f_4 = 1 + \frac{1}{4000}\sum_{i=1}^{N}x_i^2 - \prod_{i=1}^{N}\cos\left(\frac{x_i}{\sqrt{i}}\right)$	$[-600, 600]$	0 (min)
$f_5 = \frac{\pi}{N}\left(10\sin^2(\pi y_1) + \sum_{i=1}^{N-1}(y_i-1)^2[1+10\sin^2(\pi y_{i+1})] + (y_i-1)^2\right)$, where $y_i = 1 + \frac{1}{4}(x_i+1)$	$[-10, 10]$	0 (min)
$f_6 = 0.1\left(\sin^2(3\pi x_1) + \sum_{i=1}^{N-1}(x_i-1)^2[1+\sin^2(3\pi x_{i+1})] + (x_n-1)^2[1+\sin^2(2\pi x_n)]\right)$	$[-5, 5]$	0 (min)
$f_7 = \left(\prod_{i=1}^{N}x_i\right)^{0.2} - \sum_{i=1}^{N}[(\ln(x_i-2))^2 + (\ln(10-x_i))^2]$	$[2, 10]$	≈997867.469 (max)
$f_8 = 10N + \sum_{i=1}^{N}[x_i^2 - 10\cos(2\pi x_i)]$	$[-5.12, 5.12]$	0 (min)
$f_9 = \sum_{i=1}^{N-1}[100(x_{i+1} - x_i^2)^2 - (x_i-1)^2]$	$[-30, 30]$	0 (min)
$f_{10} = \sum_{i=1}^{N}x_i\sin(\sqrt{\lvert x_i\rvert})$	$[-500, 500]$	12569.487 (max)
$f_{11} = \left[2.5\prod_{i=1}^{N}\sin\left(x_i - \frac{\pi}{6}\right) + \prod_{i=1}^{N}\sin\left(5\left(x_i - \frac{\pi}{6}\right)\right)\right]$	$[0, \pi]$	3.5 (max)
$f_{12} = \sum_{i=1}^{N}x_i^2 + \left(\sum_{i=1}^{N}\frac{i}{2}x_i\right)^2 + \left(\sum_{i=1}^{N}\frac{i}{2}x_i\right)^4$	$[-5.12, 5.12]$	0 (min)
$f_{13} = \sum_{i=1}^{N}x_i^2$	$[-5.12, 5.12]$	0 (min)
$f_{14} = \sum_{i=1}^{N}ix_i^2$	$[-5.12, 5.12]$	0 (min)
$f_{15} = \sum_{i=1}^{N}\lvert x_i\rvert + \prod_{i=1}^{N}\lvert x_i\rvert$	$[-10, 10]$	0 (min)
$f_{16} = \max_i\{\lvert x_i\rvert,\ \ 1 \le i \le N\}$	$[-100, 100]$	0 (min)
$f_{17} = \sum_{i=1}^{N}(x_i^4 + \text{rand}(0, 1))$	$[-10, 10]$	0 (min)
$f_{18} = \sum_{i=1}^{N}(x_i - i)^2$	$[-N, N]$	0 (min)
$f_{19} = \frac{\pi}{N}(10\sin^2(\pi y_1) + \sum_{i=1}^{N-1}(y_i-1)^2[1+10\sin^2(\pi y_{i+1})] + (y_i-1)^2) + \sum_{i=1}^{N}u(x_i,\ 10,\ 100,\ 4)$, where $y_i = \frac{1}{4}(x_i+1)$	$[-50, 50]$	0 (min)
$f_{20} = 0.1\left(\sin^2(3\pi x_1) + \sum_{i=1}^{N-1}(x_i-1)^2[1+\sin^2(3\pi x_{i+1})] + (x_n-1)^2[1+\sin^2(2\pi x_n)]\right) + \sum_{i=1}^{N}u(x_i,\ 10,\ 100,\ 4)$	$[-50, 50]$	0 (min)

In problem number 19 and 20, the value of penalty function u is given by the following expression $u(x, a, k, m) = \begin{cases} k \times pow((x-a), m) & \text{if } x > a, \\ -k \times pow((x-a), m) & \text{if } x < -a, \\ 0 & \text{otherwise.} \end{cases}$

Table 6
Benchmark functions II [19,21].

Test functions	x_i domain	Optimum
$g_1 = \sum_{i=1}^{N} x_i^2$	$[-100, 100]$	0 (min)
$g_2 = \sum_{i=1}^{N} \lvert x\rvert + \prod_{i=1}^{N} \lvert x\rvert$	$[-10, 10]$	0 (min)
$g_3 = \sum_{i=1}^{N-1} [100(x_{i+1} - x_i^2)^2 - (x_i - 1)^2]$	$[-29, 31]$	0 (min)
$g_4 = \sum_{i=1}^{N} i x_i^4 + \text{random}[0, 1]$	$[-1.28, 1.25]$	0 (min)
$g_5 = \frac{1}{4000} \sum_{i=1}^{N} x_i^2 - \prod_{i=1}^{N} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	$[-100, 100]$	0 (min)
$g_6 = \sum_{i=1}^{N} [x_i^2 - 10\cos(2\pi x_i + 10)]$	$[-5.12, 5.12]$	0 (min)
$g_7 = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\sum_{i=1}^{n} \frac{\cos(2\pi x_i)}{n}\right)$	$[-5.12, 5.12]$	0 (min)
$g_8 = -4x_1^2 + 2.1x_1^4 - \frac{1}{3}x_1^6 - x_1x_2 + 4x_2^2 - 4x_2^4$	$x_1 \in [-4.91017, 5.0893] x_2 \in [-5.7126, 4.2874]$	1.031 (max)
$g_9 = \left[\frac{1}{500} + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^{2} (x_i - a_{i,j})^6}\right]^{-1}$, $a_{i,j} = \begin{bmatrix} -32 & -16 & 0 & 16 & 32 & -32\ldots & 0 & 32 \\ -32 & -32 & -32 & -32 & -32 & -16\ldots & 32 & 32 \end{bmatrix}$	$[-98, 34]$	0.998 (min)

mutated solutions. In the present study, $q = 2$ is employed and $\delta_{\max}$ is designed as follows:

$$\delta_{\max} = \max[x_k(t) - x_k^{LB}, x_k^{UB} - x_k(t)] \tag{21}$$

3.3.7. Power mutation (PM)

Deep and Thakur [1] proposed a mutation operator based on power distribution. The PM is used to create the muted solution $x_k(t+1)$ in the vicinity of a parent solution $x_k(t)$ as follows:

$$x_k(t+1) = \begin{cases} x_k(t) - s \cdot (x_k(t) - x_k^{LB}), & \text{if } u < r \\ x_k(t) + s \cdot (x_k^{UB} - x_k(t)), & \text{if } u \geq r \end{cases} \tag{22}$$

where $u = (x_k(t) - x_k^{LB})/(x_k^{UB} - x_k^{LB})$, r is a uniformly distributed random number between 0 and 1, and a random number s is created based on the power distribution as follows:

$$s = p \cdot s_r^{p-1}, \quad 0 \leq s_r \leq 1 \tag{23}$$

where p is the index of the power distribution, and s_r is a uniform random number between 0 and 1. The strength of mutation is governed by the index p. For large values of p more diversity is expected; for small values of p, less perturbation in the solution is achieved. In the present study, $p = 0.5$ is employed.

4. Experiments and results

The present study aims to introduce a new mutation operator, ADM, and to evaluate its performance against five conventional mutation operators and six GAs existing in literature. Therefore, four experiments were conducted in this work. Table 4 summarizes the simulation conditions used in the four experiments. All the experiments are done on an Intel Xeon X5570 2.93 GHz machine with 16 GB RAM under WINXP platform.

4.1. Test functions

To investigate the performance of the proposed ADM-RCGA algorithm, 41 real-valued, well-known benchmark test functions were employed in the four experiments by comparing with five conventional mutation operators and six existing GAs. These global optimization test problems consist of different levels of complexity and multimodality, including continuous and discontinuous functions, as well as unimodal and multimodal functions. They are divided into three groups for four experiments in the present study, as shown in Table 4: functions f_1 to f_{20} [1] group as benchmark functions I, functions g_1 to g_9 group as benchmark functions II [19,21], and functions h_1 to h_{12} [28–30] group as benchmark functions III. The corresponding test function, parameter domain, and global optimum for each function are corrected and listed in Tables 5–7 for benchmark functions I to III, respectively.

4.2. Simulation settings

In the present study, the original RCGA has been augmented with ADM operator, in which each gene is represented by a 64-bit floating-point number. Meanwhile, selection and crossover and mutation are operated on the real-valued genes and the offspring are generated by SUS, BLX-α with crossover mask, and ADM operator. There are three different stop criteria: (1) the maximum number of generations has been reached; (2) 1000 or 2000 iterations remain in the same fitness value; and (3) the absolute error between the obtained solution and the global optimum is less than a threshold (in general, 10^{-8}). As long as any of the three conditions is met, the evolution process will be terminated in the proposed ADM-RCGA approach. Due to the stochastic nature of evolutionary algorithms, the performance of all compared algorithms on each test function is evaluated based on statistics obtained from

Table 7
Benchmark functions III [28–30].

Test functions	x_i domain	Optimum
$h_1 = \sum_{i=1}^{N} \left\lvert \frac{\sin(10x_i\pi)}{10x_i\pi} \right\rvert$	[−0.5, 0.5]	0 (min)
$h_2 = \sum_{i=1}^{N-1} \left[\sin(x_i + x_{i+1}) + \sin\left(\frac{2x_i x_{i+1}}{3}\right) \right]$	[3, 13]	≈2N (max)
$h_3 = \sum_{i=1}^{N} [x_i + 0.5]^2$	[−100, 100]	0 (min)
$h_4 = \sum_{i=1}^{N} [x_i^2 - 10\cos(2\pi x_i) + 10]$	[−5.12, 5.12]	0 (min)
$h_5 = \sum_{i=1}^{N} x_i^2$	[−5.12, 5.12]	0 (min)
$h_6 = \sum_{i=1}^{N} (x_i \sin(10\pi x_i))$	[−1, 2]	≈1.85N (max)
$h_7 = -\sum_{i=1}^{N} \left[\sin(x_i) + \sin\left(\frac{2x_i}{3}\right) \right]$	[3, 13]	≈1.21598N (max)
$h_8 = 20\exp\left(-0.2\sqrt{\frac{1}{N}\sum_{i=1}^{N} x_i^2}\right) + \exp\left(\sum_{i=1}^{N} \frac{\cos(2\pi x_i)}{N}\right) - 20 - e$	[−30, 30]	0 (max)
$h_9 = 418.9828N - \sum_{i=1}^{N} x_i \sin(\sqrt{\lvert x_i \rvert})$	[−500, 500]	0 (min)
$h_{10} = \sum_{i=1}^{N-1} [100(x_{i+1} - x_i^2)^2 - (x_i - 1)^2]$	[−5.12, 5.12]	0 (min)
$h_{11} = 6N + \sum_{i=1}^{N} \lvert x_i \rvert$	[−5.12, 5.12]	0 (min)
$h_{12} = \frac{1}{4000}\sum_{i=1}^{N} x_i^2 - \prod_{i=1}^{N} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	[−600, 600]	0 (min)

Table 8
Mean and standard deviation of fitness values and ranks achieved by 6 different mutation operators for 30 variables in 30 runs.

Test functions	ADM			RM			PLM		
	Mean (SD)	|Mean–f^*|	Rank	Mean (SD)	|Mean–f^*|	Rank	Mean (SD)	|Mean–f^*|	Rank
f_1	3.279E−01 (2.737E−02)	3.279E−01	5 (4)	1.424E−03 (7.777E−04)	1.424E−03	3 (2)	3.471E−04 (1.173E−04)	3.471E−04	1 (1)
f_2	3.00000 (4.714E−07)	3.333E−07	1 (1)	2.99995 (2.627E−05)	4.717E−05	3 (3)	2.99998 (9.229E−06)	2.350E−05	2 (2)
f_3	1.000000 (6.000E−08)	2.200E−07	1 (1)	0.999994 (2.387E−06)	5.660E−06	4 (4)	0.999996 (1.120E−06)	3.557E−06	3 (2)
f_4	4.432E−03 (7.872E−03)	4.432E−03	1 (1)	1.362E−02 (1.582E−02)	1.362E−02	3 (3)	5.736E−03 (7.968E−03)	5.736E−03	2 (2)
f_5	0.000E+00 (0.000E+00)	0.000E+00	1 (1)	2.925E−07 (2.685E−07)	2.925E−07	4 (4)	8.635E−08 (8.852E−08)	8.635E−08	3 (3)
f_6	0.000E+00 (0.000E+00)	0.000E+00	1 (1)	1.018E−06 (1.394E−06)	1.018E−06	4 (4)	3.397E−07 (6.662E−07)	3.397E−07	3 (3)
f_7	997867.15 (1.258E−01)	3.190E−01	1 (1)	997842.93 (1.596E+01)	2.454E+01	4 (4)	997854.04 (7.871E+00)	1.343E+01	3 (3)
f_8	1.360E+01 (2.482E+00)	1.360E+01	5 (4)	1.296E−02 (1.062E−02)	1.296E−02	2 (2)	4.310E−03 (2.389E−03)	4.310E−03	1 (1)
f_9	6.584E+00 (4.512E+00)	6.584E+00	2 (2)	6.981E+01 (4.075E+01)	6.981E+01	5 (4)	5.126E+01 (3.432E+01)	5.126E+01	3 (3)
f_{10}	1.192E+04 (5.996E+02)	6.514E+02	6 (6)	1.257E+04 (7.399E−02)	1.487E−01	2 (2)	1.257E+04 (3.263E−02)	9.467E−02	1 (1)

Table 8 (*Continued*)

Test functions	ADM			RM			PLM		
	Mean (SD)	|Mean–f^*|	Rank	Mean (SD)	|Mean–f^*|	Rank	Mean (SD)	|Mean–f^*|	Rank
f_{11}	3.50000 (0.000E+00)	0.000E+00	1 (1)	3.49939 (3.642E−05)	6.123E−05	4 (4)	3.499962 (2.605E−05)	3.837E−05	3 (3)
f_{12}	0.000E+00 (0.000E+00)	0.000E+00	1 (1)	4.052E+00 (5.574E+00)	4.052E+00	4 (4)	3.076E−01 (3.246E−01)	3.076E−01	3 (3)
f_{13}	0.000E+00 (0.000E+00)	0.000E+00	1 (1)	3.310E−06 (1.811E−06)	3.310E−06	5 (5)	5.618E−07 (3.460E−07)	5.618E−07	3 (3)
f_{14}	0.000E+00 (0.000E+00)	0.000E+00	1 (1)	7.572E−05 (1.083E−04)	7.572E−05	5 (5)	5.361E−06 (5.511E−06)	5.361E−06	3 (3)
f_{15}	0.000E+00 (0.000E+00)	0.000E+00	1 (1)	1.771E−03 (6.387E−03)	1.771E−03	4 (4)	3.358E−04 (7.331E−04)	3.358E−04	3 (3)
f_{16}	0.000E+00 (0.000E+00)	0.000E+00	1 (1)	2.431E−01 (2.371E−01)	2.431E−01	4 (4)	4.765E−02 (1.578E−02)	4.765E−02	3 (3)
f_{17}	5.832E−04 (2.638E−04)	5.832E−04	1 (1)	1.062E−01 (4.176E−02)	1.062E−01	4 (4)	5.003E−02 (2.161E−02)	5.003E−02	3 (2)
f_{18}	0.000E+00 (0.000E+00)	0.000E+00	1 (1)	3.977E−04 (2.793E−04)	3.977E−04	4 (4)	6.360E−05 (6.003E−05)	6.360E−05	3 (3)
f_{19}	0.000E+00 (0.000E+00)	0.000E+00	1 (1)	3.337E−06 (4.186E−06)	3.337E−06	5 (4)	2.112E−06 (3.811E−06)	2.112E−06	3 (3)
f_{20}	0.000E+00 (0.000E+00)	0.000E+00	1 (1)	5.660E−05 (7.331E−05)	5.660E−05	5 (5)	3.864E−06 (3.075E−06)	3.864E−06	3 (3)
Averaged rank			1.7 (1.6)			3.9 (3.75)			2.6 (2.5)
Final rank			1 (1)			4 (4)			3 (2)

Test functions	NUM			MNUM			PM		
	Mean (SD)	|Mean–f^*|	Rank	Mean (SD)	|Mean–f^*|	Rank	Mean (SD)	|Mean–f^*|	Rank
f_1	5.908E−04 (8.682E−04)	5.908E−04	2 (3)	1.816E−01 (6.085E−02)	1.816E−01	4 (5)	4.459E−01 (2.104E−01)	4.459E−01	6 (6)
f_2	2.99985 (1.355E−04)	1.473E−04	4 (4)	2.98028 (5.024E−02)	1.972E−02	6 (6)	2.99358 (5.855E−03)	6.420E−03	5 (5)
f_3	0.999986 (9.881E−06)	1.389E−05	5 (5)	0.999997 (1.724E−06)	2.663E−06	2 (3)	0.999560 (3.657E−04)	4.396E−04	6 (6)
f_4	2.447E−02 (2.351E−02)	2.447E−02	4 (4)	3.655E−02 (3.490E−02)	3.655E−02	5 (5)	1.021E+00 (8.691E−02)	1.021E+00	6 (6)
f_5	6.430E−07 (9.436E−07)	6.430E−07	5 (5)	0.000E+00 (0.000E+00)	0.000E+00	1 (1)	5.924E−04 (1.555E−03)	5.924E−04	6 (6)
f_6	3.772E−06 (1.094E−05)	3.772E−06	5 (5)	0.000E+00 (0.000E+00)	0.000E+00	1 (1)	1.425E−03 (3.002E−03)	1.425E−03	6 (6)
f_7	997817.27 (1.962E+01)	5.020E+01	5 (5)	997866.54 (3.470E−01)	9.290E−01	2 (2)	997452.45 (1.371E+02)	4.150E+02	6 (6)
f_8	4.432E−02 (3.624E−02)	4.432E−02	3 (3)	1.801E+01 (7.725E+00)	1.801E+01	6 (6)	8.233E+00 (6.066E+00)	8.233E+00	4 (5)
f_9	6.670E+01 (4.945E+01)	6.670E+01	4 (5)	4.032E+00 (4.002E+00)	4.032E+00	1 (1)	4.043E+03 (1.617E+04)	4.043E+03	6 (6)
f_{10}	1.257E+04 (3.200E−01)	2.827E−01	3 (3)	1.200E+04 (4.659E+02)	5.726E+02	5 (5)	1.255E+04 (5.940E+01)	1.726E+01	4 (4)
f_{11}	3.499794 (1.602E−04)	2.056E−04	5 (5)	3.499995 (1.526E−06)	4.733E−06	2 (2)	3.495383 (4.471E−03)	4.617E−03	6 (6)
f_{12}	4.877E+01 (3.077E+01)	4.877E+01	5 (6)	0.000E+00 (0.000E+00)	0.000E+00	1 (1)	7.091E+01 (3.038E+01)	7.091E+01	6 (5)
f_{13}	1.128E−06 (1.701E−06)	1.128E−06	4 (4)	0.000E+00 (0.000E+00)	0.000E+00	1 (1)	1.195E−02 (9.260E−03)	1.195E−02	6 (6)
f_{14}	2.504E−05 (4.956E−05)	2.504E−05	4 (4)	0.000E+00 (0.000E+00)	0.000E+00	1 (1)	4.649E−01 (3.203E−01)	4.649E−01	6 (6)
f_{15}	7.410E−03 (3.535E−02)	7.410E−03	5 (5)	0.000E+00 (0.000E+00)	0.000E+00	1 (1)	4.600E−02 (1.101E−01)	4.600E−02	6 (6)
f_{16}	9.551E−01 (1.646E+00)	9.551E−01	6 (6)	5.888E−08 (1.378E−07)	5.888E−08	2 (2)	8.571E−01 (1.087E+00)	8.571E−01	5 (5)
f_{17}	2.609E−01 (1.365E−01)	2.609E−01	5 (5)	3.240E−02 (2.250E−02)	3.240E−02	2 (3)	2.756E+00 (2.349E+00)	2.756E+00	6 (6)
f_{18}	7.913E−04 (1.287E−03)	7.913E−04	5 (5)	1.798E−08 (4.834E−08)	1.798E−08	2 (2)	6.917E−01 (1.018E+00)	6.917E−01	6 (6)
f_{19}	2.565E−06 (4.507E−06)	2.565E−06	4 (5)	0.000E+00 (0.000E+00)	0.000E+00	1 (1)	6.102E−02 (7.037E−02)	6.102E−02	6 (6)
f_{20}	3.032E−05 (5.318E−05)	3.032E−05	4 (4)	0.000E+00 (0.000E+00)	0.000E+00	1 (1)	5.891E−01 (3.165E−01)	5.891E−01	6 (6)
Averaged rank			4.35 (4.55)			2.35 (2.5)			5.7 (5.7)
Final rank			5 5			2 (2)			6 (6)

Table 9
t-Test result achieved by 6 different mutation operators for 30 variables in 30 runs.

Test functions	ADM–RM		ADM–PLM		ADM–NUM		ADM–MNUM		ADM–PM	
	p (t-test) (%)	Result	p (t-test)) (%)	Result	p (t-test)) (%)	Result	p (t-test)) (%)	Result	p (t-test)) (%)	Result
f_1	0.00	–	0.00	–	0.00	–	0.00	–	0.00	+
f_2	0.00	+	0.00	+	0.00	+	0.02	+	0.00	+
f_3	0.00	+	0.00	+	0.00	+	0.00	+	0.00	+
f_4	0.67	+	52.60	∼	0.01	+	0.00	+	0.00	+
f_5	0.00	+	0.00	+	0.08	+	∼	∼	0.02	+
f_6	0.04	+	0.93	+	6.90	∼	∼	∼	0.00	+
f_7	0.00	+	0.00	+	0.00	+	0.00	+	0.00	+
f_8	0.00	–	0.00	–	0.00	–	0.00	+	0.00	–
f_9	0.00	+	0.00	+	0.00	+	0.00	–	1.42	+
f_{10}	0.00	–	0.00	–	0.00	–	30.02	∼	0.00	–
f_{11}	0.00	+	0.00	+	0.00	+	0.00	+	0.00	+
f_{12}	0.04	+	0.00	+	0.00	+	∼	∼	0.00	+
f_{13}	0.00	+	0.00	+	0.11	+	∼	∼	0.00	+
f_{14}	0.07	+	0.00	+	0.97	+	∼	∼	0.00	+
f_{15}	13.97	∼	1.80	+	26.03	∼	∼	∼	0.01	+
f_{16}	0.00	+	0.00	+	0.35	+	0.00	+	0.00	+
f_{17}	0.00	+	0.00	+	0.00	+	0.00	+	0.00	+
f_{18}	0.00	+	0.00	+	0.21	+	0.03	+	0.00	+
f_{19}	0.01	+	0.50	+	0.41	+	∼	∼	0.00	+
f_{20}	0.02	+	0.00	+	0.40	+	∼	∼	0.00	+

Table 10
Average execution time, average number of generations and efficiency ranks achieved by 6 different mutation operators for 30 variables in 30 runs.

Test functions	ADM		RM		PLM		NUM		MNUM		PM	
	Time(s) (generation)	Rank	Time(s) (generation)	Rank	Time(s) (generation)	Rank	Time(s) (generation)	Rank	Time(s) (generation)	Rank	Time(s) (generation)	Rank
f_1	91.25 (2348)	1 (1)	3389.03 (29824)	6 (5)	3053.57 (30000)	5 (6)	874.63 (29590)	3 (4)	453.05 (17290)	2 (2)	933.23 (25554)	4 (3)
f_2	40.05 (1041)	1 (1)	963.21 (8186)	6 (4)	752.63 (6458)	4 (3)	301.56 (11582)	3 (5)	182.97 (4991)	2 (2)	876.98 (23647)	5 (6)
f_3	39.37 (1077)	1 (1)	902.13 (7692)	6 (4)	658.11 (5549)	4 (3)	280.31 (11162)	3 (5)	151.59 (4427)	2 (2)	675.49 (20142)	5 (6)
f_4	20.47 (519)	1 (1)	3278.67 (28743)	6 (5)	2987.69 (27288)	5 (3)	819.56 (27915))	3 (4)	478.1 (15118)	2 (2)	900.25 (28864)	4 (6)
f_5	9.98 (244)	1 (1)	2667.24 (22001)	6 (4)	2121.33 (19733)	5 (3)	813.14 (25761)	3 (5)	245.73 (7677)	2 (2)	942.46 (29677)	4 (6)
f_6	11.27 (276)	1 (1)	3113.87 (25075)	6 (4)	2136.9 (19808)	5 (3)	719.52 (26649)	3 (5)	266.36 (8988)	2 (2)	864.48 (28725)	4 (6)
f_7	48.38 (1205)	1 (1)	1766.65 (9835)	6 (6)	1250.83 (7566)	5 (4)	363.62 (9387)	4 (5)	97.58 (2489)	3 (3)	52.14 (1332)	2 (2)
f_8	57.03 (1456)	1 (1)	1638.42 (12978)	6 (4)	971.72 (10467)	5 (3)	516.03 (16410)	3 (5)	214.95 (6468)	2 (2)	709.02 (20712)	4 (6)
f_9	1025.65 (30000)	3 (3)	3198.19 (29863)	6 (2)	2355.1 (29625)	5 (1)	778.72 (30000)	1 (3)	853.46 (30000)	2 (3)	1074.35 (30000)	4 (3)
f_{10}	41.46 (1091)	1 (1)	1491.56 (8555)	6 (4)	647.87 (6837)	5 (3)	379.56 (10171)	3 (5)	173.61 (4991)	2 (2)	404.41 (11487)	4 (6)
f_{11}	45.72 (1174)	1 (1)	1086.02 (10573)	6 (4)	853.5 (8706)	5 (3)	474.1 (14122)	3 (5)	144.82 (3887)	2 (2)	748.96 (23201)	4 (6)
f_{12}	70.38 (1769)	1 (1)	1249.5 (14286)	5 (4)	1306.32 (15252)	6 (5)	239.33 (6536)	3 (3)	687.27 (27744)	4 (6)	203.16 (5454)	2 (2)
f_{13}	11.45 (285)	1 (1)	2577.08 (29500)	6 (4)	2538.16 (29678)	5 (5)	899.87 (29788)	3 (6)	336.02 (11585)	2 (2)	994.49 (28498)	4 (3)
f_{14}	12.91 (323)	1 (1)	2588.35 (29472)	6 (4)	2543.91 (29697)	5 (6)	955.39 (29575)	4 (5)	417.93 (15230)	2 (2)	953.51 (29286)	3 (3)
f_{15}	2.77 (70)	1 (1)	383.62 (3001)	5 (3)	633.16 (6001)	6 (6)	77.59 (2124)	2 (2)	172.34 (4862)	3 (5)	175.85 (4811)	4 (4)
f_{16}	347.17 (8980)	1 (1)	2438.82 (21243)	6 (2)	2325.47 (25803)	5 (5)	809.03 (23514)	4 (3)	723.57 (29944)	2 (6)	806.3 (25296)	3 (4)
f_{17}	89.96 (2305)	1 (1)	400.49 (3153)	6 (4)	318.77 (2677)	5 (2)	158.56 (4486)	3 (5)	98.91 (2922)	2 (3)	215.53 (5859)	4 (6)
f_{18}	16.57 (418)	1 (1)	3165.91 (23108)	6 (4)	2131.37 (20009)	5 (3)	833.26 (24005)	4 (5)	327.69 (12157)	2 (2)	726.3 (27452)	3 (6)
f_{19}	15.76 (386)	1 (1)	3095.71 (27337)	6 (4)	2259.14 (25014)	5 (3)	886.22 (29374)	4 (6)	269.46 (10005)	2 (2)	782.56 (28467)	3 (5)
f_{20}	15.4 (386)	1 (1)	3376.56 (28599)	6 (5)	2480.88 (28435)	5 (4)	905.79 (29014)	4 (6)	318.89 (12509)	2 (2)	753.72 (28202)	3 (3)
Averaged rank		1.1 (1.1)		5.9 (4)		5 (3.7)		3.15 (4.6)		2.2 (2.7)		3.65 (4.6)
Final rank		1 (1)		6 (4)		5 (3)		3 (5)		2 (2)		4 (5)

Table 11
Performances comparison of ADM, HEY-GA and CMA-ES in 100 runs.

Test functions	Dim. N	ADM			HEY-GA [21]			CMA-ES [19]		
		Mean (SD)	|Mean–f^*|	Rank	Mean (SD)	|Mean–f^*|	Rank	Mean (SD)	|Mean–f^*|	Rank
g_1	10	0.000E+00 (0.000E+00)	0.000E+00	1 (1)	0.000E+00 (0.000E+00)	0.000E+00	1 (1)	0.000E+00 (0.000E+00)	0.000E+00	1 (1)
	20	0.000E+00 (0.000E+00)	0.000E+00	1 (1)	0.000E+00 (0.000E+00)	0.000E+00	1 (1)	0.000E+00 (0.000E+00)	0.000E+00	1 (1)
	30	0.000E+00 (0.000E+00)	0.000E+00	1 (1)	0.000E+00 (0.000E+00)	0.000E+00	1 (1)	0.000E+00 (0.000E+00)	0.000E+00	1 (1)
g_2	10	0.000E+00 (0.000E+00)	0.000E+00	1 (1)	0.000E+00 (0.000E+00)	0.000E+00	1 (1)	1.206E+00 (8.443E+00)	1.206E+00	3 (3)
	20	0.000E+00 (0.000E+00)	0.000E+00	1 (1)	2.000E−02 (7.000E−03)	2.000E−02	2 (2)	7.600E−02 (2.350E−01)	7.600E−02	3 (3)
	30	0.000E+00 (0.000E+00)	0.000E+00	1 (1)	4.800E−02 (1.000E−02)	4.800E−02	2 (2)	1.514E+00 (8.026E+00)	1.514E+00	3 (3)
g_3	10	5.183E+00 (1.715E+01)	5.183E+00	3 (3)	8.800E−02 (8.770E−01)	8.800E−02	1 (1)	3.990E−01 (1.196E+00)	3.990E−01	2 (2)
	20	9.242E+00 (2.372E+01)	9.242E+00	3 (3)	1.158E+00 (4.459E+00)	1.158E+00	2 (2)	7.570E−01 (1.564E+00)	7.570E−01	1 (1)
	30	1.911E+01 (2.542E+01)	1.911E+01	3 (3)	3.018E+00 (8.640E+00)	3.018E+00	1 (1)	3.660E+00 (8.745E+00)	3.660E+00	2 (2)
g_4	10	0.000E+00 (0.000E+00)	0.000E+00	1 (1)	9.000E−03 (1.000E−02)	9.000E−03	2 (2)	2.100E−02 (1.100E−02)	2.100E−02	3 (3)
	20	0.000E+00 (0.000E+00)	0.000E+00	1 (1)	9.000E−03 (1.300E−02)	9.000E−03	2 (2)	4.300E−02 (1.700E−02)	4.300E−02	3 (3)
	30	0.000E+00 (0.000E+00)	0.000E+00	1 (1)	2.100E−02 (3.100E−02)	2.100E−02	2 (3)	6.200E−02 (1.400E−02)	6.200E−02	3 (2)
g_5	10	5.419E−02 (2.800E−02)	5.419E−02	2 (1)	1.600E−02 (9.000E−02)	1.600E−02	1 (2)	1.227E+00 (2.998E+00)	1.227E+00	3 (3)
	20	2.297E−02 (2.800E−02)	2.297E−02	3 (3)	0.000E+00 (0.000E+00)	0.000E+00	1 (1)	2.000E−03 (5.000E−03)	2.000E−03	2 (2)
	30	1.265E−02 (1.500E−02)	1.265E−02	3 (3)	0.000E+00 (0.000E+00)	0.000E+00	1 (1)	1.000E−03 (4.000E−03)	1.000E−03	2 (2)
g_6	10	0.000E+00 (0.000E+00)	0.000E+00	1 (1)	1.760E−01 (2.800E−02)	1.760E−01	2 (2)	8.931E+01 (2.282E+01)	8.931E+01	3 (3)
	20	4.079E−01 (1.100E+00)	4.079E−01	1 (2)	5.670E−01 (3.500E−02)	5.670E−01	2 (1)	1.724E+02 (3.434E+01)	1.724E+02	3 (3)
	30	3.136E+00 (4.536E+00)	3.136E+00	2 (2)	9.930E−01 (5.100E−02)	9.930E−01	1 (1)	2.536E+02 (4.335E+01)	2.536E+02	3 (3)
g_7	10	0.000E+00 (0.000E+00)	0.000E+00	1 (1)	0.000E+00 (0.000E+00)	0.000E+00	1 (1)	0.000E+00 (0.000E+00)	0.000E+00	1 (1)
	20	0.000E+00 (0.000E+00)	0.000E+00	1 (1)	1.342E+00 (7.120E−01)	1.342E+00	3 (3)	0.000E+00 (0.000E+00)	0.000E+00	1 (1)
	30	0.000E+00 (0.000E+00)	0.000E+00	1 (1)	1.729E+00 (2.000E−03)	1.729E+00	3 (3)	0.000E+00 (0.000E+00)	0.000E+00	1 (1)
g_8	2	1.031E+00 (0.000E+00)	0.000E+00	1 (1)	1.031E+00 (0.000E+00)	0.000E+00	1 (1)	9.390E−01 (2.560E−01)	9.200E−02	3 (3)
g_9	2	9.980E−01 (0.000E+00)	0.000E+00	1 (1)	1.000E+00 (0.000E+00)	2.000E−03	2 (1)	1.069E+01 (6.575E+00)	9.688E+00	3 (3)
Averaged rank				1.52 (1.52)			1.57 (1.57)			2.22 (2.17)
Final rank				1 (1)			2 (2)			3 (3)

30 or 100 independent runs set in different experiments, as shown in Table 4. In addition, to provide a fair assessment among the compared algorithms, the dimension of variables, the number of population size, and the maximum number of fitness evaluations of all the test algorithms are fixed within each experiment, as given in Table 4.

4.3. Performance comparisons of ADM-RCGA

In GA literature, the performance of a GA is usually measured on the basis of three criteria: accuracy, reliability, and efficiency of the algorithm [1]. Accuracy evaluates the degree of precision in locating the global maximum, reliability measures the level of scattering in obtained solutions, and efficiency assesses the rate of convergence. The present study uses a distance value |Mean–f^*| to describe the accuracy between the global optimum f^* and the mean of obtained solutions *Mean*. A better result is represented by a |Mean–f^*|, which is closer to zero. The standard deviation of obtained solutions is employed to compare the reliability of a GA. Smaller standard deviations indicate steadier and, consequently, more reliable ultimate solutions [21]. The average number of function evaluations and computer execution time are shown for the efficiency of a GA. Smaller average number of function evaluations and lower computer-execution time denote a more efficient GA.

4.3.1. Comparisons with five conventional mutation operators

To verify that the proposed mutation operator ADM is a generic improvement over conventional mutations existing in literature,

including RM, PLM, NUM, MNUM, and PM, the present study investigates ADM behavior across a wide range of well-recognized test functions as shown in Exp. 1 of Table 4, which summarizes the general problem characteristics indicative of real-world optimization [1].

The comparisons of average performance of mean fitness values and accuracy ranks achieved by 6 different mutation operators for benchmark functions I with 30 variables in 30 runs are listed in Table 8. It shows 16 cases in which ADM completely outperforms the 5 other conventional mutation operators, and the proposed mutation operator ADM has the best performance with final rank 1. More precisely, the distance value |Mean–f^*| between the global optimum and the mean of obtained solutions by ADM does not exceed 1.0 for all test functions, with the exception of three cases (test functions f_8, f_9, and f_{10}).

Table 8 also shows the corresponding comparisons of standard deviations of fitness values and reliability ranks in parentheses. In 16 out of 20 cases, ADM exhibits the best performances with final rank 1. According to Table 8, ADM is the most accurate and reliable algorithm alongside the other five conventional mutation operators. In addition, MNUM is slightly superior to PLM, and MNUM comes in second based on the averaged rank. On the other hand, NUM and PM have the worst performances with final ranks 5 and 6, respectively.

To illustrate the significance of the winning algorithm, Table 9 also lists the p-value of a t-test (in terms of %) and the t-test result for benchmark functions I with 30 variables in 30 runs achieved by ADM against those obtained from other five conventional mutation operators. By using the two-tailed t-test with a 58 degree of freedom at a 5% level of significance, the t-test result is presented as "+", "−", or "∼" [1]. A "∼" sign indicates that there is no significant difference in the mean fitness values found by both compared algorithms, and a "+" or "−" sign indicates that the mean fitness values obtained by ADM are significantly better or worse than the mean fitness values achieved by other algorithms.

Clearly, the statistics displayed in Table 9 demonstrate that ADM, with more than 95% confidence, significantly outperforms RM and PLM in 16 out of 20 test cases. The proposed ADM also beats MNUM in 9, NUM in 15, and PM in 18 out of 20 test cases, respectively. It can be observed that the comparisons of ADM with the other five conventional mutation operators are statistically significant for most of the test cases in benchmark functions I.

The average execution time, average number of function evaluations (=average number of generations), and efficiency ranks achieved by six different mutation operators for benchmark functions I with 30 variables in 30 runs are shown in Table 10. From the results of Table 10, we find that ADM has both the least execution time and the lowest number of function evaluations (as shown in parentheses) in 19 out of 20 cases in comparison with the other 5 conventional mutation operators. In addition, MNUM has the second-best performance in terms of average execution time and averaged fitness evaluations. On the contrary, RM has the worst performance, with final rank 6, in terms of average function evaluations. From Tables 8–10, we can conclude that ADM provides greater accuracy, more reliability, and higher efficiency than the other five conventional mutation operators.

4.3.2. Comparisons with CMA-ES and HYK-GA

The present study is also interested in comparing the proposed ADM-RCGA method with some other leading adaptive evolutionary algorithms, besides the conventional mutation operators. We chose state-of-the-art adaptive mutation methods CMA-ES [19] and HKY-GA [21]—two of the strongest rivals. CMA-ES uses a clever learning method to adapt the full covariance matrix of a normal mutation distribution. The core idea of this method is to gather information about successful search steps, and to use that information to deterministically modify the covariance matrix of the mutation distribution. On the other hand, HKY-GA employs one of the latest Markov models of nucleotide substitution rates—the HKY evolution model—to apply to a RCGA. The HKY-GA method not only aims to adapt the mutation rates, but also tries to determine the types of genes to be subjected to mutation. To provide a fair comparison between HEY-GA, CMA-ES, and the proposed ADM-RCGA, the simulation conditions (same as [21]) are given in Exp. 2 of Table 4.

Table 12
Performances comparison of ADM, HEY-GA and CMA-ES using t-test resultsin 100 runs.

Test functions	Dim. N	ADM–HEY-GA Result(t-test)	ADM–CMA-ES Result(t-test)
g_1	10	∼	∼
	20	∼	∼
	30	∼	∼
g_2	10	∼	∼
	20	+	+
	30	+	∼
g_3	10	−	−
	20	−	−
	30	−	−
g_4	10	+	+
	20	+	+
	30	+	+
g_5	10	−	+
	20	−	−
	30	−	−
g_6	10	+	+
	20	∼	+
	30	−	+
g_7	10	∼	∼
	20	+	∼
	30	+	∼
g_8	2	∼	+
g_9	2	∼	+

Comparisons of the average performance of mean fitness values and accuracy ranks achieved by ADM-RCGA, HEY-GA, and CMA-ES for benchmark functions II in 100 runs are listed in Table 11. It shows that the proposed ADM-RCGA has the best performance, with final rank 1, and that in 16 out of 23 cases ADM-RCGA performs with rank 1. More precisely, the distance value |Mean–f^*| between the global optimum and the mean of obtained solutions by ADM-RCGA does not exceed 1.0 for all test functions, with the exception of four cases (test functions g_3 and g_6). On the opposite side, HEY-GA and CMA-ES perform with rank 1 in 12 and 7 out of 23 cases, respectively. The distance value |Mean–f^*| exceeds 1.0 for four cases (test functions g_3 and g_7) in HEY-GA, and eight cases (test functions g_2, g_3, g_5, g_6, and g_9) in CMA-ES.

Table 11 also shows the corresponding comparisons of standard deviations of fitness values and reliability ranks in parentheses. In 16 out of 23 cases, ADM-RCGA has the best performances, with final rank 1. In other words, there are 13 and 7 out of 23 cases in rank 1 for HEY-GA and CMA-EA, respectively. Table 11 indicates that ADM-RCGA is the most accurate and reliable algorithm, better than HEY-GA and CMA-ES. In addition, HEY-GA has competitive performance than CMA-GA while CMA-GA has the worst performance, with final rank 3.

By applying the two-tailed t-test with a 198 degree of freedom at a 5% level of significance, the statistics shown in Table 12 demonstrate that ADM-RCGA, with more than 95% confidence, significantly outperforms HEY-GA and CMA-ES in eight and ten test cases, respectively. On the opposite side, the mean fitness values by ADM-RCGA are significantly poorer than HEY-GA and CMA-ES

Table 13
Performances comparison of ADM and LX-PM for 30 variablesin 30 runs.

Test functions	ADM			LX-PM [1]			ADM–LX-PM
	Mean (SD)	|Mean–f^*|	Rank	Mean (SD)	|Mean–f^*|	Rank	Result(t-test)
f_1	3.225E−01 (2.518E−02)	3.225E−01	2 (2)	1.010E−10 (1.040E−10)	1.010E−10	1 (1)	–
f_2	3.000E+00 (0.000E+00)	0.000E+00	1 (1)	3.000E+00 (0.000E+00)	0.000E+00	1 (1)	∼
f_3	1.000E+00 (5.160E−16)	0.000E+00	1 (1)	1.000E+00 (1.730E−08)	0.000E+00	1 (2)	∼
f_4	8.532E−03 (1.035E−02)	8.532E−03	2 (2)	1.940E−03 (1.570E−03)	1.940E−03	1 (2)	–
f_5	1.971E−32 (4.947E−33)	1.971E−32	1 (1)	3.200E−19 (2.890E−19)	3.200E−19	2 (2)	+
f_6	2.426E−32 (5.612E−33)	2.426E−32	1 (1)	7.950E−23 (9.920E−23)	7.950E−23	2 (2)	+
f_7	9.979E+05 (9.688E−10)	2.402E−04	1 (1)	9.980E+05 (5.590E+00)	1.325E+02	2 (2)	+
f_8	1.224E+01 (3.487E+00)	1.224E+01	2 (2)	0.000E+00 (0.000E+00)	0.000E+00	1 (1)	–
f_9	4.760E+01 (2.834E+01)	4.760E+01	2 (2)	1.580E+01 (2.150E+00)	1.580E+01	1 (1)	–
f_{10}	1.203E+04 (4.024E+02)	5.369E+02	2 (2)	1.260E+04 (1.850E−12)	3.051E+01	1 (1)	–
f_{11}	3.500E+00 (5.220E−15)	0.000E+00	1 (1)	3.500E+00 (2.980E−08)	0.000E+00	1 (2)	∼
f_{12}	4.193E−22 (6.280E−22)	4.193E−22	1 (1)	1.950E−20 (6.830E−20)	1.950E−20	2 (2)	∼
f_{13}	0.000E+00 (0.000E+00)	0.000E+00	1 (1)	4.750E−11 (3.340E−11)	4.750E−11	2 (2)	+
f_{14}	0.000E+00 (0.000E+00)	0.000E+00	1 (1)	2.820E−12 (2.090E−12)	2.820E−12	2 (2)	+
f_{15}	0.000E+00 (0.000E+00)	0.000E+00	1 (1)	3.030E−08 (1.310E−08)	3.030E−08	2 (2)	+
f_{16}	3.618E−04 (4.507E−04)	3.618E−04	2 (2)	3.600E−05 (1.700E−04)	3.600E−05	1 (2)	–
f_{17}	3.410E−04 (1.165E−04)	3.410E−04	2 (1)	2.320E−04 (1.950E−04)	2.320E−04	1 (2)	–
f_{18}	0.000E+00 (0.000E+00)	0.000E+00	1 (1)	9.260E−11 (1.030E−10)	9.260E−11	2 (2)	+
f_{19}	1.872E−32 (4.362E−E−33)	1.872E−32	1 (1)	9.480E−32 (1.700E−31)	9.480E−32	2 (2)	+
f_{20}	2.508E−32 (7.954E−33)	2.508E−32	1 (1)	6.070E−31 (2.240E−30)	6.070E−31	2 (2)	∼
Averaged rank			1.35 (1.3)			1.5 (1.75)	
Final rank			1 (1)			2 (2)	

Table 14
Performances comparison of ADM, IEA, BOA and OGA for 10 variables in 30 runs.

Test functions	ADM			IEA [28]			BOA [29]			OGA [30]		
	Mean	|Mean–f^*|	Rank	Mean	|Mean–f^*|	Rank	Mean	|Mean–f^*|	Rank	Mean	|Mean–f^*|	Rank
h_1	3.898E−16	3.898E−16	1	5.400E−02	5.400E−02	3	1.900E−02	1.900E−02	2	7.200E−02	7.200E−02	4
h_2	1.675E+01	3.255E+00	1	1.532E+01	4.680E+00	2	1.229E+01	7.710E+00	4	1.524E+01	4.760E+00	3
h_3	0.000E+00	0.000E+00	1	5.130E+00	5.130E+00	3	7.700E−01	7.700E−01	2	5.300E+00	5.300E+00	4
h_4	1.066E−15	1.066E−15	1	1.542E+01	1.542E+01	3	5.320E+00	5.320E+00	2	1.762E+01	1.762E+01	4
h_5	0.000E+00	0.000E+00	1	3.000E−04	3.000E−04	2	7.700E−03	7.700E−03	4	3.000E−04	3.000E−04	2
h_6	1.850E+01	2.738E−03	1	1.460E+01	3.900E+00	3	1.810E+01	4.000E−01	2	1.401E+01	4.490E+00	4
h_7	1.216E+01	2.175E−05	1	1.212E+01	4.380E−02	3	1.215E+01	8.800E−03	2	1.211E+01	5.080E−02	4
h_8	1.471E−14	1.471E−14	1	1.000E+00	1.000E+00	3	9.300E−01	9.300E−01	2	1.700E+00	1.700E+00	4
h_9	3.978E+01	3.978E+01	2	6.674E+02	6.674E+02	4	8.400E+00	8.400E+00	1	5.842E+02	5.842E+02	3
h_{10}	2.632E+00	2.632E+00	1	1.164E+02	1.164E+02	4	8.930E+00	8.930E+00	2	9.457E+01	9.457E+01	3
h_{11}	0.000E+00	0.000E+00	1	3.380E−01	3.380E−01	2	1.007E+01	1.007E+01	4	9.000E−01	9.000E−01	3
h_{12}	8.624E−02	8.624E−02	1	9.990E−01	9.990E−01	2	1.008E+00	1.008E+00	4	1.002E+00	1.002E+00	3
Averaged rank			1.08			2.83			2.58			3.42
Final rank			1			3			2			4

Table 15
Performances comparison of ADM, IEA and OGA for 100 variables in 30 runs.

Test functions	ADM			IEA [28]			OGA [30]		
	Mean	|Mean–f^*|	Rank	Mean	|Mean–f^*|	Rank	Mean	|Mean–f^*|	Rank
h_1	1.490E−05	1.490E−05	1	6.500E−01	6.500E−01	2	1.630E+00	1.630E+00	3
h_2	1.697E+02	3.031E+01	1	1.532E+02	4.685E+01	2	1.397E+02	6.029E+01	3
h_3	3.333E−02	3.333E−02	1	6.210E+02	6.210E+02	2	6.107E+03	6.107E+03	3
h_4	7.988E+01	7.988E+01	1	2.135E+02	2.135E+02	2	3.675E+02	3.675E+02	3
h_5	0.000E+00	0.000E+00	1	1.600E+00	1.600E+00	2	1.496E+01	1.496E+01	3
h_6	1.325E+02	5.249E+01	1	1.313E+02	5.369E+01	2	1.155E+02	6.952E+01	3
h_7	1.216E+02	2.174E−04	1	1.204E+02	1.158E+00	2	1.167E+02	4.888E+00	3
h_8	3.239E+00	3.239E+00	1	3.690E+00	3.690E+00	2	9.470E+00	9.470E+00	3
h_9	1.463E+04	1.463E+04	3	8.011E+03	8.011E+03	1	1.229E+04	1.229E+04	2
h_{10}	1.007E+02	1.007E+02	1	2.081E+03	2.081E+03	2	5.282E+03	5.282E+03	3
h_{11}	5.647E+01	5.647E+01	2	4.394E+01	4.394E+01	1	6.519E+01	6.519E+01	3
h_{12}	2.629E−03	2.629E−03	1	3.286E+01	3.286E+01	2	4.825E+01	4.825E+01	3
Averaged rank			1.25			1.83			2.92
Final rank			1			2			3

in seven and five cases, respectively. It can be observed that the proposed ADM-RCGA is statistically significant and slightly superior to HEY-GA and CMA-ES. From Tables 11 and 12, we can conclude that ADM-RCGA provides more accuracy, greater reliability, and higher efficiency than HEY-GA and CMA-ES.

4.3.3. Comparison with LX-PM

Deep and Thakur [1] introduced a new mutation operator called PM for RCGAs and compared six generational real-coded GAs on a set of 20 benchmark global optimization test problems. Their results showed that the RCGA using the PM in conjunction with Laplace crossover (LX-PM) outperforms all other five GAs. To provide a fair comparison between LX-PM and the proposed ADM-RCGA, the simulation conditions (same as [1]) are given in Exp. 3 of Table 4.

The comparisons of average performance of mean fitness values and accuracy ranks achieved by ADM-RCGA and LX-PM for benchmark functions I in 30 runs are listed in Table 13. It shows that the proposed ADM-RCGA has the best performance, with final rank 1, and in 13 out of 20 cases ADM-RCGA performs with rank 1. On the contrary, LX-PM performs with rank 1 in 10 out of 20 cases.

Table 13 also shows the corresponding comparisons of standard deviations of fitness values and reliability ranks in parentheses. The results in Table 13 demonstrate that ADM-RCGA has the best performances, with final rank 1, and ranks first in 14 out of 20 cases. In addition, there are only 5 out of 20 cases in rank 1 for LX-PM.

By using the two-tailed t-test with a 58 degree of freedom at a 5% level of significance, the statistics also displayed in Table 13 demonstrate that ADM-RCGA, with more than 95% confidence, significantly outperforms LX-PM in eight test cases. On the opposite side, the mean fitness values by ADM-RCGA are significantly poorer than LX-PM in seven cases. The proposed ADM-RCGA is statistically significant and slightly superior to LX-PM. From Table 13, we can conclude that ADM-RCGA provides more accuracy, greater reliability, and higher efficiency than LX-PM.

4.3.4. Comparison with BOA, IEA, and OGA

Ho et al. [28] suggested an intelligent evolutionary algorithm (IEA) based on orthogonal experimental design for solving large parameter optimization problems. They compared some existing EAs on a set of 12 benchmark global optimization test problems. Their results showed that BOA [29] has the best performance for 10 variables by comparing with 6 other EAs, and IEA outperforms all other 5 EAs for 100 variables. To provide a fair comparison among IEA, BOA, and OGA [30], and the proposed ADM-RCGA, the simulation conditions (same as [28]) are given in Exp. 4 of Table 4.

The comparisons of average performance of mean fitness values and accuracy ranks achieved by ADM-RCGA, IEA, BOA, and OGA for benchmark functions III with 10 variables in 30 runs are listed in Table 14. From this table, it is observed that the proposed ADM-RCGA has the best performance with final rank 1, and there are 11 out of 12 cases in which ADM-RCGA performs with rank 1. On the contrary, BOA performs with rank 1 only in a single case and there is no case with rank 1 for IEA.

Table 15 shows the comparisons of average performance of mean fitness values and accuracy ranks achieved by ADM-RCGA, IEA, and OGA for benchmark functions III with 100 variables in 30 runs. Due to the long computation time, BOA is tested on the 12 test functions with only 10 variables [28]. The results of Table 15 show that ADM-RCGA has the best performances, with final rank 1 and first rank in 10 out of 12 cases. In addition, there are only 2 out of 12 cases in rank 1 for IEA and no case for OGA. From Tables 14 and 15, we can conclude that ADM-RCGA provides more accuracy, greater reliability, and higher efficiency in achieved results than IEA, BOA, and OGA.

5. Conclusion

The current study has presented a new mutation operator to ADM that focuses on simplicity, robustness, and efficiency within the context of RCGAs. To evaluate the performance of the proposed algorithm, we conducted a series of experiments on a set of 41 well-known real-valued benchmark global optimization test functions.

When compared with five conventional mutation operators, including RM, PLM, NUM, MNUM, and PM, the proposed ADM approach shows a significant improvement in the quality of the global optimum solution found under the same simulation conditions.

The present study also compared the performance of the proposed ADM-RCGA with that of six leading evolutionary algorithms, besides the conventional mutation operators. Against the state-of-the-art adaptive evolutionary methods, HYK-GA and CMA-ES, the proposed ADM-RCGA shows superior performance. Furthermore, ADM-RCGA also outperforms all other GAs, including BOA, IEA, LX-PM, and OGA. Finally, we can conclude that the proposed ADM-RCGA provides more accuracy, greater reliability, and higher efficiency than all the other GAs considered in the present study.

The experiment outcome for the proposed ADM-RCGA is excellent in most cases, but it still performed worse in some functions due to increasing the risks of local optima traps. As our future perspective, we plan to further improve the evolutionary efficiency by integrating the approach of design of experiment with the proposed ADM-RCGA algorithm.

Acknowledgments

A portion of this work was supported by the National Science Council, ROC, under grant no. NSC-98-2211-E-040-011. The council's support is greatly appreciated.

References

[1] K. Deep, M. Thakur, A new mutation operator for real coded genetic algorithms, Applied Mathematics and Computation 193 (2007) 211–230.
[2] C. Heitzunger, S. Selberherr, An extensible TCAD optimization framework combing gradient based and genetic optimizers, Microelectronics Journal 33 (2002) 61–68.
[3] T. Back, H.-P. Schwefel, An overview of evolutionary algorithms for parameter optimization, Evolutionary Computation 1 (1) (1993) 1–23.
[4] D.E. Goldberg, Genetic Algorithm in Search, Optimization and Machine Learning, Addison-Wesley, Boston, 1989.
[5] D.A. Coley, An Introduction to Genetic Algorithms for Scientists and Engineers, World Scientific, London, 1998.
[6] D.E. Goldberg, Real-coded genetic algorithms, virtual alphabets, and blocking, Complex Systems 5 (2) (1991) 139–168.
[7] S.H. Ling, F.H.F. Leung, An improved genetic algorithm with average-bound crossover and wavelet mutation operations, Soft Computing 11 (2007) 7–31.
[8] C.Z. Janikow, Z. Michalewicz, An experimental comparison of binary and floating point representation in genetic algorithms, in: Proceedings of the Fourth International Conference on Genetic Algorithms, Morgan Kaufmann, San Francisco, 1991, pp. 31–36.
[9] P.H. Tang, M.H. Tseng, Medical data mining using BGA and RGA for weighting of features in fuzzy *k*-NN classification, IEEE the International Conference on Machine Learning and Cybernetics 5 (2009) 3070–3075.
[10] I. Korejo, S. Yang, C. Li, A directed mutation operator for real coded genetic algorithms, in: C. Di Chio, et al. (Eds.), EvoApplications, Part I, LNCS 6024, 2010, pp. 491–500.
[11] D.K. Pratihar, Soft Computing, Alpha Science International Ltd., Oxford, UK, 2008.
[12] D. Bhandari, N.R. Pal, S.K. Pal, Directed mutation in genetic algorithms, Information Sciences 79 (1994) 251–270.
[13] Q. Zhou, Y. Li, Directed variation in evolutionary strategies, IEEE Transactions on Evolutionary Computation 7 (4) (2003) 356–366.
[14] A. Berry, P. Vamplew, PoD can mutate: a simple dynamic directed mutation approach for genetic algorithms, in: Proceedings of International Conference on Artificial Intelligence in Science and Technology, 2004, pp. 200–205.
[15] L. Temby, P. Vamplew, A. Berry, Accelerating real valued genetic algorithms using mutation-with-momentum, in: S. Zhang, R.A. Jarvis (Eds.), AI 2005. LNCS (LNAI), 3809, 2005, pp. 1108–1111.
[16] M. Srinivas, L.M. Patnaik, Adaptive probabilities of crossover and mutation in genetic algorithms, IEEE Transaction on System, Man, and Cybernetics 24 (4) (1994) 17–26.
[17] M.S. Chen, F.H. Liao, Adaptive mutation operators and its applications, Journal of Dayeh University 7 (1) (1998) 91–101.
[18] M.H. Tseng, H.C. Liao, The genetic algorithm for breast tumor diagnosis – the case of DNA viruses, Applied Soft Computing 9 (2009) 703–710.
[19] N. Hansen, A. Ostermeier, Completely derandomized self-adaptation in evolutionary strategies, IEEE Transactions on Evolutionary Computation 9 (2) (2001) 159–195.
[20] F. Lobo, C. Lima, Z. Michalewicz, Parameter setting in evolutionary algorithms, Studies in Computational Intelligence 54 (2007) 47–76.
[21] F. Vafaee, P.C. Nelson, A genetic algorithm that incorporates an adaptive mutation based on an evolutionary model, in: 2009 International Conference on Machine Learning and Applications, 2009, pp. 101–107.
[22] J. Baker, Reducing bias and inefficiency in the selection algorithm, in: Proceedings of the First International Conference on Genetic Algorithms, 1985, pp. 14–21.
[23] L.J. Eshelman, J.D. Schaffer, Real-coded genetic algorithms and interval-schemata, Found Genet Algorithms 2 (1993) 187–202.
[24] A.M.S. Zalzala, P.J. Fleming, Genetic algorithms in engineering systems, in: IET, 1997.
[25] Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs, Springer-Verlag, New York, 1992.
[26] C.R. Houck, J.A. Joines, M.G. Kay, A Genetic Algorithm for Function Optimization: A Matlab Implementation, Technical Report, North Carolina State University, Raleigh, NC, 1996.
[27] K. Deb, M. Goyal, A combined genetic adaptive search (GeneAS) for engineering design, Computer Sciences and Informatics 26 (4) (1996) 30–45.
[28] S.Y. Ho, L.S. Shu, J.-H. Chen, Intelligent evolutionary algorithms for large parameter optimization problems, IEEE Transactions on Evolutionary Computation 8 (6) (2004) 522–541.
[29] M. Pelikan, D.E. Goldberg, E. Cantu-Paz, BOA: the Bayesian optimization algorithm, in: Procedings of the 1st Conference GECCO-99, 1999, pp. 525–532.
[30] Q. Zhang, Y.W. Leung, An orthogonal genetic algorithm for multimedia multicast routine, IEEE Transactions on Evolutionary Computation 3 (1999) 53–62.