A Fast and Efficient Stochastic Opposition-Based Learning for Differential Evolution in Numerical Optimization

Tae Jong Choi, Julian Togelius, Yun-Gyung Cheong

Abstract

A new variant of stochastic opposition-based learning (OBL) is proposed in this paper. OBL is a relatively new machine learning concept, which consists of simultaneously calculating an original solution and its opposite to accelerate the convergence of soft computing algorithms. Recently a new opposition-based differential evolution (ODE) variant called BetaCODE was proposed as a combination of differential evolution and a new stochastic OBL variant called BetaCOBL. BetaCOBL is capable of flexibly adjusting the probability density functions used to calculate opposite solutions, generating more diverse opposite solutions, and preventing the waste of fitness evaluations. While it has shown outstanding performance compared to several state-of-the-art OBL variants, BetaCOBL is challenging with more complex problems because of its high computational cost. Besides, as it assumes that the decision variables are independent, there is a limitation in the search for decent opposite solutions on inseparable problems. In this paper, we propose an improved stochastic OBL variant that mitigates all the limitations of BetaCOBL. The proposed algorithm called iBetaCOBL reduces the computational cost from $O(NP^2 \cdot D)$ to $O(NP \cdot D)$ ($NP$ and $D$ stand for population size and dimension, respectively) using a linear time diversity measure. In addition, iBetaCOBL preserves the strongly dependent decision variables that are adjacent to each other using the multiple exponential crossover. The results of the performance evaluations on a set of 58 test functions show that iBetaCODE finds more accurate solutions than ten state-of-the-art ODE variants including BetaCODE. Additionally, we applied iBetaCOBL to two state-of-the-art DE variants, and as in the previous results, iBetaCOBL based variants exhibit significantly improved performance.

Keywords: Beta Distribution, Differential Evolution, Numerical Optimization, Opposition-Based Learning

1. Introduction

Evolutionary algorithms (EAs) are nature-inspired and population-based metaheuristic optimization algorithms. As EAs do not make any assumptions on a given problem, they can be applied to black-box optimizations and usually yield satisfactory approximate solutions. Generally, EAs randomly initialize individuals over the search space of a given problem and update them repeatedly through evolutionary operators such as selection, crossover, and mutation until one of the termination criteria is satisfied.

Differential evolution (DE) is one of the most popular EAs for optimizing multidimensional real-valued functions. Compared to other EAs, DE is simple, offers a straightforward implementation and contains fewer control parameters requiring tuning. Moreover, DE has shown outstanding performance in many competitions on numerical optimization. Furthermore, in contrast with the covariance matrix adaptation evolution strategy (CMA-ES), which is another powerful EA for optimizing multidimensional real-valued functions, DE can be applied to large-scale problems by virtue of its low space complexity. These features have gathered attention from researchers and practitioners for over two decades.

Since DE was introduced, many studies have been conducted to design new DE variants to improve performance. One of the successful branches within these studies is the combination of DE and opposition-based learning (OBL). Inspired by the concept of opposite relationships among objects, OBL is a relatively new machine learning concept, which simultaneously calculates an original solution and its opposite to accelerate convergence. Despite its simplicity, OBL has successful led to improvements on many soft computing algorithms such as artificial neural networks, EAs, fuzzy logic, and reinforcement learning. The pioneering work on the combination of DE and OBL was conducted by Rahnamayan et al., resulting in a technique called opposition-based DE (ODE). Compared with a classical DE, the ODE has two additional operators, both calculating an opposite population based on an original and merging them into one and selecting the fittest individuals as population size.

A stochastic DE-based OBL called BetaCODE was proposed recently: the approach uses the beta distribution along with the partial dimensional change and selection switching schemes. Compared with other ODE variants, BetaCODE has three advantages. First, it can control the degree of opposition by using the convex and concave density functions adjusted by the beta distribution. Second, it is able to prevent the waste in fitness evaluation by using the selection switching scheme. Fi-
nally, it is capable of preserving useful information held by the original solutions using the partial dimensional change scheme. Based on these strengths, BetaCODE has become one of the most powerful ODE variants [14]. However, BetaCODE suffers from a high computational cost and is ineffective in handling dependent variables. Because of these limitations, it is difficult to use BetaCODE to optimize more complex problems.

In this paper, we propose an improved version of BetaCODE called iBetaCODE that mitigates all BetaCODE limitations. Instead of using a power mean-based diversity measure [15] [16] in the selection switching scheme, we employed a linear time diversity measure [17] [18] [19] [20] [21] [22] [23] [24] [25] to drastically reduce computational cost. We found that, regarding the diversity measure, replacing the power mean by linear time maintains the performance of BetaCODE with considerably less time complexity. In addition, we employed the multiple exponential crossover [26] in the partial dimensional change scheme, which can preserve strongly dependent decision variables that are adjacent to each other. We conducted experiments to evaluate the performance of iBetaCODE on Congress of Evolutionary Computation (CEC) 2013 and 2017 test suites [27] [28] and compared it to ten state-of-the-art ODE variants including BetaCODE. The experimental results showed that iBetaCODE significantly outperforms all comparative ODE variants. Notably, compared to its predecessor BetaCODE, iBetaCODE is competitive with considerably less time complexity. In addition to the classical DE-based experiments, we tested the compatibility of iBetaCOBL with two state-of-the-art DE variants, CoDE [29] and SHADE [30], to measure performance enhancement. As in previous experiments, the results showed that the iBetaCOBL-based variants outperform the BetaCOBL-based and the original ones.

The main contributions of this paper are as follows.

1. A new ODE variant called iBetaCODE is proposed, which is competitive with ten popular ODE variants.
2. iBetaCODE significantly outperforms its predecessor BetaCODE with considerably less time complexity.
3. iBetaCOBL can be readily embedded into any DE variant as a module.

The remainder of the paper is organized as follows: We introduce the basics of classical DE and OBL in Section 2. In Section 3, we describe related work on OBL in EAs, especially for development. In Section 4, we review BetaCOBL, which is the basis of the proposed algorithm and illustrate the details of the proposed algorithm. We explain the experimental setup in Section 5. We present the experimental results in Sections 6 and 7. Finally, we conclude this paper in Section 8.

2. Background

2.1. Differential Evolution

DE [1] [2] is one of the most popular EAs for optimizing multidimensional real-valued functions; it involves having a population of NP individuals. Each individual is a D-dimensional vector denoted \( x_g = (x_{1g}, x_{2g}, \ldots, x_{Dg}) \) where \( g \) stands for generation. At the beginning of the optimization process, DE randomly distributes the population over the search space for a given problem. These individuals then explore the search space through simple mathematical formulas. If an individual finds a new location with a better fitness value, the individual moves the new location; otherwise, it stays in its current place. This process is repeated until one of the termination criteria is satisfied. DE consists of four operators: 1) initialization, 2) mutation, 3) crossover, and 4) selection. We briefly introduce these operators in the following subsections.

2.1.1. Initialization

The role of the initialization operator is to randomly distribute the population to cover the search space. Let the minimum and maximum bounds of the search space \( x_{min} = (x_{1min}, x_{2min}, \ldots, x_{Dmin}) \) and \( x_{max} = (x_{1max}, x_{2max}, \ldots, x_{Dmax}) \), respectively. Each individual is initialized according to

\[
x_{i0}^j = x_{min}^j + rand_{i,j} \cdot (x_{max}^j - x_{min}^j)
\]

where \( rand_{i,j} \) stands for a uniformly distributed random number within the [0, 1] range.

2.1.2. Mutation

The role of the mutation operator is to generate a set of mutant vectors. Each mutant vector \( v_{i,g} \) is generated by using a linear combination of three donor vectors, \( x_{r1,g}, x_{r2,g}, \) and \( x_{r3,g} \). These donor vectors are randomly selected from the population, which are mutually exclusive and distinct from each target vector \( x_{i,g} \). Each mutant vector is formed according to

\[
v_{i,g} = x_{r1,g} + F \cdot (x_{r2,g} - x_{r3,g})
\]

where \( F \) stands for the scaling factor that controls the scale of the difference \( (x_{r2,g} - x_{r3,g}) \). The mutation operator mentioned above is one of the classical mutation strategies. In addition, numerous advanced mutation strategies have been introduced.

2.1.3. Crossover

The role of the crossover operator is to generate a set of trial vectors by recombining each target vector and its corresponding mutant vector. An element \( f_{rand} \in \{1, 2, \ldots, D\} \) is selected randomly, which ensures that a trial vector consists of at least one element of its corresponding mutant vector. Each trial vector \( u_{i,g} \) is formed according to

\[
u_{i,g} = \begin{cases} v_{i,g}^j & \text{if } rand_{i,j} \leq CR \text{ or } j = f_{rand} \\ x_{i,g}^j & \text{otherwise} \end{cases}
\]

where \( CR \) stands for the crossover rate that controls the ratio of elements between a target vector and its corresponding mutant vector. The crossover operator mentioned above is the binomial crossover, which is the most frequently used.

2.1.4. Selection

The selection operator picks the individuals at \( g + 1 \) by comparing the fitness value of each target vector and its corresponding trial vector. If the trial vector has a better fitness value, the
trial vector is selected, and the target vector is discarded; otherwise, vice versa. Each individual $x_{i,g+1}$ is formed according to

$$
x_{i,g+1} = \begin{cases} 
  u_{i,g} & \text{if } f(u_{i,g}) \leq f(x_{i,g}) \\
  x_{i,g} & \text{otherwise.}
\end{cases}
$$

(4)

where $f(x)$ stands for the objective function to be minimized.

2.1.5. Advanced Differential Evolution Variants

Since DE was introduced, numerous studies have been conducted to design new DE variants in an effort to improve performance, including adaptive trial vector generation strategy schemes, adaptive and self-adaptive parameter control schemes, ensemble DE variants, and incorporating external techniques such as r-stable distribution based mutations, neighborhood-based mutations, and OBLs. For more detailed explanations of the state-of-the-art DE variants, please refer to the following surveys.

2.2. Opposition-Based Learning

Inspired by the concept of opposite relationships among objects, Tizhoosh [8] proposed a computational opposition concept named OBL, which simultaneously calculates an original solution and its opposite. Despite its simplicity, OBL has proven to be effective in improving many soft computing algorithms such as artificial neural networks, EAs, fuzzy logic, and reinforcement learning. In addition, it was mathematically proved that opposite values are more likely to be located near the optimal solution of a given problem than pure random values.

An opposite solution in the one-dimensional space can be defined as follows.

**Definition 1 [8]:** Let $x \in [x_{\text{min}}, x_{\text{max}}]$ be an original solution. The opposite solution for $x$ denoted by $\hat{x}$ is obtained as follows:

$$
\hat{x} = x_{\text{min}} + x_{\text{max}} - x
$$

(5)

Similarly, an opposite solution in the $D$-dimensional space can be defined as follows.

**Definition 2 [8]:** Let $x = (x^1, x^2, \ldots, x^D)$, $x^j \in [x^j_{\text{min}}, x^j_{\text{max}}]$ be an original solution. The opposite solution for $x$ denoted by $\hat{x} = (\hat{x}^1, \hat{x}^2, \ldots, \hat{x}^D)$ is obtained as follows:

$$
\hat{x}^j = x^j_{\text{min}} + x^j_{\text{max}} - x^j
$$

(6)

This is the type-I opposition in which opposite solutions are calculated in the search space of a given problem. For type-II opposition, the calculation occurs in the objective space of a given problem, which can be defined as follows.

**Definition 3 [9]:** Let $f(x)$ be an objective function where $y_{\text{min}} \leq f(x) \leq y_{\text{max}}$ and let $x = (x^1, x^2, \ldots, x^D)$, $x^j \in [x^j_{\text{min}}, x^j_{\text{max}}]$ be an original solution. The opposite solution for $x$ denoted by $\hat{x} = (\hat{x}^1, \hat{x}^2, \ldots, \hat{x}^D)$ is obtained as follows:

$$
\hat{x} = \{ c \mid \hat{y} = y_{\text{min}} + y_{\text{max}} - f(x) \}
$$

(7)

It should be noted that calculating type-II opposition requires prior knowledge of the objective space of a given problem. Therefore, it is difficult to apply type-II opposition to black-box optimization problems. Finally, OBL can be defined as follows:

**Definition 4 [8]:** Let $x = (x^1, x^2, \ldots, x^D)$ be an original solution and let $\hat{x} = (\hat{x}^1, \hat{x}^2, \ldots, \hat{x}^D)$ be the opposite solution for $x$. OBL selects the opposite solution if $f(\hat{x}) \leq f(x)$ and vice versa otherwise.

3. Related Work

After the original OBL was introduced, numerous improved OBL variants were proposed. In this section, we describe several state-of-the-art OBL variants.

As DE researchers and practitioners have actively used OBL variants [11], many improved OBL variants have been proposed in the form of ODE variants. In the pioneering work of [13], a combination of DE and OBL called ODE was proposed to accelerate the convergence of DE. To automatically tune the jumping rate of OBL, Rahnamayan et al. proposed an ODE variant called ODE with a time-varying jumping rate (ODETVJR) and found that a linearly decreasing jumping rate is more effective than a linearly increasing one [61]. To prevent the waste of fitness evaluations, Esmaizadeh and Rahnamayan proposed a quasi-ODE (QODE) variant that prevents the use of ODE with r-stable distribution based mutations, which stops OBL when the success rate of the opposite solutions decreases monotonously from a predefined threshold [62]. In [63], a quasi-ODE (QODE) was proposed, using quasi opposite solutions that were mathematically proved to be more likely to be located near the optimal solution of a given problem than opposite ones. Egeze et al. proposed a quasi-reflection OBL (QROBL) and combined it with biogeography-based optimization [64]. In contrast to the quasi OBL [65] that searches for opposite solutions between the center point and a given original solution, QROBL searches opposite solutions between the center point and a solution that is the full opposite of the originally given one. In [65], a current optimum-based ODE (COODE) was proposed, using the location of the current best individual as a reference point for calculating opposite solutions. In [66], a generalized OBL (GOBL) was proposed to accelerate convergence. The combination of DE and GOBL, named GODE, uses 1) a dynamic search space reflecting the current population, 2) a uniformly generated random reference point, and 3) a uniform distribution to calculate opposite solutions. Zhou et al. proposed an extension of GODE called elite ODE (EOODE), which only applies GOBL to the current elite individuals [67]. Liu et al. proposed another extension...
of GODE called adaptive GODE (AGODE), which automatically tunes the jumping rate of OBL based on the success rate of opposite solutions [68].

For more detailed explanations of OBL variants in other fields such as artificial neural networks, fuzzy logic, and reinforcement learning, please refer to the following surveys [10][11][12].

4. Proposed Algorithm

The proposed algorithm, namely iBetaCOBL, is introduced in this section. The details of the modified schemes will be discussed after first reviewing BetaCOBL [14], which is the basis of the proposed algorithm.

4.1. Review of BetaCOBL

![Figure 1: Example of concave and convex opposite points](image)

The following drawbacks affect numerous OBL variants: 1) As OBL variants compute opposite solutions or based on the uniform distribution, there is an inherent limitation in the deterministic search for decent opposite solutions. In other words, there is an opportunity for improvement when computing opposite solutions by using useful probability distributions such as Cauchy, Gaussian, and α-stable ones. 2) When OBL variants compute opposite solutions, the useful elements held with the original solutions can be discarded as all of the elements of the original solutions are transformed into opposites. 3) As OBL variants follow a greedy strategy, fitness evaluations can be wasted if suitable opposite solutions can no longer be discovered at the end of the optimization process.

To overcome these limitations, BetaCOBL uses the following techniques: 1) Beta distribution: BetaCOBL calculates concave or convex opposite solutions by using the beta distribution, which can create various shapes for the continuous probability density functions (PDFs) within the range [0, 1]. Here, a concave opposite solution represents a solution generated based on a PDF where the opposite point for a given original solution is selected with the highest probability. Conversely, a convex opposite solution is generated based on a PDF where the point for a given original solution is selected with the lowest probability.

As a result, with the concave and convex OBLs, BetaCOBL can find appropriate opposite solutions faster than other OBL variants. Fig. 1 shows an example of concave and convex opposite points.

2) Partial dimensional change scheme: BetaCOBL uses the binomial crossover in DE to calculate a partial opposite solution, formed by the recombination of an original solution and its complete opposite solution. Therefore, BetaCOBL can obtain more diverse opposite solutions than other OBL variants as it can have one of the 2N possible opposite solutions with a given pair of original and complete opposite solutions. In addition, as it uses the binomial crossover, BetaCOBL can preserve the useful elements held by original solutions.

3) Selection switching scheme: In general, OBL helps discover promising regions at the beginning of the optimization process, but it becomes less effective as the optimization process progresses; as a result, fitness evaluations are potentially wasted. To mitigate this issue, BetaCOBL estimates the population diversity before the concave and convex OBLs. If the population diversity is higher than a predefined threshold DT, BetaCOBL uses a (µ+1) selection with all the original solutions of the population; otherwise, it uses a (µ, λ) selection with the worst half original solutions of the population. Consequently, BetaCOBL can prevent the waste of fitness evaluations by applying one of the two selection operators depending on the convergence progress.

A concave opposite solution is calculated using the beta distribution with both α and β greater than one, as follows:

\[
x_{\text{concave}}^j = (x_{\text{max}}^j - x_{\text{min}}^j) \cdot \text{Beta}(\alpha, \beta) + x_{\text{min}}^j
\]

\[
\alpha = \begin{cases} 
\text{spread} \cdot \text{peak} & \text{if mode} < 0.5 \\
\text{spread} & \text{otherwise}
\end{cases}
\]

\[
\beta = \begin{cases} 
\text{spread} \cdot \text{peak} & \text{if mode} < 0.5 \\
\text{spread} & \text{otherwise}
\end{cases}
\]

\[
\text{spread} = \left(\frac{1}{\sqrt{\text{normDiv}}}\right)^{1+\text{N}(0,0.5)}
\]

\[
\text{peak} = \begin{cases} 
\frac{(\text{spread} - 2) \cdot \text{mode} + 1}{\text{spread} \cdot \text{mode}} & \text{if mode} < 0.5 \\
\frac{2 \cdot \text{spread} - \text{spread} \cdot \text{mode}}{\text{spread} \cdot \text{mode} + \text{spread} - \text{mode}} & \text{otherwise}
\end{cases}
\]

\[
\text{mode} = \frac{x_{\text{max}}^j + x_{\text{min}}^j - x_{\text{concave}}^j - x_{\text{min}}^j}{x_{\text{max}}^j - x_{\text{min}}^j}
\]

where Beta(α, β) and N(0, 0.5) denote the beta distribution with parameters α and β, and the Gaussian distribution with the mean 0 and variance 0.5, respectively. In addition, the normalized diversity denoted by normDiv is calculated as follows:

\[
\text{normDiv} = \frac{1}{N P} \sum_{i=1}^{NP} CD(x_{i,g}, P_g)
\]

\[
CD(x_{i,g}, P_g) = \min_{e \in P_g, e \neq x_{i,g}} d(e, x_{i,g})
\]

4
\[ d(c, x_{i,g}) = \sqrt{\frac{1}{D} \sum_{j=1}^{D} \left( \frac{x^j_{i,g} - c^j}{x^\text{max}_i - x^\text{min}_i} \right)^2} \] (16)

The same formulas calculate a convex opposite solution except for the mode and spread, calculated as follows:

\[
\text{mode} = \frac{x^\text{max}_i - x^\text{min}_i}{\frac{1}{D} \sum_{j=1}^{D} (x^j_{i,g} - x^\text{min}_i)}
\] (17)

\[
\text{spread} = 0.1 \cdot \sqrt{\text{normDiv} + 0.9}
\] (18)

4.2. Modified Selection Switching Scheme

4.2.1. Problem of Selection Switching Scheme

BetaCOBL uses the selection switching scheme to prevent the waste of fitness evaluations, which applies one of the two selection operators depending on the population diversity. To estimate the population diversity, BetaCOBL calculates the average of the minimum distance between all possible pairs, which is a power mean-based diversity measure. A generalized definition of the power mean-based diversity measure is presented in [15] [16], where it is defined as the mapping \( D_k : \mathbb{R}^{NP \times D} \rightarrow \mathbb{R} \)

\[
D_k(P_g, a, b) = \sqrt{\frac{1}{NP} \sum_{i=1}^{NP} d_i^a}
\] (19)

\[
d_i^a = \frac{1}{NP-1} \sum_{j=1}^{NP} \| x^j_{i,g} - x_{j,b} \|^a
\] (20)

where \( a, b \neq 0 \). The two parameters \( a \) and \( b \) determine the behavior of the diversity measure. If \( a = 1 \), the arithmetic mean distance of all possible pairs is computed. If \( a = 0 \), the geometric mean distance of all possible pairs is computed. In addition, if \( a = -\infty \), the diversity measure evaluates the minimum distance of all possible pairs. Finally, the lower the value of \( a \) and \( b \), the larger the penalty to the collocation of individuals. The power mean-based diversity measure with \( a = -\infty \) and \( b = 1 \) that BetaCOBL uses was experimentally proven not to be \((\rho, \epsilon)\)-ectropic where both \( \rho \) and \( \epsilon \) can simultaneously take values close to zero [69], which means it can discourage the collocation of individuals.

However, the power mean-based diversity measure with \( a = -\infty \) and \( b = 1 \) incurs a \( O(NP^2 \cdot D) \) computational cost; as a result it is difficult to use BetaCOBL for optimizing more complex problems with a large population size.

4.2.2. Applying Linear Time Diversity Measure

To reduce the computational cost, we replaced the power mean-based diversity measure with a linear time diversity measure in the selection switching scheme. Of the two well-known measures, we employed one that computes the arithmetic mean of the Euclidean distances of all possible pairs [17] [18] [19] [20] [21] [22] [23] [24] [25], where it can be defined as the mapping \( D_d : \mathbb{R}^{NP \times D} \rightarrow \mathbb{R} \)

\[
D_d(P_g) = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \| x_{i,g} - x_{j,b} \|\] (21)

A naive implementation for equation (21) incurs a \( O(NP^2 \cdot D) \) computational cost. Wineberg and Oppacher [24] [25] reformulated the equation for a linear time diversity measure as follows:

\[
D'_d(P_g) = \frac{1}{D} \sqrt{\sum_{k=1}^{D} (\mathbf{x}_k^g)^2 - (\bar{x}_g)^2}
\] (22)

where \( (\mathbf{x}_k^g)^2 = \frac{1}{NP} \sum_{i=1}^{NP} (x_{i,g}^k)^2 \) and \( (\bar{x}_g)^2 = \frac{1}{NP} \sum_{i=1}^{NP} x_{i,g}^k \). The computational cost of the reformulated diversity measure is \( O(NP \cdot D) \). Note that the proposed algorithm uses the normalized version of the diversity measure, obtained by dividing \((\mathbf{x}_k^g)^2 - (\bar{x}_g)^2\) by \((x_{\text{max}}^k - x_{\text{min}}^k)^2\) in the equation (22).

4.2.3. Rationale of Employing Linear Time Diversity Measure

As mentioned in Section 4.2.1, BetaCOBL uses the power mean-based diversity measure to check the convergence progress, which leads to a high computational cost. Therefore, we must replace it with a fast diversity measure to apply BetaCOBL to more complex problems with a large population size.

There are two linear time diversity measures in the multidimensional continuous space. The first measure was discussed in Section 4.2.2 and the other computes the arithmetic mean of the Euclidean distances of every point to the center [70] [71] [72] [73] [74] [22] [23], where it can be defined as the mapping \( D_c : \mathbb{R}^{NP \times D} \rightarrow \mathbb{R} \)

\[
D_c(P_g) = \sum_{i=1}^{n} \| x_{i,g} - \bar{x}_g \|
\] (23)

where \( \bar{x}_g = (M^1, M^2, \ldots, M^D) \) and the centroid of the population with \( M^k = \frac{1}{NP} \sum_{i=1}^{NP} x_{i,g}^k \) \( k = 1, 2, \ldots, D \). Fig. 2 shows the two linear time diversity measures.

We chose the first measure as it was theoretically proven to discourage the collocation of individuals bigger than the second measure [69]. Let the population size for each measure be \( NP = 2^m + p \). The ectropic property of the first measure \( D_d \) is \((\frac{2^m}{2^m + p}, 0)\), while that of the second measure \( D_c \) is \((\frac{1}{2^m + p}, 0)\). Therefore, in a situation where many individuals are in overlapping positions, the second measure is more likely to return a higher value than the first one. In other words, BetaCOBL with
the second measure is likely to continue to use the \((\lambda + \mu)\) selection instead of \((\lambda, \mu)\) at the end of the optimization process, which may not prevent the waste of fitness evaluations.

Consequently, the proposed algorithm can estimate the population diversity faster than BetaCOBL with the replacement. In addition, we analyze the relative performance of the original BetaCODE and BetaCODE with the linear time diversity measure and found that there were no significant differences, as reported in Section 6.

4.3. Modified Partial Dimensional Change Scheme

4.3.1. Problem of Partial Dimensional Change Scheme

BetaCOBL uses the binomial crossover in the partial dimensional change scheme to calculate partial opposite solutions. The binomial crossover is the most frequently used crossover operator in DE literature, and has the following properties [74, 26]. First, the relationship between the mutation probability \(\mu\) and the control parameter \(CR\) [75] is calculated and copied from the complete opposite solution as follows. First, an element \(n \in [1, D]\) is selected randomly. Then, the four constants, \(E_m = T \cdot CR\), \(E_s = T \cdot (1 - CR)\), \(CR_m = E_m / E_{m+1}\), and \(CR_s = E_s / E_{s+1}\), are initialized where \(E_m\) and \(E_s\) stand for the approximate size of each component copied from the complete opposite solution and the target vector, respectively. Here, the length of the exchanged component \(T\) is initialized at ten, as in [26]. Following this, the multiple exponential crossover calculates a partial opposite solution as follows:

1. Starting from the element \(n\), a component of Bernoulli trials with \(CR_m\) is calculated and copied from the complete opposite solution.
2. Starting from the last failure element, the next component of Bernoulli trials with \(CR_s\) is calculated and copied from the target vector.
3. Repeat from Step 1 until all of the elements are decided.

The pseudo code of the multiple exponential crossover is presented in Algorithm 1.

```
Algorithm 1: Multiple Exponential Crossover

Input: Target vector \(x_{i,g}\), mutant vector \(v_{i,g}\), crossover rate \(CR\), and length of exchanged components \(T\)
Output: Trial vector \(u_{i,g}\)
1 Select random integer \(n\) within the range \([1, D]\);
2 \(E_m = T \cdot CR\), \(E_s = T \cdot (1 - CR)\);
3 \(CR_m = E_m / E_{m+1}\), \(CR_s = E_s / E_{s+1}\);
4 \(L = 1\), Mutation_Enable = 1;
5 repeat

If Mutation_Enable == 1 then

6 repeat
7 \(u_{i,g}^{(n+L-1)} = v_{i,g}^{(n+L-1)}\);
8 \(L = L + 1\);
9 until \(L \leq D\) and \(rand_{i,j} \leq CR_m\);
10 Mutation_Enable = 0;
else

11 repeat
12 \(u_{i,g}^{(n+L-1)} = x_{i,g}^{(n+L-1)}\);
13 \(L = L + 1\);
14 until \(L \leq D\) and \(rand_{i,j} \leq CR_s\);
15 Mutation_Enable = 1;

end

16 until \(L \leq D\);
```

4.3.2. Applying Multiple Exponential Crossover

To improve the performance on inseparable problems, we employed the multiple exponential crossover [26] in the partial dimensional change scheme. The multiple exponential crossover is a semi-consecutive crossover operator that divides a trial vector into several components, and each component is a copy of the component at the location of either the target or the mutant vector [25]. Therefore, the multiple exponential crossover is the same as the exponential crossover that is being repeated. Fig. 3 shows the behavior of the binomial, exponential, and multiple exponential crossovers. In the proposed algorithm, the multiple exponential crossover calculates a partial opposite solution with a given pair of target vectors and complete opposite solution as follows. First, an element \(n \in [1, D]\) is selected randomly. Then, the four constants, \(E_m = T \cdot CR\), \(E_s = T \cdot (1 - CR)\), \(CR_m = E_m / E_{m+1}\), and \(CR_s = E_s / E_{s+1}\), are initialized where \(E_m\) and \(E_s\) stand for the approximate size of each component copied from the complete opposite solution and the target vector, respectively. Here, the length of the exchanged component \(T\) is initialized at ten, as in [26]. Following this, the multiple exponential crossover calculates a partial opposite solution as follows:

1. Starting from the element \(n\), a component of Bernoulli trials with \(CR_m\) is calculated and copied from the complete opposite solution.
2. Starting from the last failure element, the next component of Bernoulli trials with \(CR_s\) is calculated and copied from the target vector.
3. Repeat from Step 1 until all of the elements are decided.

The pseudo code of the multiple exponential crossover is presented in Algorithm 1.

### Algorithm 1: Multiple Exponential Crossover

**Input**: Target vector \(x_{i,g}\), mutant vector \(v_{i,g}\), crossover rate \(CR\), and length of exchanged components \(T\)

**Output**: Trial vector \(u_{i,g}\)

1. Select random integer \(n\) within the range \([1, D]\);
2. \(E_m = T \cdot CR\), \(E_s = T \cdot (1 - CR)\);
3. \(CR_m = E_m / E_{m+1}\), \(CR_s = E_s / E_{s+1}\);
4. \(L = 1\), Mutation_Enable = 1;
5. repeat

   If Mutation_Enable == 1 then

6.   repeat
7.     \(u_{i,g}^{(n+L-1)} = v_{i,g}^{(n+L-1)}\);
8.     \(L = L + 1\);
9.   until \(L \leq D\) and \(rand_{i,j} \leq CR_m\);
10.    Mutation_Enable = 0;

   else

11.   repeat
12.     \(u_{i,g}^{(n+L-1)} = x_{i,g}^{(n+L-1)}\);
13.     \(L = L + 1\);
14.   until \(L \leq D\) and \(rand_{i,j} \leq CR_s\);
15.    Mutation_Enable = 1;

16. end

17. until \(L \leq D\);

4.3.3. Rationale of Employing Multiple Exponential Crossover

As mentioned in Section 4.3.1, it is of critical importance to preserve strongly dependent decision variables on inseparable
problems when searching for satisfactory solutions. However, the exponential crossover is not an alternative to the binomial crossover as tuning the control parameter CR is difficult and it cannot generate all of the possible partial opposite solutions. Using a covariance matrix helps identify the interactions between variables, but it leads to a high computational cost. Therefore, we employed the multiple exponential crossover, which has the strengths of the exponential crossover but also retains the properties of the binomial crossover. With the replacement, the proposed algorithm can achieve better performance than BetaCOBL on inseparable problems by preserving the strongly dependent decision variables that are adjacent to each other.

4.4. iBetaCODE

iBetaCODE is the combination of DE and iBetaCOBL. As with other ODE variants, iBetaCOBL is executed in the initialization and iteration phases of iBetaCODE. In the initialization phase, iBetaCODE executes iBetaCOBL with the initialized individuals. In the iteration phase, iBetaCODE executes iBetaCOBL or the evolutionary operators of DE alternatively according to a predefined jumping rate \( J_r \). If a random number generated according to the uniform distribution is lower than or equal to the jumping rate, iBetaCODE performs iBetaCOBL. Otherwise, iBetaCODE executes the evolutionary operators. Regarding the jumping rate, we set \( J_r = 0.05 \) in all the experiments in this paper, as in [14]. The entire pseudo code of iBetaCODE is presented in Algorithm 2.

5. Experimental Setup

5.1. Test Functions

We used a set of 58 test functions from two well-known suites, the CEC 2013 and 2017 real-valued benchmark problems, for demonstrating the performance of the proposed algorithm. In the CEC 2013 test suite, there are five unimodal functions \( F_1-F_5 \), fifteen simple multimodal functions \( F_6-F_{20} \), and eight composition functions \( F_{21}-F_{28} \). In the CEC 2017 test suite, there are three unimodal functions \( F_1-F_3 \), seven simple multimodal functions \( F_4-F_{10} \), ten expanded multimodal functions \( F_{11}-F_{20} \), and ten hybrid composition functions \( F_{21}-F_{30} \). For more detail explanations of the CEC 2013 and 2017 test suites, please refer to the following technical reports [27, 28].

The settings for the tests including the maximum number of function evaluations, search range, global optimum, and termination criteria for each test function are all initialized in the same way as in [27] and [28].

5.2. Performance Metrics

To assess the performance of the test algorithms, we utilized the following performance metrics.

Algorithm 2: iBetaCODE

```plaintext
Input : Objective function \( f(x) \), upper bound \( x_{\text{max}} \), lower bound \( x_{\text{min}} \), maximum number of function evaluations \( NFEs \), scale factor \( F \), crossover rate \( CR \), population size \( NP \), diversity threshold \( DT \), and jumping rate, \( J_r \).

Output: Best objective value \( f(x_{\text{best}}) \)

/* Initialization phase */
1 for \( i = 0; i < NP; i = i + 1 \) do
2 for \( j = 0; j < D; j = j + 1 \) do
3 \( x^j_{i,0} = x^j_{\text{min}} + \text{rand}^j_i \cdot (x^j_{\text{max}} - x^j_{\text{min}}); \)
4 end
5 end
6 \( NFEs = NP \cdot g = 1; \)
7 Calculate normDiv using equation (22);
8 if \( \text{normDiv} > DT \) then
9 \( (\mu + \lambda) \) selection phase of iBetaCOBL (Algorithm 3);
10 else
11 \( (\mu, \lambda) \) selection phase of iBetaCOBL (Algorithm 4);
12 end
13 /* Iteration phase */
14 while None of termination criteria is satisfied do
15 if \( \text{rand} \leq J_r \) then
16 Calculate normDiv using equation (22);
17 if \( \text{normDiv} > DT \) then
18 \( (\mu + \lambda) \) selection phase of iBetaCOBL (Algorithm 3);
19 else
20 \( (\mu, \lambda) \) selection phase of iBetaCOBL (Algorithm 4);
21 end
22 else
23 Select random three donor vectors \( x_{r_1,g}, x_{r_2,g}, x_{r_3,g} \) where \( r_1 \neq r_2 \neq r_3 \neq i; \)
24 Select random integer \( j_{\text{rand}} \) within the range \( [1, D]; \)
25 for \( j = 0; j < D; j = j + 1 \) do
26 if \( \text{rand}^j \leq CR \) or \( j = j_{\text{rand}} \) then
27 \( u^j_i = x^j_{i,g} + F \cdot (x^j_{r_2,g} - x^j_{r_3,g}); \)
28 else
29 \( u^j_i = x^j_{i,g}; \)
30 end
31 end
32 for \( i = 0; i < NP; i = i + 1 \) do
33 if \( f(u_{i,g}) \leq f(x_{i,g}) \) then
34 \( x_{i,g+1} = u_{i,g}; \)
35 else
36 \( x_{i,g+1} = x_{i,g}; \)
37 end
38 end
39 \( NFEs = NFEs + NP; \)
40 end
41 \( g = g + 1; \)
42 end
```
Algorithm 3: (μ + λ) Selection Phase of iBetaCOBL

Input: Population \( P^g \)
Output: Population \( P'^g \)

/* (μ + λ) selection */
1 Set opposite population \( OP_x = (\tilde{x}_{1,g}, \tilde{x}_{2,g}, \ldots, \tilde{x}_{NP_x,g}) \);
2 for \( i = 0; i < NP; i = i + 1 \) do
3 \hspace{1em} if \( rand_i \leq 0.5 \) then
4 \hspace{2em} Calculate spread using equation (11);
5 \hspace{2em} for \( j = 0; j < D; j = j + 1 \) do
6 \hspace{3em} Calculate mode using equation (13);
7 \hspace{3em} Calculate peak using equation (12);
8 \hspace{3em} Calculate alpha using equation (9);
9 \hspace{3em} Calculate beta using equation (10);
10 \hspace{3em} \( t^j_{i,g} = (x^j_{\text{max}} - x^j_{\text{min}}) \cdot \text{Beta}(\alpha, \beta) + x^j_{\text{min}} \);
11 \hspace{2em} end
12 \hspace{1em} end
13 \hspace{1em} else
14 \hspace{2em} Calculate spread using equation (18);
15 \hspace{2em} for \( j = 0; j < D; j = j + 1 \) do
16 \hspace{3em} Calculate mode using equation (17);
17 \hspace{3em} Calculate peak using equation (12);
18 \hspace{3em} Calculate alpha using equation (9);
19 \hspace{3em} Calculate beta using equation (10);
20 \hspace{3em} \( t^j_{i,g} = (x^j_{\text{max}} - x^j_{\text{min}}) \cdot \text{Beta}(\alpha, \beta) + x^j_{\text{min}} \);
21 \hspace{2em} end
22 \hspace{1em} end
23 \hspace{1em} Calculate a partial opposite solution \( \bar{x}_{i,g} \) using Algorithm \( \Box \) with \( t^j_{g}, x_{i,g} \), and \( CR = 0.1 \);
24 \hspace{1em} Calculate a partial opposite solution \( \bar{x}_{i NP,g} \) using Algorithm \( \Box \) with \( t^j_{g}, x_{i,g} \), and \( CR = 0.9 \);
25 \hspace{1em} end
26 Merge original and opposite populations \( P^g + OP^g \);
27 Select NP best individuals \( P'^g \) from merged population \( P^g + OP^g \);
28 \( NFES = NFES + (NP \cdot 2) \);

Algorithm 4: (μ, λ) Selection Phase of iBetaCOBL

Input: Population \( P^g \)
Output: Population \( P'^g \)

/* (μ, λ) selection */
1 Sort population \( P^g \);
2 for \( i = \frac{NP}{2}; i < NP; i = i + 1 \) do
3 \hspace{1em} if \( rand_i \leq 0.5 \) then
4 \hspace{2em} Calculate spread using equation (11);
5 \hspace{2em} for \( j = 0; j < D; j = j + 1 \) do
6 \hspace{3em} Calculate mode using equation (13);
7 \hspace{3em} Calculate peak using equation (12);
8 \hspace{3em} Calculate alpha using equation (9);
9 \hspace{3em} Calculate beta using equation (10);
10 \hspace{3em} \( t^j_{i,g} = (x^j_{\text{max}} - x^j_{\text{min}}) \cdot \text{Beta}(\alpha, \beta) + x^j_{\text{min}} \);
11 \hspace{2em} end
12 \hspace{1em} end
13 \hspace{1em} else
14 \hspace{2em} Calculate spread using equation (18);
15 \hspace{2em} for \( j = 0; j < D; j = j + 1 \) do
16 \hspace{3em} Calculate mode using equation (17);
17 \hspace{3em} Calculate peak using equation (12);
18 \hspace{3em} Calculate alpha using equation (9);
19 \hspace{3em} Calculate beta using equation (10);
20 \hspace{3em} \( t^j_{i,g} = (x^j_{\text{max}} - x^j_{\text{min}}) \cdot \text{Beta}(\alpha, \beta) + x^j_{\text{min}} \);
21 \hspace{2em} end
22 \hspace{1em} end
23 \hspace{1em} Calculate a partial opposite solution \( \bar{x}_{i,g} \) using Algorithm \( \Box \) with \( t^j_{g}, x_{i,g} \), and \( CR = 0.1 \);
24 \hspace{1em} Calculate a partial opposite solution \( \bar{x}_{i NP,g} \) using Algorithm \( \Box \) with \( t^j_{g}, x_{i,g} \), and \( CR = 0.9 \);
25 \hspace{1em} if \( f(\bar{x}_{i,g}) \leq f(\tilde{x}_{2,g}) \) then
26 \hspace{2em} if \( f(\bar{x}_{i,g}) \leq f(x_{i,g}) \) then
27 \hspace{3em} \( x_{i,g} = \bar{x}_{i,g} \);
28 \hspace{2em} end
29 \hspace{1em} end
30 \hspace{1em} else
31 \hspace{2em} if \( f(\tilde{x}_{2,g}) \leq f(x_{i,g}) \) then
32 \hspace{3em} \( x_{i,g} = \tilde{x}_{2,g} \);
33 \hspace{2em} end
34 \hspace{1em} end
35 \hspace{1em} end
36 \hspace{1em} end
37 \hspace{1em} NFES = NFES + NP;
5.2.1. Function Error Value

We utilized the function error value (FEV) to assess the accuracy of a test algorithm. This metric can be defined as follows.

\[
FEV = f(x_{\text{best,gen}}) - f(x_i)
\]

where \( f(x) \) denotes the objective function to minimize. In addition, \( x_{\text{best,gen}} \) is the best solution for the test algorithm in the last generation, and \( x_i \) is the global optimum of a given problem. The lower the value of FEV, the higher the accuracy of the test algorithm.

5.2.2. Statistical Test

To determine whether the difference in performance for two test algorithms is significant or not, we utilized the Wilcoxon signed-rank test with an \( \alpha = 0.05 \) significance level [76]. The symbols in the tables of this paper all have the following meaning unless stated otherwise.

1. \+: The outperformance of the proposed algorithm over the benchmark is significant based on the Wilcoxon signed-rank test.
2. \=\: The performance difference between the proposed algorithm and the benchmark is not statistically significant based on the Wilcoxon signed-rank test.
3. \(-\): The underperformance of the proposed algorithm relative to the benchmark is significant based on the Wilcoxon signed-rank test.

In addition, we utilized the Friedman test with Hochbergs post hoc analysis to determine whether the difference in performance for multiple test algorithms was significant or not [76].

6. Results and Comparisons

6.1. Comparison with Ten ODE Variants

We conducted experiments to evaluate the performance of iBetaCODE and compared it to ten state-of-the-art ODE variants, namely: 1) ODE [13], 2) ODETVJR [61], 3) ODEPGJ [62], 4) QODE [63], 5) QRODE [64], 6) COODE [65], 7) GODE [66], 8) EODE [67], 9) AGODE [68], and 10) BetaCODE [14]. For a fair comparison, we used the same classical DE variant DE/rand/1/bin; regarding the control parameters associated with the DE, we used the following values: \( F = 0.5 \), \( CR = 0.9 \), and \( NP = 100 \). Additionally, we used the values recommended by the authors of each paper for the remaining control parameters.

6.1.1. Performance Evaluation on CEC 2013 Test Suite

In this subsection, the performance evaluation results on the CEC 2013 test suite are presented. Table 1 shows the mean and standard deviation of the FEVs for each algorithm for 30 dimensions, collected through 51 independent runs. As we can see from the table, the proposed algorithm has a clear edge over all the others. Compared with COODE, ODE, ODETVJR, QODE, and QRODE, iBetaCODE found more significantly accurate solutions on more than half of the test functions. In particular, iBetaCODE outperformed COODE and QRODE considerably on approximately four-fifths of the test functions. The second and third best algorithms are BetaCODE and ODEPGJ. In addition, Table 2 shows the Friedman test with Hochbergs post hoc analysis, which supports the experimental results in Table 1 where iBetaCODE ranked first among the test algorithms and the outperformance over COODE, ODE, ODETVJR, QODE, and QRODE was statistically significant. As a result, the proposed algorithm is superior to the ten ODE variants.

Additionally, we analyzed the performance evaluation results in Table 1 based on the attributes of the test functions. The proposed algorithm achieved a similar or slightly weaker optimization performance comparatively on the unimodal functions \( (F_1-F_5) \). However, it achieved a significantly better optimization performance in solving the multimodal \( (F_6-F_{20}) \) and composition functions \( (F_{21}-F_{28}) \). These results revealed that the algorithm proposed has a strong exploration property, and is thus capable of discovering more satisfactory solutions comparatively for more complex test functions.

We found similar tendencies for fifty dimensions in Tables 3 and 4. Compared with the experimental results for thirty dimensions, the outperformance of the proposed algorithm is slightly larger for fifty dimensions. For example, iBetaCODE found more significantly accurate solutions on more than half the test functions compared with all ten ODE variants except AGODE. In particular, iBetaCODE outperformed COODE, QODE, and QRODE considerably on approximately four-fifths of the test functions. Therefore, the proposed algorithm demonstrates clearly that it can achieve better searchability than all of the compared ones including its predecessor BetaCODE, particularly in the optimization of the multimodal and composition functions of the CEC 2013 test suite for both thirty and fifty dimensions.

6.1.2. Performance Evaluation on CEC 2017 Test Suite

In this subsection, the performance evaluation results on the CEC 2017 test suite are presented. Table 5 shows the mean and standard deviation of the FEVs for each algorithm for thirty dimensions, collected through 51 independent runs. As we can see from the table, the proposed algorithm has a clear edge over all the others, which found more significantly accurate solutions on more than half of the test functions. In particular, iBetaCODE outperformed QRODE considerably on all the test functions. The second and third best algorithms are BetaCODE and QODE. In addition, Table 6 shows the Friedman test with Hochbergs post hoc analysis, which supports the experimental results in Table 5 where iBetaCODE ranked first among the test algorithms with an outperformance that was statistically significant over all the compared algorithms except BetaCODE. Therefore, we can conclude that the proposed algorithm is superior to the ten ODE variants.

Additionally, we analyzed the performance evaluation results in Table 5 based on the attributes of the test functions. The proposed algorithm showed similar or slightly better optimization performance than the compared algorithms on the unimodal functions \( (F_1-F_5) \). However, it showed significantly better optimization performance in solving the multimodal \( (F_6-F_{10}) \), ex-
| Algorithm | Average ranking | p-value | p-value (Hochberg) | Sig. | Test statistics |
|-----------|----------------|---------|-------------------|------|----------------|
| iBetaCODE | 3.66 | | | | Chi Square = 58.78 |
| AGODE | 5.52 | -2.07E+00 | 3.80E-02 | 1.82E-01 | No |
| BasODE | 3.96 | | | | Yes |
| COOGE | 7.86 | -4.71E+00 | 2.43E-06 | 2.19E-05 | Yes |
| EDE | 5.95 | -2.94E+00 | 1.11E-02 | 8.57E-02 | No |
| GODE | 5.16 | -1.87E+00 | 9.45E-02 | 2.84E-01 | Yes |
| ODE | 7.16 | -3.91E+00 | 8.59E-05 | 6.84E-04 | Yes |
| ODEPG | 4.86 | -1.31E+00 | 1.84E-01 | 3.87E-01 | No |
| ODEVR | 9.31 | 3.52E+00 | 2.06E-04 | 1.80E-03 | Yes |
| QDE | 6.88 | -5.31E+00 | 7.13E-04 | 4.28E-03 | Yes |
| QODE | 8.59 | -5.20E+00 | 2.02E-07 | 2.02E-06 | Yes |

Table 2: Friedman test with Hochberg’s post hoc for iBetaCODE and ten ODE variants on CEC 2013 test suite at 30-D.
### Table 3: Mean and standard deviation of FEVs for iBetaCODE and ten ODE variants on CEC 2013 test suite at 50-D.

| Algorithm | iBetaCODE | ODEFPG | ODEVBR | QODE |
|-----------|-----------|--------|--------|------|
| MEAN (STD DEV) | MEAN (STD DEV) | MEAN (STD DEV) | MEAN (STD DEV) |
| F1 | 8.55E-01 (3.94E-03) | 4.43E-03 (6.35E-04) | 4.34E-04 (9.34E-05) | 4.34E-04 (9.34E-05) |
| F2 | 4.71E-01 (3.94E-03) | 3.10E-01 (7.93E-05) | 4.51E-01 (1.82E-06) | 5.08E-01 (1.51E-06) |
| F3 | 5.77E-01 (4.16E-02) | 1.05E-01 (2.32E-03) | 2.28E-01 (4.13E-03) | 2.28E-01 (4.13E-03) |
| F4 | 3.10E-01 (7.39E-03) | 1.80E-01 (2.85E-03) | 2.20E-01 (4.13E-03) | 2.20E-01 (4.13E-03) |
| F5 | 1.14E-01 (7.39E-03) | 1.14E-01 (7.39E-03) | 2.13E-01 (1.52E-03) | 2.13E-01 (1.52E-03) |
| F6 | 4.37E-01 (4.50E-03) | 4.37E-01 (4.50E-03) | 3.85E-01 (5.18E-03) | 3.85E-01 (5.18E-03) |
| F7 | 1.60E-01 (4.50E-03) | 1.60E-01 (4.50E-03) | 2.82E-01 (5.18E-03) | 2.82E-01 (5.18E-03) |
| F8 | 2.10E-01 (4.50E-03) | 2.10E-01 (4.50E-03) | 2.12E-01 (4.50E-03) | 2.12E-01 (4.50E-03) |
| F9 | 2.93E-01 (3.77E-03) | 3.92E-02 (2.17E-03) | 2.93E-01 (3.90E-03) | 2.93E-01 (3.90E-03) |
| F10 | 3.72E-01 (1.37E-02) | 3.40E-02 (2.70E-02) | 3.40E-02 (2.70E-02) | 3.40E-02 (2.70E-02) |
| F11 | 2.68E-01 (3.78E-03) | 1.90E-01 (2.41E-03) | 2.54E-01 (4.90E-03) | 2.54E-01 (4.90E-03) |
| F12 | 3.66E-01 (1.38E-02) | 3.54E-02 (1.52E-03) | 3.64E-02 (1.27E-03) | 3.64E-02 (1.27E-03) |
| F13 | 3.66E-01 (1.38E-02) | 3.54E-02 (1.52E-03) | 3.64E-02 (1.27E-03) | 3.64E-02 (1.27E-03) |
| F14 | 1.33E-01 (4.17E-03) | 1.34E-01 (5.80E-03) | 1.28E-01 (4.99E-03) | 1.28E-01 (4.99E-03) |
| F15 | 3.38E-01 (2.28E-02) | 3.24E-02 (2.07E-02) | 3.38E-02 (1.99E-02) | 3.38E-02 (1.99E-02) |
| F16 | 3.66E-01 (2.28E-02) | 3.24E-02 (2.07E-02) | 3.38E-02 (1.99E-02) | 3.38E-02 (1.99E-02) |
| F17 | 3.66E-01 (2.28E-02) | 3.24E-02 (2.07E-02) | 3.38E-02 (1.99E-02) | 3.38E-02 (1.99E-02) |
| F18 | 4.60E-01 (2.28E-04) | 6.00E-02 (2.07E-04) | 4.60E-02 (2.07E-04) | 4.60E-02 (2.07E-04) |

### Table 4: Friedman test with Hochberg’s post hoc for iBetaCODE and ten ODE variants on CEC 2013 test suite at 50-D.

| Algorithm | Average ranking | p-value | p-value (Hochberg) | Sig. |
|-----------|-----------------|---------|-------------------|------|
| 1 | iBetaCODE | 3.41 | - | - | 1.00 |
| 2 | AMODE | 5.23 | -1.81E+00 | 6.68E-02 | 2.34E-01 | No |
| 3 | iBetaCODE | 4.07 | 5.24E-01 | 6.00E-01 | 0.03E-00 | 0.03E-00 | Yes |
| 4 | COODE | 7.79 | -4.71E+00 | 2.43E-06 | 2.19E-05 | Yes |
| 5 | EODE | 5.32 | 5.31E-01 | 5.31E-01 | 0.03E-00 | 0.03E-00 | Yes |
| 6 | EODE | 5.23 | -1.81E+00 | 6.68E-02 | 2.34E-01 | No |
| 7 | EODE | 4.72 | -4.71E+00 | 2.43E-06 | 2.19E-05 | Yes |
| 8 | ODEFPG | 5.00 | -1.81E+00 | 6.68E-02 | 2.34E-01 | No |
| 9 | QODE | 5.64 | -3.42E+00 | 6.15E-04 | 3.04E-03 | Yes |
| 10 | QODE | 6.64 | -5.86E+00 | 1.34E-08 | 1.34E-07 | Yes |

Test statistics: Chi Square = 65.44, Sig. = Yes, p-value = 3.33E-10
Table 5: Mean and standard deviation of FEVs for iBetaCODE and ten ODE variants on CEC 2017 test suite at 30-D.

| Algorithm                  | ODE (Mean [STD DEV]) | ODE3 (Mean [STD DEV]) | ODE4 (Mean [STD DEV]) | ODE5 (Mean [STD DEV]) | ODE6 (Mean [STD DEV]) | ODE7 (Mean [STD DEV]) | ODE8 (Mean [STD DEV]) | ODE9 (Mean [STD DEV]) | ODE10 (Mean [STD DEV]) |
|----------------------------|----------------------|------------------------|-----------------------|------------------------|-----------------------|-----------------------|-----------------------|-----------------------|------------------------|
| iBetaCODE                  | 5.18 (0.76)          | 5.28 (1.26)            | 5.34 (0.96)           | 5.31 (1.23)            | 5.30 (1.10)           | 5.39 (1.07)           | 5.35 (1.00)          | 5.37 (1.03)          | 5.37 (1.08)            |
| ABGC                       | 6.49 (1.02)          | 6.49 (1.02)            | 6.39 (1.00)           | 6.39 (1.00)            | 6.39 (1.00)           | 6.40 (1.00)           | 6.40 (1.00)          | 6.40 (1.00)          | 6.40 (1.00)            |
| ADS                        | 5.73 (0.80)          | 5.73 (0.80)            | 5.73 (0.80)           | 5.73 (0.80)            | 5.73 (0.80)           | 5.73 (0.80)           | 5.73 (0.80)          | 5.73 (0.80)          | 5.73 (0.80)            |
| EFO                        | 5.39 (0.81)          | 5.39 (0.81)            | 5.39 (0.81)           | 5.39 (0.81)            | 5.39 (0.81)           | 5.39 (0.81)           | 5.39 (0.81)          | 5.39 (0.81)          | 5.39 (0.81)            |
| EGO                        | 5.39 (0.81)          | 5.39 (0.81)            | 5.39 (0.81)           | 5.39 (0.81)            | 5.39 (0.81)           | 5.39 (0.81)           | 5.39 (0.81)          | 5.39 (0.81)          | 5.39 (0.81)            |
| EGE                        | 5.39 (0.81)          | 5.39 (0.81)            | 5.39 (0.81)           | 5.39 (0.81)            | 5.39 (0.81)           | 5.39 (0.81)           | 5.39 (0.81)          | 5.39 (0.81)          | 5.39 (0.81)            |
| EOE                        | 5.39 (0.81)          | 5.39 (0.81)            | 5.39 (0.81)           | 5.39 (0.81)            | 5.39 (0.81)           | 5.39 (0.81)           | 5.39 (0.81)          | 5.39 (0.81)          | 5.39 (0.81)            |
| EOE1                       | 5.39 (0.81)          | 5.39 (0.81)            | 5.39 (0.81)           | 5.39 (0.81)            | 5.39 (0.81)           | 5.39 (0.81)           | 5.39 (0.81)          | 5.39 (0.81)          | 5.39 (0.81)            |
| EOE2                       | 5.39 (0.81)          | 5.39 (0.81)            | 5.39 (0.81)           | 5.39 (0.81)            | 5.39 (0.81)           | 5.39 (0.81)           | 5.39 (0.81)          | 5.39 (0.81)          | 5.39 (0.81)            |
| EOE3                       | 5.39 (0.81)          | 5.39 (0.81)            | 5.39 (0.81)           | 5.39 (0.81)            | 5.39 (0.81)           | 5.39 (0.81)           | 5.39 (0.81)          | 5.39 (0.81)          | 5.39 (0.81)            |
| EOE4                       | 5.39 (0.81)          | 5.39 (0.81)            | 5.39 (0.81)           | 5.39 (0.81)            | 5.39 (0.81)           | 5.39 (0.81)           | 5.39 (0.81)          | 5.39 (0.81)          | 5.39 (0.81)            |

Table 6: Friedman test with Hochberg's post hoc for iBetaCODE and ten ODE variants on CEC 2017 test suite at 30-D.
| Algorithm | Friedman Test | Wilcoxon Signed-Rank Test | Average of Ranks | Z Value | p-Value | Average of Ranks | Z Value | p-Value |
|-----------|--------------|---------------------------|------------------|---------|---------|------------------|---------|---------|
| 1 | 3.05 | 0.05 | 3.65 | 1.83 | 0.01 | Yes | | |
| 2 | 3.53 | 0.03 | 5.95 | 2.22 | 0.02 | Yes | | |
| 3 | 5.51 | 1.98 | 5.11 | 1.61 | 0.02 | Yes | | |
| 4 | 5.86 | 1.98 | 5.22 | 1.61 | 0.02 | Yes | | |
| 5 | 5.11 | 1.98 | 5.22 | 1.61 | 0.02 | Yes | | |
| 6 | 5.00 | 1.98 | 5.22 | 1.61 | 0.02 | Yes | | |
| 7 | 5.50 | 1.98 | 5.11 | 1.61 | 0.02 | Yes | | |
| 8 | 5.50 | 1.98 | 5.11 | 1.61 | 0.02 | Yes | | |
| 9 | 5.50 | 1.98 | 5.11 | 1.61 | 0.02 | Yes | | |
| 10 | 5.50 | 1.98 | 5.11 | 1.61 | 0.02 | Yes | | |

Table 7: Mean and standard deviation of FEVs for iBetaCODE and ten ODE variants on CEC 2017 test suite at 50-D.

| iBetaCODE | ACOODE | BIOCODE | CODE | EDOE | CODE |
|-----------|--------|---------|------|------|------|
| N 30      | 2.95E  | 6.52E   | 7.85E | 9.25E | 9.52E |
|            | +      | +       | +    | +    | +    |
| F1        | 3.78E  | 4.10E   | 4.10E | 4.10E | 4.10E |
| F13       | 3.78E  | 4.10E   | 4.10E | 4.10E | 4.10E |
| F14       | 3.78E  | 4.10E   | 4.10E | 4.10E | 4.10E |
| F15       | 3.78E  | 4.10E   | 4.10E | 4.10E | 4.10E |
| F16       | 3.78E  | 4.10E   | 4.10E | 4.10E | 4.10E |
| F17       | 3.78E  | 4.10E   | 4.10E | 4.10E | 4.10E |
| F18       | 3.78E  | 4.10E   | 4.10E | 4.10E | 4.10E |
| F19       | 3.78E  | 4.10E   | 4.10E | 4.10E | 4.10E |
| F20       | 3.78E  | 4.10E   | 4.10E | 4.10E | 4.10E |
| F21       | 3.78E  | 4.10E   | 4.10E | 4.10E | 4.10E |
| F22       | 3.78E  | 4.10E   | 4.10E | 4.10E | 4.10E |
| F23       | 3.78E  | 4.10E   | 4.10E | 4.10E | 4.10E |
| F24       | 3.78E  | 4.10E   | 4.10E | 4.10E | 4.10E |
| F25       | 3.78E  | 4.10E   | 4.10E | 4.10E | 4.10E |
| F26       | 3.78E  | 4.10E   | 4.10E | 4.10E | 4.10E |
| F27       | 3.78E  | 4.10E   | 4.10E | 4.10E | 4.10E |
| F28       | 3.78E  | 4.10E   | 4.10E | 4.10E | 4.10E |
| F29       | 3.78E  | 4.10E   | 4.10E | 4.10E | 4.10E |
| F30       | 3.78E  | 4.10E   | 4.10E | 4.10E | 4.10E |

Table 8: Friedman test with Hochberg's post hoc for iBetaCODE and ten ODE variants on CEC 2017 test suite at 50-D.
1) DE/best/1/bin

\[ u_{i,g}^{j} = \begin{cases} 
    x_{i,best,g}^{j} + F \cdot (x_{i,rand,g}^{j} - x_{i,best,g}^{j}) & \text{if } rand_{i,j} \leq CR \text{ or } j == j_{rand} \\
    x_{i,g}^{j} + F \cdot (x_{i,rand,g}^{j} - x_{i,best,g}^{j}) & \text{otherwise}
\end{cases} \]  

(25)

2) DE/target-to-best/1/bin

\[ u_{i,g}^{j} = \begin{cases} 
    x_{i,best,g}^{j} + F \cdot (x_{i,rand,g}^{j} - x_{i,best,g}^{j}) + F \cdot (x_{i,best,g}^{j} - x_{i,best,g}^{j}) & \text{if } rand_{i,j} \leq CR \text{ or } j == j_{rand} \\
    x_{i,g}^{j} + F \cdot (x_{i,rand,g}^{j} - x_{i,best,g}^{j}) + F \cdot (x_{i,best,g}^{j} - x_{i,best,g}^{j}) & \text{otherwise}
\end{cases} \]  

(26)

3) DE/current-to-rand/1

\[ u_{i,g} = x_{i,g} + K \cdot (x_{i,rand,g} - x_{i,best,g}) + F \cdot (x_{i,rand,g} - x_{i,best,g}) \]  

(27)

4) DE/best/2/bin

\[ u_{i,g}^{j} = \begin{cases} 
    x_{i,best,g}^{j} + F \cdot (x_{i,rand,g}^{j} - x_{i,best,g}^{j}) + F \cdot (x_{i,best,g}^{j} - x_{i,best,g}^{j}) & \text{if } rand_{i,j} \leq CR \text{ or } j == j_{rand} \\
    x_{i,g}^{j} + F \cdot (x_{i,rand,g}^{j} - x_{i,best,g}^{j}) + F \cdot (x_{i,best,g}^{j} - x_{i,best,g}^{j}) & \text{otherwise}
\end{cases} \]  

(28)

5) DE/target-to-best/2/bin

\[ u_{i,g}^{j} = \begin{cases} 
    x_{i,best,g}^{j} + F \cdot (x_{i,rand,g}^{j} - x_{i,best,g}^{j}) + F \cdot (x_{i,best,g}^{j} - x_{i,best,g}^{j}) & \text{if } rand_{i,j} \leq CR \text{ or } j == j_{rand} \\
    x_{i,g}^{j} + F \cdot (x_{i,rand,g}^{j} - x_{i,best,g}^{j}) + F \cdot (x_{i,best,g}^{j} - x_{i,best,g}^{j}) & \text{otherwise}
\end{cases} \]  

(29)

6) DE/current-to-rand/2

\[ u_{i,g} = x_{i,g} + K \cdot (x_{i,rand,g} - x_{i,best,g}) + F \cdot (x_{i,rand,g} - x_{i,best,g}) + F \cdot (x_{i,rand,g} - x_{i,best,g}) \]  

(30)

Pandemic multimodal \((F_{11},F_{20})\) and hybrid composition test functions \((F_{21},F_{30})\). These results revealed that the proposed algorithm has a strong exploitation and exploration property, and thus, it is capable of discovering more decent solutions than the others overall.

We found similar tendencies for fifty dimensions in Tables 7 and 8. As a result, the proposed algorithm clearly demonstrates that it has better searchability than all the others including its predecessor BetaCODE on the CEC 2017 test suite. Therefore, the proposed algorithm demonstrates a superior overall performance compared to BetaCODE on the CEC 2017 test suite.

6.2. Comparison with BetaCODE

In the previous section, we compared the proposed algorithm to BetaCODE based on a classical DE variant DE/rand/1/bin. To more rigorously compare iBetaCODE to BetaCODE, we carried out additional experiments using six other classical DE variants, namely: DE/best/1/bin, DE/target-to-best/1/bin, DE/current-to-rand/1, DE/best/2/bin, DE/target-to-best/2/bin, and DE/current-to-rand/2. These classical DE variants are frequently used after DE/rand/1/bin. For a fair comparison, we used the same values for the control parameters associated with DE: \(F = 0.5, CR = 0.5, \text{ and } NP = 100\). Additionally, we used \(DT = 1e^{-6}\) for the diversity threshold and \(J_r = 0.05\) for the jumping rate.

6.2.1. iBetaCODE versus BetaCODE on CEC 2013 Test Suite

Table 9 presents the mean and standard deviation of the FEVs for each algorithm on the CEC 2013 test suite, collected through 51 time-independent runs for thirty dimensions. The table shows that the iBetaCODE variants searched out more precise solutions than the BetaCODE variants with all the classical DE variants. More specifically, each iBetaCODE variant yielded more statistically accurate solutions than its corresponding BetaCODE variant for more than 35% of the overall problems. In addition, Table 10 presents the experimental results for fifty dimensions, where the iBetaCODE variants all discovered more accurate solutions than the BetaCODE variants for more than 50% of the overall problems. Therefore, the proposed algorithm demonstrates a superior overall performance compared to BetaCODE on the CEC 2013 test suite.

6.2.2. iBetaCODE versus BetaCODE on CEC 2017 Test Suite

Table 11 presents the mean and standard deviation of the FEVs for each algorithm on the CEC 2017 test suite, collected through 51 time-independent runs for thirty dimensions. The table shows that the iBetaCODE variants searched out more precise solutions than the BetaCODE variants with all of the classical DE variants except DE/best/1/bin. In addition, Table 12 presents the experimental results for fifty dimensions, where the iBetaCODE variants all discovered more accurate solutions than the BetaCODE variants for more than 20% of the overall
Table 9: Mean and standard deviation of FEVs for iBetaCOBL and BetaCOBL using six classical DEs on CEC 2013 test suite at 30-D.

| DEs     | Best CODE | Best CODE | DEs RANGE to best CODE | DEs current to CODE | Mean (STD DEV) | Mean (STD DEV) | Mean (STD DEV) |
|---------|-----------|-----------|------------------------|---------------------|----------------|----------------|----------------|
| iBetaCOBL | 3.000±0.1 (1.40±0.1) | 3.120±0.1 (1.20±0.1) | 3.250±0.1 (1.50±0.1) | 3.340±0.1 (1.60±0.1) | 3.480±0.1 (1.80±0.1) | 3.620±0.1 (1.90±0.1) | 3.740±0.1 (2.00±0.1) | 3.870±0.1 (2.10±0.1) | 3.990±0.1 (2.20±0.1) | 4.080±0.1 (2.30±0.1) | 4.180±0.1 (2.40±0.1) | 4.280±0.1 (2.50±0.1) |
| BetaCOBL | 3.250±0.1 (1.75±0.1) | 3.340±0.1 (1.60±0.1) | 3.480±0.1 (1.80±0.1) | 3.620±0.1 (1.90±0.1) | 3.740±0.1 (2.00±0.1) | 3.870±0.1 (2.10±0.1) | 3.990±0.1 (2.20±0.1) | 4.080±0.1 (2.30±0.1) | 4.180±0.1 (2.40±0.1) | 4.280±0.1 (2.50±0.1) | 4.380±0.1 (2.60±0.1) | 4.480±0.1 (2.70±0.1) |
| | | | | | | | | | | | | |
Table 10: Mean and standard deviation of FEVs for iBetaCOBL and BetaCOBL using six classical DEs on CEC 2013 test suite at 50-D.

| DE/betaCOBL | BestCODE Mean (STD DEV) | BestCODE Mean (STD DEV) | DE/betaCOBL Mean (STD DEV) | DE/betaCOBL Mean (STD DEV) | DE/betaCOBL Mean (STD DEV) | DE/betaCOBL Mean (STD DEV) |
|-------------|--------------------------|--------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|
| P1          | 2.295x10^10 (5.13x10^9)  | 2.295x10^10 (5.13x10^9)  | 2.295x10^10 (5.13x10^9)    | 2.295x10^10 (5.13x10^9)    | 2.295x10^10 (5.13x10^9)    | 2.295x10^10 (5.13x10^9)    |
| P2          | 2.295x10^10 (5.13x10^9)  | 2.295x10^10 (5.13x10^9)  | 2.295x10^10 (5.13x10^9)    | 2.295x10^10 (5.13x10^9)    | 2.295x10^10 (5.13x10^9)    | 2.295x10^10 (5.13x10^9)    |
| P3          | 2.295x10^10 (5.13x10^9)  | 2.295x10^10 (5.13x10^9)  | 2.295x10^10 (5.13x10^9)    | 2.295x10^10 (5.13x10^9)    | 2.295x10^10 (5.13x10^9)    | 2.295x10^10 (5.13x10^9)    |
| P4          | 2.295x10^10 (5.13x10^9)  | 2.295x10^10 (5.13x10^9)  | 2.295x10^10 (5.13x10^9)    | 2.295x10^10 (5.13x10^9)    | 2.295x10^10 (5.13x10^9)    | 2.295x10^10 (5.13x10^9)    |
| P5          | 2.295x10^10 (5.13x10^9)  | 2.295x10^10 (5.13x10^9)  | 2.295x10^10 (5.13x10^9)    | 2.295x10^10 (5.13x10^9)    | 2.295x10^10 (5.13x10^9)    | 2.295x10^10 (5.13x10^9)    |

Total | 1/1/17 | 18/1/17 |

Total | 1/1/17 | 18/1/17 |

Total | 1/1/17 | 18/1/17 |
Table 11: Mean and standard deviation of FEVs for $i$BetaCODEL and BetaCODEL using six classical DEs on CEC 2017 test suite at 30-D.

| DEcurrent-to-fed | DE<sup>1</sup> | DE<sup>2</sup> | DE<sup>3</sup> |
|-----------------|---------------|---------------|---------------|
| BetaCODEL       | (mean, std dev) | (mean, std dev) | (mean, std dev) |
| BetaCODEL       | (mean, std dev) | (mean, std dev) | (mean, std dev) |

| DE<sup>1</sup> | DE<sup>2</sup> | DE<sup>3</sup> |
|---------------|---------------|---------------|
| 2.10E-02 (0.32) | 5.06E-03 (0.15) | 5.54E-03 (0.19) |
| 1.98E-01 (0.62) | 5.02E-01 (0.16) + 6.22E-01 (0.46) | 2.35E-01 (1.34) + 3.71E-01 (1.77) |
| 9.78E-01 (0.32) | 2.36E-01 (0.16) + 6.22E-01 (0.37) | 2.35E-01 (1.34) + 3.71E-01 (1.77) |
| 4.68E-01 (0.49) | 1.24E-01 (0.16) + 6.22E-01 (0.37) | 2.35E-01 (1.34) + 3.71E-01 (1.77) |
| 4.53E-01 (0.30) | 1.24E-01 (0.16) + 6.22E-01 (0.37) | 2.35E-01 (1.34) + 3.71E-01 (1.77) |
| 5.24E-01 (0.56) | 1.60E-01 (0.17) + 6.22E-01 (0.37) | 2.35E-01 (1.34) + 3.71E-01 (1.77) |
| 3.90E-01 (0.26) | 2.06E-01 (0.18) + 6.22E-01 (0.37) | 2.35E-01 (1.34) + 3.71E-01 (1.77) |
| 9.77E-01 (0.11) | 8.28E-01 (0.23) + 6.22E-01 (0.37) | 2.35E-01 (1.34) + 3.71E-01 (1.77) |
| 5.43E-01 (0.27) | 8.28E-01 (0.23) + 6.22E-01 (0.37) | 2.35E-01 (1.34) + 3.71E-01 (1.77) |
| 5.45E-01 (0.26) | 3.86E-01 (0.17) + 6.22E-01 (0.37) | 2.35E-01 (1.34) + 3.71E-01 (1.77) |
| 4.70E-01 (0.35) | 3.86E-01 (0.17) + 6.22E-01 (0.37) | 2.35E-01 (1.34) + 3.71E-01 (1.77) |
| 3.90E-01 (0.26) | 3.86E-01 (0.17) + 6.22E-01 (0.37) | 2.35E-01 (1.34) + 3.71E-01 (1.77) |
| 5.43E-01 (0.27) | 3.86E-01 (0.17) + 6.22E-01 (0.37) | 2.35E-01 (1.34) + 3.71E-01 (1.77) |
| 3.90E-01 (0.26) | 5.06E-01 (0.15) + 6.22E-01 (0.37) | 2.35E-01 (1.34) + 3.71E-01 (1.77) |
| 5.43E-01 (0.27) | 5.06E-01 (0.15) + 6.22E-01 (0.37) | 2.35E-01 (1.34) + 3.71E-01 (1.77) |
| 3.90E-01 (0.26) | 1.50E-01 (0.17) + 6.22E-01 (0.37) | 2.35E-01 (1.34) + 3.71E-01 (1.77) |
| 5.43E-01 (0.27) | 1.50E-01 (0.17) + 6.22E-01 (0.37) | 2.35E-01 (1.34) + 3.71E-01 (1.77) |
| 3.90E-01 (0.26) | 1.50E-01 (0.17) + 6.22E-01 (0.37) | 2.35E-01 (1.34) + 3.71E-01 (1.77) |
| 5.43E-01 (0.27) | 1.50E-01 (0.17) + 6.22E-01 (0.37) | 2.35E-01 (1.34) + 3.71E-01 (1.77) |
| 3.90E-01 (0.26) | 1.50E-01 (0.17) + 6.22E-01 (0.37) | 2.35E-01 (1.34) + 3.71E-01 (1.77) |
| 5.43E-01 (0.27) | 1.50E-01 (0.17) + 6.22E-01 (0.37) | 2.35E-01 (1.34) + 3.71E-01 (1.77) |
| 3.90E-01 (0.26) | 1.50E-01 (0.17) + 6.22E-01 (0.37) | 2.35E-01 (1.34) + 3.71E-01 (1.77) |
| 5.43E-01 (0.27) | 1.50E-01 (0.17) + 6.22E-01 (0.37) | 2.35E-01 (1.34) + 3.71E-01 (1.77) |
| 3.90E-01 (0.26) | 1.50E-01 (0.17) + 6.22E-01 (0.37) | 2.35E-01 (1.34) + 3.71E-01 (1.77) |
| 5.43E-01 (0.27) | 1.50E-01 (0.17) + 6.22E-01 (0.37) | 2.35E-01 (1.34) + 3.71E-01 (1.77) |
| 3.90E-01 (0.26) | 1.50E-01 (0.17) + 6.22E-01 (0.37) | 2.35E-01 (1.34) + 3.71E-01 (1.77) |

**Total:** 19110

22/11

7/105
Table 12: Mean and standard deviation of FEVs for iBetaCOBL and BetaCOBL using six classical DEs on CEC 2017 test suite at 50-D.

| DE | iBetaCOBL | BetaCOBL |
|---|---|---|
| FEV | Mean (STD DEV) | Mean (STD DEV) | Mean (STD DEV) | Mean (STD DEV) |
| F1 | 1.54E+07 (9.72E+06) | 6.17E+06 (1.02E+07) | 7.42E+06 (7.37E+06) | 7.42E+06 (7.94E+06) |
| F2 | 3.13E+07 (3.58E+06) | 5.47E+06 (3.34E+07) | 2.35E+07 (7.36E+06) | 1.80E+07 (7.42E+06) |
| F3 | 2.15E+07 (4.46E+06) | 7.46E+06 (1.06E+07) | 2.34E+07 (3.67E+06) | 1.83E+07 (3.98E+06) |
| F4 | 5.41E+07 (7.66E+06) | 3.19E+06 (1.64E+07) | 2.60E+07 (4.47E+06) | 1.87E+07 (4.97E+06) |
| F5 | 7.31E+07 (4.27E+06) | 4.13E+06 (7.28E+06) | 4.31E+07 (4.21E+06) | 3.42E+07 (5.93E+06) |
| F6 | 1.15E+08 (4.27E+06) | 5.15E+06 (1.22E+07) | 2.20E+07 (4.20E+06) | 2.10E+07 (4.24E+06) |
| F7 | 1.71E+08 (4.12E+06) | 5.12E+06 (1.19E+07) | 1.01E+08 (4.01E+06) | 7.54E+07 (1.50E+07) |
| F8 | 3.83E+08 (5.63E+06) | 5.09E+06 (1.13E+07) | 3.08E+08 (5.32E+06) | 5.89E+07 (1.03E+07) |
| F9 | 8.25E+08 (3.94E+06) | 9.10E+06 (2.13E+07) | 1.01E+09 (2.09E+06) | 1.43E+08 (4.48E+06) |
| F10 | 4.34E+09 (8.68E+06) | 6.01E+06 (1.47E+07) | 9.27E+08 (2.02E+06) | 8.64E+07 (2.29E+06) |

Total: 7/5

| DE | iBetaCOBL | BetaCOBL |
|---|---|---|
| FEV | Mean (STD DEV) | Mean (STD DEV) | Mean (STD DEV) | Mean (STD DEV) |
| F1 | 5.06E+02 (5.95E+02) | 1.17E+02 (1.02E+03) | 7.62E+01 (7.71E+03) | 7.62E+01 (7.94E+03) |
| F2 | 2.51E+02 (1.35E+03) | 5.47E+02 (3.34E+03) | 2.35E+02 (7.36E+05) | 1.80E+02 (7.42E+05) |
| F3 | 3.14E+02 (4.46E+02) | 7.46E+02 (1.06E+03) | 2.34E+02 (3.67E+06) | 1.83E+02 (3.98E+06) |
| F4 | 6.51E+02 (7.66E+02) | 3.19E+02 (1.64E+03) | 2.60E+02 (4.47E+06) | 1.87E+02 (4.97E+06) |
| F5 | 7.31E+02 (4.27E+02) | 4.13E+02 (7.28E+02) | 4.31E+02 (4.21E+06) | 3.42E+02 (5.93E+06) |
| F6 | 1.15E+03 (4.27E+02) | 5.15E+02 (1.22E+03) | 2.20E+02 (4.20E+05) | 2.10E+02 (4.24E+05) |
| F7 | 1.71E+03 (4.12E+02) | 5.12E+02 (1.19E+03) | 1.01E+03 (4.01E+05) | 7.54E+02 (1.50E+05) |
| F8 | 3.83E+03 (5.63E+02) | 5.09E+02 (1.13E+04) | 3.08E+03 (5.32E+04) | 5.89E+02 (1.03E+04) |
| F9 | 8.25E+03 (3.94E+02) | 9.10E+02 (2.13E+05) | 1.01E+03 (2.09E+06) | 1.43E+02 (4.48E+06) |
| F10 | 4.34E+04 (8.68E+02) | 6.01E+02 (1.47E+06) | 9.27E+02 (2.02E+06) | 8.64E+01 (2.29E+06) |
problems. Therefore, the proposed algorithm demonstrates superior overall performance compared to BetaCODE on the CEC 2017 test suite.

Additionally, we examined Tables 11 and 12 with respect to the attributes of the test functions. We found that the iBetaCODE variants significantly outperform the BetaCODE variants not only on the unimodal functions but also on the multimodal, expand multimodal and hybrid composition functions. Notably, the BetaCODE variants based on DE/best/2/bin for thirty dimensions and DE/target-to-best/1/bin and DE/target-to-best/2/bin for fifty dimensions could not find any better solutions than their corresponding iBetaCODE variants in any of the test functions. In summary, the proposed algorithm is superior to its predecessor on the CEC 2017 test suite for both thirty and fifty dimensions.

6.3. Performance of BetaCODE with Linear Time Diversity Measures

Table 13: Mean and standard deviation of FEVs for original BetaCODE and linear time BetaCODE variants on CEC 2013 test suite at 30-D.

| BetaCODE       | MEAN (STD DEV) | BetaCODE_linear | MEAN (STD DEV) | BetaCODE_linear2 | MEAN (STD DEV) |
|----------------|----------------|-----------------|----------------|-------------------|----------------|
| F1             | 0.00E+00 (0.00E+00) | 0.00E+00 (0.00E+00) | 0.00E+00 (0.00E+00) |                   |                 |
| F2             | 3.88E+01 (5.29E+01) | 6.65E+01 (1.32E+02) | 3.84E+01 (2.61E+01) |                   |                 |
| F3             | 2.48E+01 (1.00E+02) | 3.61E+01 (1.05E+01) | 2.12E+01 (1.54E+01) |                   |                 |
| F4             | 1.52E+02 (1.52E+02) | 1.71E+02 (1.39E+02) | 1.33E+02 (4.65E+02) |                   |                 |
| F5             | 9.39E+14 (4.92E+14) | 9.82E+14 (8.40E+14) | 9.39E+14 (9.49E+14) |                   |                 |
| F6             | 1.07E+01 (1.95E+01) | 1.12E+01 (1.64E+01) | 1.04E+01 (4.38E+00) |                   |                 |
| F7             | 1.57E+01 (2.60E+01) | 2.01E+01 (2.70E+01) | 1.87E+01 (3.83E+01) |                   |                 |
| F8             | 2.10E+01 (3.41E+01) | 2.00E+01 (2.90E+01) | 2.10E+01 (8.60E+01) |                   |                 |
| F9             | 6.73E+00 (2.32E+00) | 6.42E+00 (2.54E+00) | 6.75E+00 (2.55E+00) |                   |                 |
| F10            | 8.60E+00 (8.35E+00) | 9.60E+00 (6.28E+00) | 6.03E+00 (7.79E+00) |                   |                 |
| F11            | 7.52E+04 (1.56E+04) | 6.42E+04 (1.22E+04) | 6.46E+04 (1.00E+05) |                   |                 |
| F12            | 1.57E+01 (1.21E+01) | 1.74E+01 (2.22E+01) | 1.74E+01 (1.57E+01) |                   |                 |
| F13            | 1.77E+01 (2.30E+01) | 1.78E+01 (2.45E+01) | 1.76E+01 (8.71E+01) |                   |                 |
| F14            | 1.00E+03 (1.31E+03) | 1.01E+03 (1.20E+03) | 1.01E+03 (5.72E+02) |                   |                 |
| F15            | 6.88E+08 (9.57E+08) | 6.91E+08 (7.92E+08) | 6.03E+08 (8.37E+08) |                   |                 |
| F16            | 2.48E+00 (1.03E+00) | 2.46E+00 (1.02E+00) | 2.46E+00 (1.01E+00) |                   |                 |
| F17            | 3.01E+02 (3.97E+00) | 3.01E+02 (4.03E+00) | 3.01E+02 (2.82E+00) |                   |                 |
| F18            | 2.11E+02 (1.41E+01) | 2.11E+02 (1.48E+01) | 2.11E+02 (9.57E+01) |                   |                 |
| F19            | 1.15E+02 (1.34E+02) | 1.16E+02 (1.29E+02) | 1.15E+02 (5.32E+01) |                   |                 |
| F20            | 1.20E+01 (1.02E+01) | 1.21E+01 (1.00E+01) | 1.21E+01 (1.00E+01) |                   |                 |
| F21            | 3.24E+00 (3.97E+00) | 3.15E+00 (3.15E+00) | 2.81E+00 (3.87E+00) |                   |                 |
| F22            | 1.04E-02 (2.31E-03) | 1.04E-02 (2.38E-03) | 1.07E-02 (2.52E-03) |                   |                 |
| F23            | 9.08E+00 (1.55E+00) | 9.77E+00 (1.54E+00) | 9.60E+00 (2.64E+00) |                   |                 |
| F24            | 2.08E-01 (2.70E-02) | 2.08E-01 (2.70E-02) | 2.08E-01 (2.70E-02) |                   |                 |
| F25            | 2.64E+01 (7.77E-01) | 2.64E+01 (4.33E-01) | 2.64E+01 (4.33E-01) |                   |                 |
| F26            | 2.06E+00 (3.00E-02) | 2.06E+00 (3.00E-02) | 2.06E+00 (3.00E-02) |                   |                 |
| F27            | 3.05E+00 (2.00E-02) | 3.05E+00 (2.00E-02) | 3.05E+00 (2.00E-02) |                   |                 |
| F28            | 3.05E+00 (2.00E-02) | 3.05E+00 (2.00E-02) | 3.05E+00 (2.00E-02) |                   |                 |

Table 14: Mean and standard deviation of FEVs for original BetaCODE and linear time BetaCODE variants on CEC 2017 test suite at 30-D.

The symbols ‘+/-’ show the statistical results of the Wilcoxon signed-rank test with α = 0.05 significance level. ‘+’ represents that the original BetaCODE is significantly superior than the compared algorithm. ‘-’ represents that the performance difference between the original BetaCODE and the compared algorithm is not statistically significant. And, ‘=’ represents that BetaCODE is significantly inferior than the compared algorithm.

To reduce computational cost, the proposed algorithm uses a linear time diversity measure $D_{A}$ in the selection switching scheme instead of a power mean-based diversity measure $D_{b}$. However, using a different diversity measure can lead to performance degradation if the diversity measure is not suitable for the selection switching scheme. Therefore, we conducted experiments to evaluate the performance of two BetaCODE variants using linear time diversity measures $D_{A}$ and $D_{b}$ and compared them with the original BetaCODE to investigate the impact of replacing the diversity measure. As in the previous experiments, we used the same classical DE variant DE/rand/1/bin, and for the control parameters associated with DE, with the following values: $F = 0.5$, $CR = 0.9$, and $NP = 100$. Additionally, we used $DT = 1e - 6$ for the diversity threshold and $J_{s} = 0.05$ for the jumping rate.

7. Performance Enhancement of DE Variants

In the previous section, we confirmed that the proposed algorithm is able to improve the convergence performance of classical DE variants significantly. We investigated further to check the compatibility of the proposed algorithm with two state-of-the-art DE variants, CoDE [29] and SHADE [30]. CoDE is a powerful DE variant that uses multiple trial vector generation strategies. SHADE is an improved version of JADE
that uses a fast mutation strategy DE/current-to-pbest, an external archive that stores the convergence progress, and a historical memory-based adaptive parameter control. We assessed the performance of CoDE, CoDE-BetaCOBL, and CoDE-iBetaCOBL, as well as that of SHADE variants. We used the diversity threshold $D^T = 1e-6$ and the jumping rate $J_r = 0.05$, and the values recommended by the authors of each paper for the remaining control parameters.

### 7.1. Performance Enhancement of CoDE Algorithm

#### Table 15: Mean and standard deviation of FEVs for original CoDE, CoDE with BetaCOBL, and CoDE with iBetaCOBL on CEC 2013 test suite at 50-D.

| Algorithm                  | CEC06-DECOBL | CEC06-iBetaCOBL | CEC06-BetaCOBL |
|----------------------------|--------------|-----------------|-----------------|
| Mean (STD DEV)             | Mean (STD DEV) | Mean (STD DEV)  | Mean (STD DEV)  |
| CoDE                        | 5.46E+03 (1.27E+03) | 8.28E+03 (1.29E+03) | 8.28E+03 (1.29E+03) |
| CoDE-iBetaCOBL             | 5.00E+03 (1.35E+03) | 8.28E+03 (1.29E+03) | 8.28E+03 (1.29E+03) |
| CoDE-BetaCOBL              | 6.12E+03 (1.29E+03) | 8.28E+03 (1.29E+03) | 8.28E+03 (1.29E+03) |

#### Table 16: Mean and standard deviation of FEVs for original CoDE, CoDE with BetaCOBL, and CoDE with iBetaCOBL on CEC 2017 test suite at 50-D.

| Algorithm                  | CEC07-DECOBL | CEC07-iBetaCOBL | CEC07-BetaCOBL |
|----------------------------|--------------|-----------------|-----------------|
| Mean (STD DEV)             | Mean (STD DEV) | Mean (STD DEV)  | Mean (STD DEV)  |
| CoDE                        | 4.51E+03 (5.47E+03) | 5.03E+03 (3.46E+03) | 7.56E+03 (5.67E+03) |
| CoDE-iBetaCOBL             | 5.00E+03 (5.72E+03) | 5.03E+03 (3.46E+03) | 7.56E+03 (5.67E+03) |
| CoDE-BetaCOBL              | 5.19E+03 (5.46E+03) | 5.03E+03 (3.46E+03) | 7.56E+03 (5.67E+03) |

The symbols ‘*’ and ‘**’ show the statistical results of the Wilcoxon signed-rank test with a 0.05 significance level. ‘*’ represents that CoDE with iBetaCOBL is significantly superior than the compared algorithms. ‘**’ represents that the performance difference between CoDE with iBetaCOBL and the compared algorithm is not statistically significant. And, ‘***’ represents that CoDE with BetaCOBL is significantly inferior than the compared algorithm.

The performance evaluation results of the CoDE variants on the CEC 2013 and 2017 test suites are presented in Tables 15 and 16 as collected through 51 independent runs. As we can see from Table 15, CoDE-iBetaCOBL points to promising overall performance compared to CoDE-BetaCOBL and the original CoDE on the CEC 2013 test suite. More specifically, CoDE-iBetaCOBL found 19 significantly better solutions than CoDE-BetaCOBL, while CoDE-BetaCOBL only found three considerably better solutions than CoDE-iBetaCOBL. Similarly, CoDE-iBetaCOBL discovered 21 more accurate solutions than the original CoDE, while the original CoDE could not discover any better solution than CoDE-iBetaCOBL. We found similar tendencies on the CEC 2017 test suite where CoDE-iBetaCOBL performed significantly better than CoDE-BetaCOBL, which found fifteen statistically better solutions. In addition, CoDE-iBetaCOBL performed significantly better than the original CoDE, which found 23 statistically better solutions.

Figs. 4 and 5 present the convergence graphs of CoDE variants on $F_8$, $F_{10}$, $F_{15}$, and $F_{16}$ of the CEC 2013 benchmark problems and $F_{10}$, $F_{11}$, $F_{21}$, and $F_{22}$ of the CEC 2017 benchmark problems, respectively. As we can see from the figures, the convergence progress of CoDE-iBetaCOBL is significantly better than that of the compared algorithms. In particular, Fig. 4 shows that CoDE-iBetaCOBL was able to escape the local optimum while both CoDE-BetaCOBL and the original CoDE were not. In addition, the benefit of using iBetaCOBL and BetaCOBL is apparent in Figs. 4(b), 5(a), 5(b), 5(c), and 5(d) where CoDE-iBetaCOBL and CoDE-BetaCOBL found more accurate solutions than SHADE-iBetaCOBL, while SHADE-BetaCOBL only found two considerably better solutions than SHADE-BetaCOBL. Similarly, SHADE-iBetaCOBL discovered eight considerably better solutions than SHADE-BetaCOBL, while SHADE-BetaCOBL only found two considerably better solutions than SHADE-iBetaCOBL. SHADE-iBetaCOBL discovered eight significantly better solutions than SHADE-BetaCOBL, while SHADE-BetaCOBL only found two considerably better solutions than SHADE-iBetaCOBL. SHADE-iBetaCOBL discovered eight significantly better solutions than SHADE-BetaCOBL, while SHADE-BetaCOBL only found two considerably better solutions than SHADE-iBetaCOBL.

### 7.2. Performance Enhancement of SHADE Algorithm

The performance evaluation results of the SHADE variants on the CEC 2013 and 2017 test suites are presented in Tables 17 and 18 as collected through 51 independent runs. As we can see from Table 17, SHADE-iBetaCOBL exhibits a promising overall performance compared to SHADE-BetaCOBL and the original SHADE on the CEC 2013 test suite. More specifically, SHADE-iBetaCOBL found eight significantly better solutions than SHADE-BetaCOBL, while SHADE-BetaCOBL only found two considerably better solutions than SHADE-iBetaCOBL. SHADE-iBetaCOBL discovered eight more accurate solutions compared to the original SHADE, while the original SHADE only discovered three more precise solutions compared to SHADE-iBetaCOBL. On the CEC
Figure 4: Convergence graphs of CoDE variants on CEC 2013 benchmark problems at 50-D

Figure 5: Convergence graphs of CoDE variants on CEC 2017 benchmark problems at 50-D
Figure 6: Convergence graphs of SHADE variants on CEC 2013 benchmark problems at 50-D

(a) $F_8$
(b) $F_9$
(c) $F_{15}$
(d) $F_{16}$

Figure 7: Convergence graphs of SHADE variants on CEC 2017 benchmark problems at 50-D

(a) $F_{10}$
(b) $F_{11}$
(c) $F_{13}$
(d) $F_{14}$
Table 17: Mean and standard deviation of FEVs for original SHADE, SHADE with BetaCOBL, and SHADE with iBetaCOBL on CEC 2013 test suite at 50-D.

The symbols "+" and "-" show the statistical results of the Wilcoxon signed-rank test with α = 0.05 significance level. "+" represents that SHADE with BetaCOBL is significantly superior than the compared algorithm. 

2017 test suite. SHADE-iBetaCOBL performed slightly better than SHADE-BetaCOBL, which found two statistically better solutions although SHADE-iBetaCOBL led to a significantly lower computational cost. In addition, SHADE-iBetaCOBL performed significantly better than the original SHADE, which found eight statistically better solutions.

Fig. 6(a) shows that SHADE-iBetaCOBL was able to escape the local optimum while both SHADE-BetaCOBL and the original SHADE were not. In addition, the benefit of using iBetaCOBL and BetaCOBL is apparent in Figs. 6(b) and 6(d) where both SHADE-iBetaCOBL and SHADE-BetaCOBL found more accurate solutions than the original SHADE.

Therefore, we make the following observations on the performance evaluation results.

1. A significant performance improvement of SHADE can be achieved by incorporating the proposed OBL.
2. SHADE-iBetaCOBL searched out more accurate solutions than SHADE-BetaCOBL on the CEC 2013 test suite. Although SHADE-iBetaCOBL performed slightly better than SHADE-BetaCOBL on the CEC 2017 test suite, it incurs a significantly lower computational cost.
3. SHADE-iBetaCOBL achieves promising convergence performance, with a better searchability than SHADE-BetaCOBL and the original SHADE.

Table 18: Mean and standard deviation of FEVs for original SHADE, SHADE with BetaCOBL, and SHADE with iBetaCOBL on CEC 2017 test suite at 50-D.

8 Conclusion

We have proposed a new stochastic OBL variant called iBetaCOBL, as an improved version of BetaCOBL. BetaCOBL was introduced to improve the convergence performance of DE using the beta distribution, the partial dimensional change scheme, and the selection switching scheme. However, using it to optimize more complex problems is challenging because of the high computational cost and ineffectiveness in handling dependent variables. To drastically reduce the computational cost, we applied a linear time diversity measure in the selection switching scheme. In addition, we applied the multiplicative exponential rebalibration in the partial dimensional change scheme to preserve structures with strongly dependent decision variables adjacent to each other.

The performance of iBetaCOBL was evaluated on a set of 58 challenging test functions from the CEC 2013 and 2017 test suites. Compared to ten state-of-the-art DE variants including BetaCODE, the proposed method achieved an outstanding performance overall. In addition, we applied iBetaCOBL to two state-of-the-art DE variants, CoDE and SHADE, to measure the performance enhancement. The results of the experiment show that CoDE-iBetaCOBL and SHADE-iBetaCOBL have the ability to find more accurate solutions than the BetaCOBL-based
variants, as well as CoDE and SHADE. Consequently, we confirmed that a significant performance improvement for the DE variants can be achieved using the proposed algorithm.

Possible directions for future work include 1) devising a Cauchy or Gaussian distribution-based OBL; 2) applying iBeta-COBL to multi-objective EAs; and 3) applying iBetaCOBL to other swarm and evolutionary computations.

Acknowledgement

This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. NRF-2017R1C1B2012752). The correspondence should be addressed to Dr. Yun-Gyung Cheong.

References

[1] R. Storn, K. Price, Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces, Journal of global optimization 11 (4) (1997) 341–359.
[2] K. Price, R. M. Storn, J. A. Lampinen, Differential evolution: a practical approach to global optimization, Springer Science & Business Media, 2006.
[3] S. Das, P. N. Suganthan, Differential evolution: A survey of the state-of-the-art, IEEE transactions on evolutionary computation 15 (1) (2011) 4–31.
[4] N. Hansen, A. Ostermeier, Completely derandomized self-adaptation in evolution strategies, Evolutionary computation 9 (2) (2001) 159–195.
[5] F. Neri, F. Tirronen, Recent advances in differential evolution: a survey and experimental analysis, Artificial Intelligence Review 33 (1–2) (2010) 61–106.
[6] S. Das, S. S. Mullick, P. N. Suganthan, Recent advances in differential evolution—an updated survey, Swarm and Evolutionary Computation 27 (2016) 1–30.
[7] R. D. Al-Dabbagh, F. Neri, N. Idris, M. S. Baba, Algorithmic design issues in adaptive differential evolution schemes: Review and taxonomy, Swarm and Evolutionary Computation 43 (2018) 284–311.
[8] H. R. Tizhoosh, Opposition-based learning: a new scheme for machine intelligence, in: International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (ICMA–IATWIC’06), Vol. 1, IEEE, 2005, pp. 695–701.
[9] H. R. Tizhoosh, M. Ventresca, Oppositional concepts in computational intelligence, Vol. 155, Springer, 2008.
[10] F. S. Al-Qunaieer, H. R. Tizhoosh, S. Rahnamayan, Opposition based computing survey, in: The 2010 International Joint Conference on Neural Networks (IJCNN), IEEE, 2010, pp. 1–7.
[11] Q. Xu, L. Wang, N. Wang, X. Hei, L. Zhao, A review of opposition-based learning from 2005 to 2012, Engineering Applications of Artificial Intelligence 29 (2014) 1–12.
[12] S. Mahdavi, S. Rahnamayan, K. Deb, Opposition based learning: A literature review, Swarm and evolutionary computation 39 (2018) 1–23.
[13] S. Rahnamayan, H. R. Tizhoosh, M. M. Salama, Opposition-based differential evolution, IEEE Transactions on Evolutionary computation 12 (1) (2008) 64–79.
[14] S.-Y. Park, J.-J. Lee, Stochastic opposition-based learning using a beta distribution in differential evolution, IEEE transactions on cybernetics 46 (10) (2016) 2184–2194.
[15] L. Haixun, W. Chengke, Genetic algorithm with adaptive mutation probability and analysis of its property, Acta Electron. Sin.(China) 27 (5) (1999) 90–92.
[16] M. Lozano, F. Herrera, J. R. Cano, Replacement strategies to preserve useful diversity in steady-state genetic algorithms, Information Sciences 178 (23) (2008) 4421–4433.
[17] R. Myers, E. R. Hancock, Genetic algorithms for ambiguous labelling problems, Pattern Recognition 33 (4) (2000) 685–704.
[18] A. L. Barker, W. N. Martin, Population diversity and fitness measures based on genomic distances.
[19] A. L. Barker, W. N. Martin, Dynamics of a distance-based population diversity measure, in: Proceedings of the 2000 Congress on Evolutionary Computation. CEC00 (Cat. No. 00TH8512), Vol. 2, IEEE, 2000, pp. 1002–1009.
[20] C. Mattiusi, M. Waiwel, D. Floreano, Measures of diversity for populations and distances between individuals with highly reorganizable genomes, Evolutionary Computation 12 (4) (2004) 495–515.
[21] M. Wineberg, F. Oppacher, Distance between populations, in: Genetic and Evolutionary Computation Conference, Springer, 2003, pp. 1481–1492.
[22] Y. Shi, R. C. Eberhart, Population diversity of particle swarms, in: 2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence), IEEE, 2008, pp. 1063–1067.
[23] R. Mallipeddi, P. N. Suganthan, B.-Y. Qu, Diversity enhanced adaptive evolutionary programming for solving single objective constrained problems, in: 2009 IEEE Congress on Evolutionary Computation, IEEE, 2009, pp. 2106–2113.
[24] M. Wineberg, F. Oppacher, The underlying similarity of diversity measures used in evolutionary computation, in: Genetic and Evolutionary Computation Conference, Springer, 2003, pp. 1493–1504.
[25] M. Wineberg, F. Oppacher, A linear time algorithm for determining population diversity in evolutionary computation, Proceedings of the IASTED International Conference on Intelligent Systems and Control (ICS 2003).
[26] X. Qiu, K. C. Tan, J.-X. Xu, Multiple exponential recombination for differential evolution, IEEE transactions on cybernetics 47 (4) (2017) 995–1006.
[27] J. Liang, B. Qu, P. Suganthan, A. G. Hernandez-Diaz, Problem definitions and evaluation criteria for the CEC 2013 special session on real-parameter optimization, Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou, China and Nanyang Technological University, Singapore, Technical Report 201212 (34) (2013) 281–295.
[28] N. Awad, M. Ali, J. Liang, B. Qu, P. Suganthan, Problem definitions and evaluation criteria for the CEC 2017 special session and competition on single objective bound constrained real-parameter numerical optimization, in: Technical Report, NTU, Singapore, 2016.
[29] Y. Wang, Z. Cai, Q. Zhang, Differential evolution with composite trial vector generation strategies and control parameters, IEEE Transactions on Evolutionary Computation 15 (1) (2011) 55–66.
[30] R. Tanabe, A. Fukunaga, Success-history based parameter adaptation for differential evolution, in: 2013 IEEE conference on evolutionary computation, IEEE, 2013, pp. 71–78.
[31] A. K. Qin, V. L. Huang, P. N. Suganthan, Differential evolution algorithm with strategy adaptation for global numerical optimization, IEEE transactions on Evolutionary Computation 13 (2) (2008) 398–417.
[32] W. Gong, Z. Cai, C. X. Ling, H. Li, Enhanced differential evolution with adaptive strategies for numerical optimization, IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 41 (2) (2010) 397–413.
[33] S.-Z. Zhao, P. N. Suganthan, S. Das, Self-adaptive differential evolution with multi-trajectory search for large-scale optimization, Soft Computing 15 (11) (2011) 2175–2185.
[34] T. J. Choi, C. W. Ahn, An Adaptive Cauchy Differential Evolution Algorithm with Bias Strategy Adaptation Mechanism for Global Numerical Optimization, JCP 9 (9) (2014) 2139–2145.
[35] T. J. Choi, C. W. Ahn, An adaptive cauchy differential evolution algorithm with population size reduction and modified multiple mutation strategies, in: Proceedings of the 18th Asia Pacific Symposium on Intelligent and Evolutionary Systems-Volume 2, Springer, 2015, pp. 13–26.
[36] T. J. Choi, C. W. Ahn, Adaptive Cauchy Differential Evolution with Strategy Adaptation and Its Application to Training Large-Scale Artificial Neural Networks, in: International Conference on Bio-Inspired Computing: Theories and Applications, Springer, 2017, pp. 502–510.
[37] J. Brest, S. Greiner, B. Boskovic, M. Mernik, V. Zumer, Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems, IEEE transactions on evolutionary computation 10 (6) (2006) 646–657.
[38] M. Leon, N. Xiong, Adapting differential evolution algorithms for continuous optimization via greedy adjustment of control parameters, Journal of Artificial Intelligence and Soft Computing Research 6 (2) (2016)
103–118.

[39] S. M. Islam, S. Das, S. Ghosh, S. Roy, P. N. Suganthan, An adaptive differential evolution algorithm with novel mutation and crossover strategies for global numerical optimization, IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 42 (2) (2011) 482–500.

[40] T. J. Choi, C. W. Ahn, J. An, An adaptive cauchy differential evolution algorithm for global numerical optimization, The Scientific World Journal 2013.

[41] T. J. Choi, C. W. Ahn, An adaptive differential evolution algorithm with automatic population resizing for global numerical optimization, in: Bio-Inspired Computing-Theories and Applications, Springer, 2014, pp. 68–72.

[42] T. J. Choi, C. W. Ahn, An adaptive population resizing scheme for differential evolution in numerical optimization, Journal of Computational and Theoretical Nanoscience 12 (7) (2015) 1336–1350.

[43] T. J. Choi, C. W. Ahn, Adaptive α-stable differential evolution in numerical optimization, Natural Computing 16 (4) (2017) 657–675.

[44] T. J. Choi, C. W. Ahn, An Improved Differential Evolution Algorithm and Its Application to Large-Scale Artificial Neural Networks, in: Journal of Physics: Conference Series, Vol. 806, IOP Publishing, 2017, p. 012010.

[45] R. Maltipedi, P. N. Suganthan, Q.-K. Pan, M. F. Tasgetiren, Differential evolution algorithm with ensemble of parameters and mutation strategies, Applied soft computing 11 (2) (2011) 1679–1696.

[46] N. H. Awand, M. Z. Ali, P. N. Suganthan, R. G. Reynolds, An ensemble sinusoidal parameter adaptation incorporated with L-SHADE for solving CECC2014 benchmark problems, in: 2016 IEEE congress on evolutionary computation (CEC), IEEE, 2016, pp. 2958–2965.

[47] N. H. Awand, M. Z. Ali, P. N. Suganthan, Ensemble of parameters in a sinusoidal differential evolution with niching-based population reduction, Swarm and evolutionary computation 39 (2018) 141–156.

[48] M. Ali, M. Pant, Improving the performance of differential evolution algorithm using Cauchy mutation, Soft Computing 15 (5) (2011) 991–1007.

[49] H. Qin, J. Zhou, Y. Lu, Y. Wang, Y. Zhang, Multi-objective differential evolution with adaptive Cauchy mutation for short-term multi-objective optimal hydro-thermal scheduling, Energy Conversion and Management 51 (4) (2010) 788–794.

[50] T. J. Choi, C. W. Ahn, Accelerating differential evolution using multiple exponential cauchy mutation, in: Proceedings of the Genetic and Evolutionary Computation Conference Companion, ACM, 2018, pp. 207–208.

[51] T. J. Choi, T. Jogelius, Y.-G. Cheong, ACM-DE: Adaptive p-best Cauchy Mutation with linear failure threshold reduction for Differential Evolution in numerical optimization, arXiv preprint arXiv:1907.01095.

[52] S. Das, A. Abraham, U. K. Chakraborty, A. Konar, Differential evolution using a neighborhood-based mutation operator, IEEE Transactions on Evolutionary Computation 13 (3) (2009) 526–533.

[53] M. Basu, Quasi-oppositional differential evolution for optimal reactive power dispatch, International Journal of Electrical Power & Energy Systems 78 (2016) 29–40.

[54] T. R. Chelliah, R. Thangaraj, S. Allamsetty, M. Pant, Coordination of directional overcurrent relays using opposition based chaotic differential evolution algorithm, International Journal of Electrical Power & Energy Systems 55 (2014) 341–350.

[55] H. V. H. Ayala, L. dos Santos Coelho, V. C. Mariani, A. Askarzadeh, An improved free search differential evolution algorithm: A case study on parameters identification of one diode equivalent circuit of a solar cell module, Energy 93 (2015) 1515–1522.

[56] B. Subudhi, D. Jena, A differential evolution based neural network approach to nonlinear system identification, Applied Soft Computing 11 (1) (2011) 861–871.

[57] M. Koohi-Moghadam, A. T. Rahmani, Molecular docking with opposition-based differential evolution, in: Proceedings of the 27th Annual ACM Symposium on Applied Computing, ACM, 2012, pp. 1387–1392.

[58] E.-N. Dragoi, S. Curteanu, A.-I. Galaction, D. Cascaval, Optimization methodology based on neural networks and self-adaptive differential evolution algorithm applied to an aerobic fermentation process, Applied Soft Computing 13 (1) (2013) 222–238.

[59] D. Sidhu, J. Dhillon, D. Kaur, Hybrid heuristic search method for design of digital IIR filter with conflicting objectives, Soft Computing 21 (12) (2017) 3461–3476.

[60] T. J. Choi, J.-H. Lee, H. Y. Youn, C. W. Ahn, Adaptive Differential Evolution with Elite Opposition-Based Learning and its Application to Training Artificial Neural Networks, Fundamenta Informaticae 164 (2-3) (2019) 227–242.

[61] S. Rahnamayan, H. R. Tizhoosh, M. M. Salama, Opposition-based differential evolution (ODE) with variable jumping rate, in: Foundations of Computational Intelligence, 2007. FOCI 2007. IEEE Symposium on, IEEE, 2007, pp. 81–88.

[62] A. Esmailzadeh, S. Rahnamayan, Opposition-based differential evolution with protective generation jumping, in: Differential Evolution (SDE), 2011 IEEE Symposium on, IEEE, 2011, pp. 1–8.

[63] S. Rahnamayan, H. R. Tizhoosh, M. M. Salama, Quasi-oppositional differential evolution, in: Evolutionary Computation, 2007. CEC 2007. IEEE Congress on, IEEE, 2007, pp. 2229–2236.

[64] M. Ergezer, D. Simon, D. Du, Oppositional biogeography-based optimization, in: Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on, IEEE, 2009, pp. 1009–1014.

[65] Q. Xu, L. Wang, B. He, N. Wang, Modified opposition-based differential evolution for function optimization, Journal of Computational Information Systems 7 (5) (2011) 1582–1591.

[66] H. Wang, Z. Wu, S. Rahnamayan, Enhanced opposition-based differential evolution for solving high-dimensional continuous optimization problems, Soft Computing 15 (11) (2011) 2127–2140.

[67] X. Zhou, Z. Wu, H. Wang, Elite opposition-based differential evolution for solving large-scale optimization problems and its implementation on GPU, in: Parallel and Distributed Computing, Applications and Technologies (PDCAT), 2012 13th International Conference on, IEEE, 2012, pp. 727–732.

[68] H. Liu, Z. Wu, H. Wang, S. Rahnamayan, C. Deng, Improved differential evolution with adaptive opposition strategy, in: 2014 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2014, pp. 1776–1783.

[69] B. Lacevic, E. Almaidi, Ectropy of diversity measures for populations in Euclidean space, Information Sciences 181 (11) (2011) 2316–2339.

[70] R. K. Ursem, Diversity-guided evolutionary algorithms, in: International Conference on Parallel Problem Solving from Nature, Springer, 2002, pp. 462–471.

[71] M. A. Bedau, M. Zwicky, A. Bahm, Variance and uncertainty measures of population diversity dynamics, Advances in Systems Science and Applications (1995) 7.

[72] J. Riget, J. S. Vesterstrom, A diversity-guided particle swarm optimizer the ARPSO, Dept. Comput. Sci., Univ. of Aarhus, Aarhus, Denmark, Tech. Rep 2 (2002) 2002.

[73] J. Jie, J. Zeng, Particle swarm optimization with diversity-controlled acceleration coefficients, in: Third International Conference on Natural Computation (ICNC 2007), Vol. 4, IEEE, 2007, pp. 150–154.

[74] C. Lin, A. Qin, Q. Feng, A comparative study of crossover in differential evolution, Journal of Heuristics 17 (6) (2011) 675–703.

[75] D. Zaharie, Influence of crossover on the behavior of differential evolution algorithms, Applied soft computing 9 (3) (2009) 1126–1138.

[76] J. Demšar, Statistical comparisons of classifiers over multiple data sets, Journal of Machine learning research 7 (Jan) (2006) 1–30.

[77] J. Zhang, A. C. Sanderson, JADE: adaptive differential evolution with optional external archive, IEEE Transactions on evolutionary computation 13 (5) (2009) 945–958.