Swarm and Evolutionary Computation 68 (2022) 101010

Contents lists available at ScienceDirect

Swarm and Evolutionary Computation

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

Original research

Differential evolution with orthogonal array-based initialization and a novel selection strategy

Abhishek Kumar[a], Partha P. Biswas[b,*], Ponnuthurai N. Suganthan[c]

[a] Department of Electrical Engineering, Indian Institute of Technology (BHU), Varanasi, Varanasi, 221005, India
[b] Advanced Digital Sciences Center, Singapore 138602, Singapore
[c] School of Electrical Electronic Engineering, Nanyang Technological University, Singapore 639798, Singapore

ARTICLE INFO

Keywords:
Differential evolution
Orthogonal array-based initialization
Neighborhood search
Conservative selection
Parameter adaptation technique

ABSTRACT

Differential evolution (DE) has been a simple yet effective algorithm for global optimization problems. The performance of DE highly depends on its operators and parameter settings. In the last couple of decades, many advanced variants of DE have been proposed by modifying the operators and introducing new parameter tuning methods. However, the majority of the works on advanced DE have been concentrated upon the mutation and crossover operators. The initialization and selection operators are less explored in the literature. In this work, we implement the orthogonal array-based initialization of the population and propose a neighborhood search strategy to construct the initial population for the DE-based algorithms. We also introduce a conservative selection scheme to improve the performance of the algorithm. We analyze the influence of the proposed initialization and selection schemes on several variants of DE. Results suggest that the proposed methods highly improve the performance of DE algorithm and its variants. Furthermore, we introduce an ensemble strategy for parameter adaptation techniques in DE. Incorporating all the proposed initialization, selection, and parameter adaptation strategies, we develop a new variant of DE, named OLSHADE-CS. The performance of OLSHADE-CS is found to be highly competitive and significantly better in many cases when compared with the performance of the state-of-the-art algorithms on CEC benchmark problems.

1. Introduction

In this paper, we evaluate the global bound-constrained optimization problems. Without losing the generality, a global bound-constrained optimization problem is mathematically defined as follows:

$$\bar{x}^* = \text{argmin } f(\bar{x}),\ \bar{x} \in \Omega,\ \bar{x} = [x_1, x_2, \ldots, x_D]^T,\ L_j \le x_j \le U_j \quad (1)$$

where $f(.)$ represents the objective function of the given problem; D is the number of decision-variables in the given problem; and $\bar{x}$ is a D-dimensional vector with real-parameter elements in search-space Ω. Here, Ω represents the D-dimensional search space where j-th dimension is bounded by $[L_j, U_j]$. In Eqn. (1), we describe the optimization problem as a minimization problem. Moreover, we consider the mathematical expressions for only minimization problems throughout the paper. Since the last three decades, research on single objective bound-constrained optimization problems has been the basis on more complex optimization areas such as multimodal, constrained, multi-, many-, dynamic optimizations, etc [1]. Outcomes of the research done on the bound-constrained optimization problems have directly influenced the development of the above-mentioned optimization areas [2]. The research area has been evolving rapidly in recent years, and numerous bound-constrained optimization algorithms have been proposed [3]. However, this research avenue is still open for further exploration as a single algorithm cannot always perform satisfactorily on different optimization problem classes [4]. Real-world optimization problems bring forth numerous challenging attributes to the algorithms such as a large number of non-separable variables, asymmetric distribution of the local minima, high multimodality, a mixture of aforementioned characteristics, etc [5].

The problem defined in Eqn. (1) can be classified into two groups on the basis of the budget of function evaluations. The first group includes problems with a small budget of function evaluations ($100 \times D$), called *expensive optimization problems*. On the other hand, problems with a relatively large budget of function evaluations have been included in the second group, denoted as *inexpensive optimization problems*. Inexpensive optimization problems with larger budget for function evaluations (defined in CEC2020 competition [6]) are considered in this paper. Although the No Free Lunch theorem [4] states that an optimization algorithm can not always perform better than other algorithms on all problems, it is possible to design an algorithm that works better than others on a specific class of problems [7]. With a larger budget for func-

* Corresponding author.:.
E-mail addresses: partha.b@adsc-create.edu.sg (P.P. Biswas), epnsugan@ntu.edu.sg (P.N. Suganthan).

https://doi.org/10.1016/j.swevo.2021.101010
Received 25 January 2021; Received in revised form 22 October 2021; Accepted 25 October 2021
Available online 10 November 2021

tion evaluations in inexpensive optimization problems, state-of-the-art algorithms cannot provide an appropriate balance between exploration and exploitation [8]. These algorithms are principally designed for low-budget inexpensive problems, where convergence should be faster to get an optimum solution within the budget [9]. Generally, exploratory search should be emphasized during the initial phase of the optimization process, while the exploitative search during the final stage [10]. In recent time, the focus of optimization has gradually shifted more towards exploitative search as the optimization process evolves [11]. Therefore, even for high-budget problems, the optimization process switches to exploitative search far early in the algorithms. Thus, they easily get trapped inside the local basins of the problems, and the final results of the algorithms may become non-optimal [12]. Moreover, these algorithms cannot utilize the high budget appropriately to find the global optimum solution. To deal with such issues, we develop a DE-based algorithm to solve high-budget inexpensive problems effectively and robustly.

DE, one of the popular derivative-free population-based optimization algorithms, is used to solve problems defined in Eqn. (1). DE is one of the most popular optimization tools for real parameter optimization problems as it is compact with simple structure, easy to use and understand, and robust with moderate convergence. Although DE has adopted the common concepts of the natural evolutionary processes, it has some unique characteristics among the members of the family of evolutionary algorithms [13]. The main characteristics of DE are (i) ways to generate trial solutions (offsprings) from solutions of the current population (parents), and (ii) selection procedure to create a new population for the next iteration. DE uses a one-to-one selection and spawning relationship between each individual (parent) and its trial individual (offspring). Although this feature of selection provides strength in DE, sometimes, it turns into weakness when the global optimal solution is located between narrow boundaries. While selection mechanisms used in other algorithms can possibly be adopted to improve the performance of DE, very few attempts have been made in the past on this aspect.

Further, several efforts have been made in the last few decades to enhance the application of DE over different kinds of problems, and the quest is still there due to its versatile behavior [14]. Since the introduction of DE in seminal paper [15], numerous modifications have been proposed to enhance the performance. Research progress in the enhancement of DE can be found in [16–23]. Some of the research directions adopted in these attempts are: 1) Parameter adaptation technique based on learning from past experiences, novel crossover and mutation schemes, the ensemble of various mutation and crossover schemes, and population resizing during the course of the search. Moreover, DE-based algorithms have been successfully applied to the constrained, multimodal and high-dimensional optimization problems. In recent years, DE algorithms have also been widely applied to many real-world complex optimization problems in the domains of chemical engineering [24], wireless sensor networks [25], electrical networks [26,27], general engineering design [28], computer vision [29], data clustering [30], etc. In these research problem areas, most of the DE-based algorithms are coupled with other schemes due to DE's simple framework yet robust performance over complex search space. From the above discussion, it is evident that DE-based variants have created a class of popular and robust algorithms for diverse optimization fields. These observations motivate the present work. Moreover, contrary to the earlier approaches made on the enhancement of DE, we propose to modify the selection operator along with other operators.

A one-to-one selection scheme has been used in the classical DE algorithm to determine the next set of solutions for the ongoing iterations. In this scheme, each individual of the population retains the best solution at its index-point, and the population also retains the best-so-far solution. However, this scheme traces the improvement at only the individual level, and the improvement of other individuals does not affect the selection of any individual's trial solution. For example, a trial solution is selected when it is better than an individual's current solution, even if it is worse than some solutions currently available in the population. Due to this phenomenon, the performance of DE may deteriorate in highly multimodal problems. To address this issue, we propose a new conservative selection procedure to select a trial solution in place of the one-to-one selection procedure. In the proposed selection procedure, a sub-population is created by selecting solutions from the current population and trial population for each trial solution. A trial solution is only selected in the current population when it satisfies the following criteria:

- The trial solution must be better than the corresponding current solution.
- The trial solution must be better than $p\%$ of the solutions of its sub-population.

Here, the size of sub-population and p are the user-defined parameters of the proposed selection scheme. By adding one more criterion in the selection scheme of classical DE, this scheme promotes efficient exploration without losing the exploitation capability in the population. In this work, we employ this selection scheme in popular DE variants to study its effectiveness.

As mentioned earlier, a review of literature on the variants of DE reveals that the improvements focused mainly on the adaptation of parameters (scale factor sF and crossover rate CR) and mutation strategies. Other operators like initialization and selection have received minimal attention. In DE, the random initialization should supposedly distribute the population members uniformly over the entire search space. However, the 'random uniform' generation of solutions does not produce accurate uniform distribution. The population does not cover the entire search space either as the population size is usually small. Therefore, to distribute the initial solutions over the entire search space, we propose an orthogonal array and neighborhood search-based initialization operator for DE. In this approach, a large number of solutions are initialized by using the orthogonal array-based design. Subsequently, we apply neighborhood search on each solution for some iterations to distribute these solutions on lower objective function contours of search space. After that, we pick a subset of better solutions from all solutions as an initial population for DE. This initialization process improves the performance of DE as the initial population is more concentrated near the optimal locations in the search space. By consolidating the proposed schemes within the framework of DE, we propose a new algorithm, named OLSHADE-CS, to solve real parameter bound-constrained optimization problems. In this algorithm, we incorporate the following operations:

1. Orthogonal array and neighborhood search-based initialization to generate the initial population for further processing in DE operations.
2. An improved version of the parameter adaptation technique for dynamic tuning of parameters during the search.
3. A conservative selection scheme to select the solutions from current and trial populations for processing in the next iteration.
4. An ensemble of several mutation strategies for generating trial solutions.

2. Primordial information and literature survey

In this section, we present a brief survey on DE algorithm and its evolution in the last few decades.

2.1. Differential evolution

DE is a population-based global optimization algorithm that uses differential vectors to generate trial solutions [17,18]. In general, DE uses four evolutionary operators: *initialization, mutation, crossover*, and *selection* to update the population sequentially for finding the optimal solution.

2.1.1. Initialization

DE initializes optimization with N_p nos. D-dimensional solutions, called population, where N_p is the population's size, and D is the number of decision variables. Population, P, can be represented as a matrix of size $(N_p \times D)$, i.e.,

$$P = \left[\bar{x}_1, \bar{x}_2, \ldots., \bar{x}_{N_p}\right]^T \tag{2}$$

where,

$$\bar{x}_i = \left[x_{i1}, x_{i,2}, \ldots., x_{i,D}\right],\ i = 1, 2, \ldots, N_p \tag{3}$$

Here, the initial population is generated at random positions of search space uniformly using the following equation.

$$x_{ij} = x_L + rand.(x_U - x_L),\ i = 1, 2, \ldots., N_p \text{ and } j = 1, 2, \ldots., D \tag{4}$$

where $rand$ represents a random number generated from a uniform distribution within range (0,1), x_L and x_U are lower and upper bounds of j-th dimension of search space, respectively.

2.1.2. Mutation

In this step, a mutation strategy is utilized to generate a mutant vector for each solution of the current population. Numerous mutation strategies have been proposed. The most popular one, DE/rand/1, incorporates the following mutation strategy to generate mutant vectors.

$$\bar{v}_i = \bar{x}_{r_1} + sF.\left(\bar{x}_{r_2} - \bar{x}_{r_3}\right) \tag{5}$$

where sF is the user-defined parameter, called scaling factor, to scale the difference vector $\left(\bar{x}_{r_2} - \bar{x}_{r_3}\right)$. Some other popular mutation strategies are listed below.

1. DE/rand/2

$$\bar{v}_i = \bar{x}_{r_1} + sF.\left(\bar{x_{r_2}} - \bar{x_{r_3}} + \bar{x_{r_4}} - \bar{x_{r_5}}\right) \tag{6}$$

2. DE/rand-to-best/1

$$\bar{v}_i = \bar{x}_{r_1} + sF.\left(\bar{x}_{best} - \bar{x}_{r_1} + \bar{x}_{r_2} - \bar{x}_{r_3}\right) \tag{7}$$

3. DE/current-to-best/1

$$\bar{v}_i = \bar{x}_i + sF.\left(\bar{x}_{best} - \bar{x}_i + \bar{x}_{r_1} - \bar{x}_{r_2}\right) \tag{8}$$

4. DE/current-to-rand/1

$$\bar{v}_i = \bar{x}_i + rand.\left(\bar{x}_{r_1} - \bar{x}_i\right) + sF.\left(\bar{x}_{r_2} - \bar{x}_{r_3}\right) \tag{9}$$

where $\bar{x}_{r_1}$, $\bar{x}_{r_2}$, $\bar{x}_{r_3}$, $\bar{x}_{r_4}$, and $\bar{x}_{r_5}$ are mutually different solutions selected from the current population and $\bar{x}_{best}$ is the best solution found so far.

2.1.3. Crossover

In this step, a crossover operator is employed to share elements of each pair $(\bar{v}_i, \bar{x}_i)$ for creating trial solutions. The frequently used uniform or binomial crossover is described in the following equations.

$$\bar{u}_i = \left[u_{i1}, u_{i2}, \ldots., u_{iD}\right]^T \tag{10}$$

and

$$u_{ij} = \begin{cases} v_{ij}, \text{ if } rand_j < CR \text{ or } j == j_{rand} \\ x_{ij}, \text{ otherwise} \end{cases} \tag{11}$$

where $\bar{u}_i$ is the i-th trial solution created after crossover between $\bar{x}_i$ and $\bar{v}_i$; CR represents the user-defined parameter, called crossover rate, to control the number of crossover elements in trial solutions; j_{rand} is randomly chosen within $[1, D]$.

2.1.4. Selection

Selection is the final step of the optimization cycle in DE. In this step, the population for the next iteration $(k+1)$ is created from the current population and trial solutions. A better one is selected from each pair $(\bar{x}_i, \bar{u}_i)$ as a solution for the next population. Mathematically, the selection operator is defined by the following equation:

$$\bar{x}_i^{k+1} = \begin{cases} \bar{x}_i^k, \text{ if } f(\bar{x}_i^k) < f(\bar{u}_i^k) \\ \bar{u}_i^k, \text{ otherwise} \end{cases} \tag{12}$$

2.2. Orthogonal array

The orthogonal array (oa), denoted by $L_Q(N^M)$, is a predefined table with M factors and N levels per factor, where the size of $L_Q(N^M)$ is $(Q \times M)$ [31,32]. For example

$$L_4(2^3) = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 2 \\ 2 & 1 & 2 \\ 2 & 2 & 1 \end{bmatrix} \tag{13}$$

where $L_4(2^3)$ has four combinations of levels, two levels per factor and three factors. The properties of $L_Q(N^M)$ are as follows:

1. Each level comes Q/N times in each column of $L_Q(N^M)$.
2. Each combination of two corresponding levels of any two columns comes Q/N^2 times.
3. In any two-column of $L_Q(N^M)$, there are Q combinations: (1,1), (1,2), ..., $(1, Q)$, (2,1), (2,2), ..., $(2, Q)$, $(Q, 1)$, $(Q, 2)$, ..., (Q, Q).
4. The resulting array after swapping the two columns of an orthogonal array is also an orthogonal array.
5. After taking away some columns of any orthogonal array, the resulting array is also an orthogonal array having a smaller number of factors.

Due to the above properties, all combinations of factors are distributed uniformly over the search space of all feasible combinations [33].

2.3. Ensemble of mutation strategies based variants of differential evolution

The performance of DE has been highly dependent upon parameter settings and mutation strategies [19,34]. Numerous works have been carried out to identify the optimal combination for particular optimization applications. Although many DE variants had been proposed in the literature, some studies also reported that a specific strategy could not perform satisfactorily over all types of problems [30]. As a consequence of the shortcomings of DE variants with a single mutation strategy, an ensemble of mutation strategies has been getting more attention in the community since the last decade [19,35].

Fan *et al.* [36] propose an auto-selection mechanism during the evolutionary search for optimizing combinatorial problems. An adaptive operator selection method is introduced by Sallem *et al.* [37] to select a suitable mutation strategy using the functions landscape information and success history of all mutation strategies available in the selection pool. In [38], Elsayed *et al.* apply a fuzzy rule-based heuristic to select the best performing evolutionary algorithm from the pool of many algorithms during the evolutionary process.

An ensemble of discrete variants of DE is developed by Tasgetiren *et al.* [39] to solve the traveling salesman problems. For the DE framework, an ensemble of 16 combinations of mutation, crossover, and constraint handling mechanisms is proposed by Elsayed *et al.* [40] to solve constrained optimization problems effectively. In [19], a multi-population-based framework is designed to select DE variants using their performance characteristics. Zhang *et al.* [41] propose a multi-layer competitive-cooperative framework to enhance the performance of DE over global optimization problems. Another variant of the DE algorithm is proposed in [42], where three mutation strategies with their associated control parameters are employed to improve the balance between exploration and exploitation. Yu *et al.* [43] introduce a novel mutation mechanism to accelerate convergence and maintain diversity. Similarly, a triangular mutation operator is proposed by Mohamed [44] to provide a balance between explorative and exploitative search in DE. Interested readers can refer to the articles by Das *et al.* [18], Wu *et al.* [45], and a recent survey paper by Wu *et al.* [46].

2.4. Initialization techniques in evolutionary algorithms

For generating the initial population, evolutionary algorithms (EAs) utilize a pseudo-random number generator, where a sequence of the random numbers is generated from the uniform distribution [47]. Although this initialization technique is simple, it encounters difficulties where the search space dimension is high as the generated sequence of random numbers may not be fully distributed [48]. The initial population generated from uniform distribution can easily be transformed into a biased population [48]. Some EAs incorporated with biased randomly generated population have been proposed in the literature [49,50].

In addition to a uniform distribution, initialization techniques based on chaotic theory [51] have also been utilized to generate initial population in genetic algorithm (GA) [52], DE [53,54], artificial bee colony (ABC) [55], and particle swarm optimization (PSO) [52,54,56,57]. As shown in these works, use of chaotic-based initialization techniques enhances the performance of EAs in terms of convergence speed, success rate, and population diversity [58]. However, these techniques have many disadvantages. One major disadvantage is that these methods are devised only for one-, two-, or three-dimensional search space, not for higher dimensions [59]. An investigation is required to examine the performance of these initialization techniques on high-dimensional search spaces.

Besides aforementioned stochastic techniques, deterministic techniques have also been utilized to generate the initial population in EAs. These techniques always generate the same population, where the sequence of generated points is evenly distributed over the entire search space [60]. Recently, these techniques are gaining popularity as the uniformly distributed initial population can improve the exploration ability of EAs in case of inadequate prior knowledge about the search space [61]. Consequently, the algorithm converges to a better solution with a faster convergence speed [49]. Deterministic techniques, such as uniform experimental design and orthogonal design, generate points that are evenly distributed in a given range of search space [62]. However, developing points for a large population is expensive in uniform experimental design. Thus, the orthogonal design-based initialization schemes are preferred by the researchers. In the literature, these orthogonal design-based schemes improve the performance of several popular algorithms, such as DE [63,64], PSO [65,66], and GA [67,68].

Apart from these techniques, Latin hypercube sampling (LHS) [69] and opposition-based learning (OBL) [70] initialization techniques are also adopted in EAs. In LHS, variables are divided into a fixed number of intervals to create grids. Thereafter, a random number is generated within each grid. In OBL, the population can be initialized using any of the above techniques; however, a heuristic operator is employed to calculate opposite solutions from each population solution. For more information about the initialization schemes employed in EAs, interested readers can follow the survey paper [47].

3. The proposed algorithm

This section introduces the OLSHADE-CS algorithm which employs orthogonal array-based initialization, an ensemble of four mutation strategies, a parameter adaptation technique, and a conservative selection scheme to solve global optimization problems.

3.1. Orthogonal array-based initialization

Our proposed algorithm uses orthogonal array-based initialization to generate the initial population members. When solving an optimization problem, we do not know the location of the global optimal solution in the beginning. Therefore, we need to generate initial solutions uniformly such that the algorithm can explore the whole search space evenly. An orthogonal array is able to provide combinations of locations that are distributed evenly. This property of orthogonal array inspires us to utilize orthogonal array for constructing initial solutions.

3.1.1. Orthogonal array calculation

In this work, we utilize the procedure suggested in [71] to construct an orthogonal array, $L_Q(N^M)$, where N is an odd integer and $Q = N^I$. Here, I is chosen in such a manner that it satisfies the following equation.

$$N = \frac{Q^I - 1}{Q - 1} \tag{14}$$

In this work, the following steps are used to construct the orthogonal array, **A**.

- **Step 1:** Calculation of basic elements of **A**:

$$\alpha_{ij} = \lfloor * \rfloor \frac{i-1}{N^{I-k}},\ k = 1 \text{ to } I \text{ and } i = 1 \text{ to } Q \tag{15}$$

where,

$$j = \frac{N^{k-1} - 1}{N - 1} + 1$$

- **Step 2:** Calculation of nonbasic elements of **A**:

$$\boldsymbol{\alpha}_{j+(s-1)(N-1)+t} = \mathrm{rem}\left(\boldsymbol{\alpha}_s t + \boldsymbol{\alpha}_j, N\right),\ s = 1 \text{ to } (j-1) \text{ and } t = 1 \text{ to } (N-1) \tag{16}$$

where,

$$\boldsymbol{\alpha}_j = \left[\alpha_{1j}, \alpha_{2j}, \ldots., \alpha_{Qj}\right]^T$$

- **Step 3:** Deletion of last $(M - D)$ columns of **A** to restrict the matrix up to D columns.
- **Step 4:** Deletion of randomly selected $(N_p - Q)$ rows of **A** to restrict the matrix up to N_p rows.

After calculating matrix **A**, initial solutions are generated using the following equation.

$$x_{ij} = \alpha_{ij}\left(\frac{x_U - x_L}{\max(\mathbf{A}) - \min(\mathbf{A})}\right) + x_L,\ i = 1, 2, \ldots., N_p \text{ and } j = 1, 2, \ldots., D \tag{17}$$

3.1.2. Neighborhood search

In this step, we shift the location of each orthogonal initialized solution within its neighborhood. For this purpose, we adopt the neighborhood search proposed in [72]. However, we employ this search operator for 20% of the available budget of function evaluations in place of 60% suggested in [72]. In [72], neighborhood search is applied as a local search operator to improve the solutions. In contrast, we utilize this operator to create initial solutions with a random shift from orthogonal initialization for the proposed search operator. Following steps are utilized in neighborhood search [72]:

- **Step 1:** We create a neighborhood for each solution. The neighborhood members are defined as the nearest n_b solutions based on the euclidean distance.
- **Step 2:** We update the solutions according to the DE operators proposed in [73] within the neighborhood members for 20% of the available function evaluation budget.

3.2. Mutation strategies

In the proposed algorithm, we utilize an ensemble of four different mutation strategies to evolve the solutions in each iteration.

- *mut1:* DE/current-to-$best_1$/1 with external archive [74]:

$$\bar{v}_i = \bar{x}_i + sF_i\left(\bar{x}_{best_1} - \bar{x}_i + \bar{x}_{r1} - \bar{x}_{r3}\right) \tag{18}$$

- *mut2:* DE/current-to-$best_1$/1 without external archive [74]:

$$\bar{v}_i = \bar{x}_i + sF_i\left(\bar{x}_{best_1} - \bar{x}_i + \bar{x}_{r1} - \bar{x}_{r2}\right) \tag{19}$$

- *mut3:* DE/$best_1$/1 with external archive:

$$\bar{v}_i = \bar{x}_{best_1} + sF_i\left(\bar{x}_{r2} - \bar{x}_{r3}\right) \tag{20}$$

- *mut4:* DE/scaled $best_2$/1 with external archive:

$$\bar{v}_i = sF_i \bar{x}_{best_2} + \left(\bar{x}_{r2} - \bar{x}_{r3}\right) \tag{21}$$

where $r_1 \neq r_2 \neq r_3 \neq best_1 \neq best_2 \neq i$ are randomly generated integers in such a manner that *i)* $\bar{x}_{r1}$ and $\bar{x}_{r2}$ are selected from P, *ii)* $\bar{x}_{r3}$ is selected from the union of P and external archive, *iii)* $\bar{x}_{best1}$ is selected from the 25% of best solutions, and *iv)* $\bar{x}_{best2}$ is selected from the 50% of best solutions. Here, we employ an archive to improve the diversity of solutions by utilizing successful trial solutions of past iterations for generating new trial solutions [74].

For generating a trial solution for the i-th individual, we select one mutation strategy from the ensemble pool based on the success of each mutation strategy in the last iteration. According to each mutation strategy's relative success, we calculate the probability of selection of a mutation strategy, $Prob_i$, for the next iteration.

$$Prob_i^{k+1} = \frac{Succ_i^k}{\sum_{j=1}^{4} Succ_j^k}, \; i = \{1,2,3,4\} \tag{22}$$

where,

$$Succ_i^k = \sum_{j \in S_i^k} \max\left\{0, \frac{f(\bar{x}_j^k) - f(\bar{x}_j^{k+1})}{\left|f(\bar{x}_j^k)\right|}\right\} \tag{23}$$

and S_i^k is the set of individuals which use i-th mutation strategy at k-th iteration.

3.3. Crossover

After mutation, a crossover operation is done to generate trial solutions. In the proposed algorithm, two popular crossover strategies: uniform and exponential crossover, are employed randomly on each mutant solution. The uniform crossover, Xor_{bin}, is applied with a probability of 0.4 while the exponential crossover, $Xor_{\exp}$, is applied with a probability of 0.6.

$$\bar{u}_i = \begin{cases} Xor_{bin}\left(\bar{v}_i, \bar{x}_i\right), & \text{if } rand \leq 0.4 \\ Xor_{exp}\left(\bar{v}_i, \bar{x}_i\right), & \text{otherwise} \end{cases} \tag{24}$$

where, in $Xor_{bin}\left(\bar{v}_i, \bar{x}_i\right)$, $\bar{u}_i$ is generated using the following equation:

$$u_{ij} = \begin{cases} v_{ij}, & \text{if } rand_j < CR_j \text{ or } j == l \\ x_{ij}, & \text{otherwise} \end{cases} \tag{25}$$

where l is an integer randomly generated between 1 to D. Similarly, in $Xor_{\exp}\left(\bar{v}_i, \bar{x}_i\right)$, $\bar{u}_i$ is generated using the following equation:

$$u_{ij} = \begin{cases} v_{ij}, & \text{if } j \in \left\{\langle l \rangle_D, \langle l+1 \rangle_D, \ldots, \langle l+L-1 \rangle_D\right\} \\ x_{ij}, & \text{otherwise} \end{cases} \tag{26}$$

where $\langle l \rangle_D$ represents a remainder function where this function returns a remainder left over when l is divided by D.

3.4. Selection

A new conservative selection procedure is proposed to select a trial solution instead of the one-to-one selection procedure. In the approach, for each trial solution, a neighborhood, $N_{s,i}$, of size ns is formed by randomly selecting solutions from the union of the current population and trial population. For the initial 60% of the optimization process, a trial solution is selected to create the population member of subsequent iteration if it is better than the current solution and 25% of its neighborhood's solutions, i.e.,

$$\bar{x}_i^{k+1} = \begin{cases} \bar{u}_i^k, & \text{if } \left(f(\bar{u}_i^k) \leq f(\bar{x}_i^k)\right) \wedge \left(\frac{1}{ns}\sum_{j \in N_{s,i}} S_{i,j} > 0.25\right) \\ \bar{x}_i^k, & \text{otherwise} \end{cases} \tag{27}$$

where,

$$S_{i,j} = \begin{cases} 1, & \text{if } f(\bar{u}_i^k) \leq f(\bar{x}_j^k) \\ 0, & \text{otherwise} \end{cases}, \; j \in N_{s,i} \tag{28}$$

For the rest of the optimization process, trial solutions are selected using the conventional selection scheme defined in Eqn. (12).

3.5. Parameter adaptation technique

The parameter adaptation technique used here is a modified version of the proposed one in [73], called success-history-based parameter adaptation technique (SHPAT), for the adaptation of parameters CR_i and sF_i. In SHPAT, a solution $\bar{x}_i$ is associated with CR_i and sF_i set probabilistically at the initial step of each iteration. Here, a historical memory with H elements for both parameters named μ_{CR} and μ_{SF}, respectively, is maintained for adapting the value of CR_i and sF_i during the search. The values of CR_i and sF_i are dependent upon μ_{CR} and μ_{sF}, respectively, and are calculated using the following equations:

$$\begin{cases} CR_i = randn_i\left(\mu_{CR,r_i}, 0.1\right) \\ sF_i = randc_i\left(\mu_{sF,r_i}, 0.1\right) \end{cases} \tag{29}$$

where, r_i is randomly selected index from $[1, H]$; $randn_i(\mu_{CR}, \sigma_{CR})$ and $randc_i(\mu_{sF}, \sigma_{sF})$ are random number generators from Gaussian and Cauchy distributions, respectively. For Gaussian distribution, mean and standard deviation are μ_{CR} and σ_{CR}, respectively; while for Cauchy distribution location and scale parameters are μ_{sF} and σ_{sF}, respectively. If CR_i is generated outside of the range [0,1], then it is replaced by the violated limit (0 or 1). Similarly, sF_i greater than 1 is set to 1, and for $sF_i \leq 0$, sF_i is repeatedly generated until we get a positive number.

In the algorithm, suitable initial values of both μ_{CR} and μ_{sF} are set. To update one element in μ_{CR} and μ_{sF} sequentially, CR_i and sF_i of each i-th individual that finds a better trial solution than itself are recorded in S_{CR} and S_{sF} at the end of each iteration. Moreover, elements of S_{CR} and S_{sF} are sorted according to the degree of improvement of the solutions. If S_{CR} and S_{sF} are not empty, $\mu_{CR,k}$ and $\mu_{sF,k}$ are updated using the following equations:

$$\begin{cases} \mu_{CR,k} = \text{mean}_w\left(S_{CR}\right) \\ \mu_{sF,k} = \text{mean}_w\left(S_{sF}\right) \end{cases} \tag{30}$$

where index k ($\in \{1,2,3,\ldots,H\}$) is the memory position updated in the current iteration, and it is calculated as follows:

$$k = \text{rem}(t-1, H) + 1 \tag{31}$$

where t is the current iteration count and $\text{rem}(a,b)$ is the remainder after dividing a by b. Here, mean_w represents a weighted Lehmer mean calculated using the following equation:

$$\text{mean}_w(C) = \frac{\sum_{k=1}^{|C|} w_k C_k^2}{\sum_{k=1}^{|C|} w_k C_k}, \text{ where, } C \in \{S_{CR}, S_{sF}\} \tag{32}$$

and

$$w_k = \log(|C| + 0.5) - \log(k) \tag{33}$$

Note that in [73], the w_k is calculated differently. In [73], value to w_k is dependent on the magnitude of the objective function improvement of the successful offsprings, i.e.,

$$w_k = \frac{\triangle f_i}{\sum_{i=1}^{|C|} \triangle f_i} \tag{34}$$

where

$$\triangle f_i = abs\left(f\left(\bar{u}_i^k\right) - f\left(\bar{x}_i^k\right)\right)$$

However, this update procedure is prone to overestimation or underestimation due to the magnitude of the improvement. If parameter values cause large improvement in the objective function while the other settings result in small improvement, the latter would be ignored in the next $\text{mean}_w(C)$. To avoid such issues, we employ Eqn. (33) to calculate w_k, where the equation is rooted to the rank rather than to the magnitude of the improvement.

3.6. Archive and population management

To preserve information on the direction of progress during iterations, an archive N_{Arch} is maintained where some inferior solutions are stored [74]. Initially, this archive is empty. At the end of each iteration, the solution of the population, which is updated in the selection operation, is added to this archive. However, the size of the archive can exceed a pre-defined size limit after some iterations. Therefore, to keep the size within the limit, extra solutions are removed randomly from the archive. In the proposed algorithm, population size keeps changing from iteration to iteration [74]. Thus, the archive size also keeps changing to maintain an archive size proportional to the population size.

Here, we utilize a linear population reduction scheme [75] to resize the population in each iteration,i.e.,

$$N_p^k = \lfloor \frac{N_p^{min} - N_p^{init}}{Max_{nfes}} * nfes \rfloor + N_p^{init} \tag{35}$$

where $N_p^{\min}$ is set to 4; N_p^{init} represents the initial population size; Max_{nfes} is the maximum allowed function evaluations and $nfes$ represents the current count of function evaluations at k-th iteration. Similarly, archive size N_{Arch} is resized using the following equation:

$$N_{Arch}^k = \boldsymbol{\beta} N_p^k, \tag{36}$$

where, β is the multiplying factor which is the ratio of archive size and population size. Here, the archive provides information related to the progressive direction. Moreover, it also improves the population diversity.

3.7. Overall framework

Algorithm 1 shows the basic framework of OLSHADE-CS. Search operator utilizes orthogonal array-based initialization to generate initial solutions. In each iteration, parameters are calculated using the proposed parameter adaptation technique, trial solutions are generated using an ensemble of four mutation strategies, and selection of solutions is done using the proposed conservative selection procedure. These steps are repeated until one of the termination criteria is satisfied.

Algorithm 1: OLSHADE-CS.

Result: $\bar{x}_1$

$P^0 \preceq \left(\bar{x}_1^0, \bar{x}_2^0, ..., \bar{x}_{N_p}^0\right) \preceq \leftarrow \preceq$**OAInitialization**$\left(\bar{x}_U, \bar{x}_L\right)$;
$P^0 \preceq \leftarrow \preceq$**Sorted**$\left(P^0\right)$;
$\mu_{CR,1:H} \preceq \leftarrow \preceq 0.2, \preceq \mu_{sF,1:H} \preceq \leftarrow \preceq 0.6, \preceq Prob_{1:4} \preceq \leftarrow \preceq 0.25$;
$A \preceq \leftarrow \preceq \Phi, \preceq t \leftarrow \preceq 0, \preceq nfes \preceq \leftarrow 0.2 Max_{nfes}, \preceq t \leftarrow t+1$;
while *(Termination criteria are not met)* **do**
  $t \leftarrow t+1$;
  for $i = 1 \preceq to \preceq N_p$ **do**
    $j \leftarrow$**RandomlySelectIndex**$(1, H)$;
    $CR_i \leftarrow \max\{0, \min\{1, randn(\mu_{CR,j}, 0.1)\}\}$;
    $sF_i \leftarrow \min\{1, randc(\mu_{CR,j}, 0.1)\}$;
    while $sF_i \leq 0$ **do**
      $sF_i \leftarrow \min\{1, randc(\mu_{CR,j}, 0.1)\}$;
    end
  end
  $[U, Prob, S_{CR}, S_{sF}] \leftarrow$**TrialSolution**$(P, A, CR, sF, Prob)$;
  $nfes \leftarrow nfes + N_p$;
  $P \leftarrow$**ConservativeSelection**$(P, U, nfes)$;
  $[\mu_{CR}, \mu_{sF}] \leftarrow$**MemoryUpdate**$\left(S_{CR}, S_{sF}, t\right)$;
end

4. Experimental results and discussion

In this section, we conduct different experiments to evaluate the performance of OLSHADE-CS algorithm and its operators at different stages. To study the influence of various proposed operations in OLSHADE-CS, we evaluate the performance of each operation individually on benchmark problems of CEC2017 [2] and CEC2020 [6]. Thereafter, we investigate the performance of OLSHADE-CS on the problems of CEC2020 and compare it with many recent state-of-the-art algorithms. In this work, we consider single-objective bound-constrained optimization problems in the benchmark suites of CEC competitions. There are 30 and 10 test problems in CEC2017 and CEC2020 benchmark suites, respectively, with four different scales. Unless specified otherwise, we usually followed the guidelines depicted in the respective CEC competitions while performing the experiments. Results of all algorithms are stored, including the *best-of-the-run* error values over 51 independent runs for each problem with the guidelines suggested in [2]. The error value is calculated as $|f_{best} - f^*|$, where f_{best} is the objective function value of *best-of-the-run* solution (a solution with lower objective function value picked from all searched solutions). Bayesian statistical tests [76], such as Bayesian signed rank test (BST), Bayesian rank sum test (BRT), and Bayesian Friedman test (BFT), are conducted to verify whether the performance of two or more algorithms differ from each other in a statistically significant way. Most large result tables are reported in the supplementary file. These tables are denoted as S.# in this manuscript, where # is the table number reported in the supplementary file. All our experiments are implemented in MATLAB R2018b environment and executed on a PC with Windows 10, i7 3.3 GHz CPU, and 16 GB RAM.

4.1. Effectiveness of the proposed ensemble of mutation strategies

In this experiment, we demonstrate the effectiveness of the proposed ensemble of mutation strategies on the performance of OLSHADE-CS. We also analyze the performance of individual mutation strategy on a given problem. Therefore, we design four variants of the proposed algorithm with one of the four different mutation strategies mentioned in Section 3.2: 1) OLSHADE-CS-*mut1*, 2) OLSHADE-CS-*mut2*, 3) OLSHADE-CS-*mut3*, and 4) OLSHADE-CS-*mut4*. These algorithms are benchmarked on CEC2020 test-suite.

Statistical outcomes of these algorithms in terms of the mean and standard deviation of 'best-of-the-run' are reported in Table S.1. In the table, outcomes of the BRT are also provided to verify the superiority of OLSHADE-CS to other algorithms. As seen from the same table, OLSHADE-CS performs significantly better than other variants on majority of the problems. Summary of observations obtained from this experiment is as follows:

1) *5D problems:* All algorithms obtain the optimum solutions in all independent runs for problems F1, F5, F6, F7, and F8. OLSHADE-CS-*mut4* exhibits better performance on problem F9, where F9 is the composite function of four multimodal functions. For the remaining problems, in general, OLSHADE-CS is found to be the best.
2) *10D problems:* In all independent runs, all algorithms converge to the global optima for problems F1 and F4. OLSHADE-CS-*mut2* shows better performance on problem F7, a hybrid function of five highly multimodal functions. Further, OLSHADE-CE-*mut4* performs well on problem F8 where problem F8 is a composite function. For other composite functions (F9 and F10), the performance of all algorithms is statistically similar. Again, OLSHADE-CS shows better performance for the rest of the problems.
3) *15D problems:* Like the above cases, all algorithms find the near optimum solution in each run for problems F1 and F4. OLSHADE-CE-*mut4* outperforms others on the composite problem F8. All algorithms perform similarly on the composite function F9. The performance of only OLSHADE-CS-*mut2* is statistically similar to OLSHADE-CS in problem F10. For other problems, the overall performance of OLSHADE-CS is found to be the best.
4) *20D problems:* In this case, for problems F1 and F4, all algorithms find the near optimum solution in each run. OLSHADE-CS-*mut2* is the best among all algorithms on the hybrid problem F7. In the composite

Table 1
Mean rank of algorithms using Bayesian Friedman test (BFT).

Algorithm	5D	10D	15D	20D
OLSHADE-CS	**2.2727**	**2.0000**	**1.7727**	**1.9545**
OLSHADE-CS-*mut1*	3.3636	3.1364	3.1818	3.1364
OLSHADE-CS-*mut2*	2.7273	3.3182	3.3636	3.5000
OLSHADE-CS-*mut3*	3.4545	3.0455	3.1818	3.1364
OLSHADE-CS-*mut4*	3.1818	3.5000	3.5000	3.2727

problem F9, the performance of all algorithms is statistically similar. For the rest of the problems, OLSHADE-CS beats all other algorithms.

We also calculate the ranking of the algorithms based on the mean error value using BFT. The results of this test are reported in Table 1. As evident from the table, the ranking of the algorithms changes in different dimensions with the exception that OLSHADE-CS is the best among all in all dimensions. OLSHADE-CS utilizes an ensemble of all these mutation strategies while solving the problems. OLSHADE-CS performs better in all dimensions than these variants as the effectiveness and robustness of each mutation strategy are united in one the ensemble method. Moreover, the demerits of one mutation strategy is overcome by the other as a mutation strategy is chosen according to its success probability.

Furthermore, we perform BST to analyze the performance of each pair of algorithms, and the results of this test are presented in Fig 1. In this figure, *reddish* blocks represent that algorithm 1 (algorithms written in the vertical axis) performs better than Algorithm 2 (algorithms written in the horizontal axis), *blueish* blocks signify that Algorithm 2 performs better than algorithm 1, and *greenish* blocks represent that Algorithm 1 and Algorithm 2 perform similarly. The color density of the blocks is dependent upon the probability (mentioned in the blocks) of one algorithm doing better than the other. Similar conclusions, like in the above cases, are also drawn from this test. Each mutation strategy shows a significantly different performance from other in different dimensions. The performance of OLSHADE-CS is better than all variants based on these mutation strategies.

In conclusion, OLSHADE-CS performs better than other algorithms with individual mutation strategies when dealing with the CEC2020 benchmark suite. The BRT shows (Table S.1) that OLSHADE-CS is better than OLSHADE-CS-*mut1*, OLSHADE-CS-*mut2*, OLSHADE-CS-*mut3*, and OLSHADE-CS-*mut4* on - 1) five, four, five, and five problems, respectively, in 5-dimension problems; 2) five, four, five, and five problems, respectively, in 10-dimension problems; 3) five, seven, five, and six problems, respectively, in 15-dimension problems; and 4) six problems in each case in 20-dimension problems. OLSHADE-CS performs worse than OLSHADE-CS-*mut2* and OLSHADE-CS-*mut4* on one and two problems, respectively, in 5-dimension problems; while in 10-dimension, its statistically inferior performance to both is only on one problem. OLSHADE-CS-*mut4* betters OLSHADE-CS only in one problem in 15-dimension, while OLSHADE-CS-*mut2* betters it in one problem in 20-dimension.

4.2. Effectiveness of the proposed conservative selection scheme

In this paper, the selection operator of DE algorithm is made more conservative than the classical one to improve the performance on complex multimodal problems. Subsequently, this proposed conservative selection (CS) scheme is incorporated in OLSHADE-CS. Here, CS plays a critical role in the proposed algorithm. To validate its effectiveness, we design two algorithms after incorporating the cs scheme in LSHADE-Sin [77] and jSO [78] framework (top-ranked DE-based algorithms in IEEE CEC 2017 competition), called as LSHADE-Sin-CS and jSO-CS, respectively. The performance of LSHADE-Sin-CS and jSO-CS is compared with their counterparts on 30 test problems with dimensions = 10, 30, 50, and 100 in IEEE CEC 2017 test-suite [2]. For statistical comparison, we implement BRT test at a 0.05 significance level.

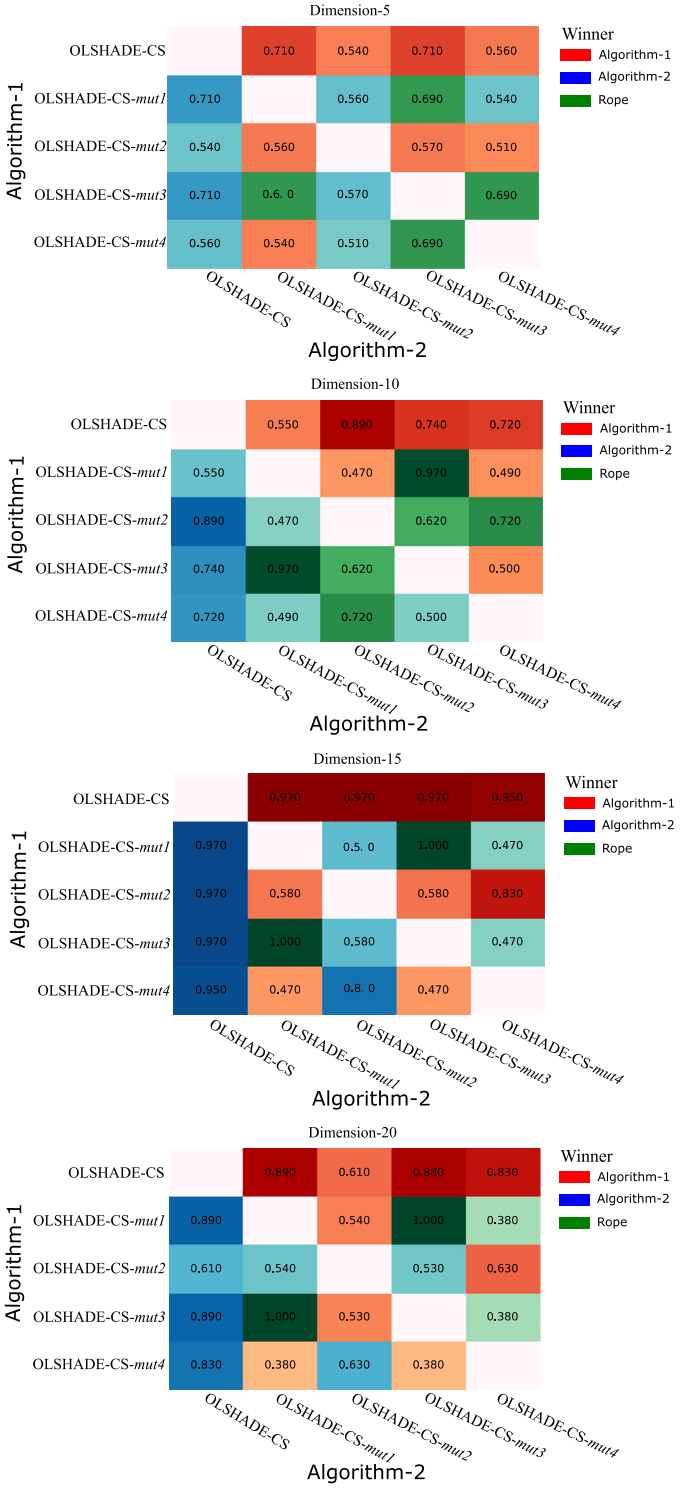

Fig. 1. Bayesian signed rank test (BST) of the algorithms.

We summarize the outcomes of this experiment in Tables S.2-S.5. In these tables, according to BRT, statistical significance test results are shown as "+/=/-", where '+', '-', and '=' represent that the proposed scheme or algorithm is better than, worse than, and significantly equivalent to the corresponding comparable algorithm, respectively. As shown in Tables S.2 and S.3, jSO-CS performs better than jSO in eight, ten, eleven, and thirteen out of 30 test problems with dimensions 10, 30, 50, and 100, respectively. However, jSO surpasses jSO-CS in one, four,

Algorithm 2: TrialSolution($P, A, CR, sF, Prob$).

Result: $U, Prob, S_{CR}, S_{sF}$
$S_{CR} \leftarrow \Phi, \preceq S_{sF} \leftarrow \Phi, \preceq Mut_{1:4} \leftarrow \Phi, \preceq Succ_{1:4} \leftarrow 0, \preceq U \leftarrow \Phi$;
for $i = 1 \preceq to \preceq N_p$ **do**
  $best_1 \leftarrow$ **RandomlySelected**$\left(1, \lfloor 0.25N_p \rfloor\right)$;
  $best_2 \leftarrow$ **RandomlySelected**$\left(1, \lfloor 0.5N_p \rfloor\right)$;
  $r_1 \leftarrow$ **RandomlySelected**$\left(1, N_p\right) \neq \{i, best_1, best_2\}$;
  $r_3 \leftarrow$ **RandomlySelected**$\left(1, N_p\right) \neq \{i, r_1, best_1, best_2\}$;
  $r_3 \leftarrow$ **RandomlySelected**$\left(1, |P^t \cup A^t|\right) \neq \{i, r_1, best_1, best_2\}$;
  if $rn < Prob_1$ **then**
    $\bar{v}_i \leftarrow \bar{x}_i + sF_i\left(\bar{x}_{best_1} - \bar{x}_i + \bar{x}_{r1} - \bar{x}_{r3}\right)$;
    $Mut_1 \leftarrow Mut_1 \cup i$;
  else if $Prob_1 \leq rn < (Prob_1 + Prob_2)$ **then**
    $\bar{v}_i \leftarrow \bar{x}_i + sF_i\left(\bar{x}_{best_1} - \bar{x}_i + \bar{x}_{r1} - \bar{x}_{r2}\right)$;
    $Mut_2 \leftarrow Mut_2 \cup i$;
  else if $(Prob_1 + Prob_2) \leq rn < (Prob_1 + Prob_2 + Prob_3)$ **then**
    $\bar{v}_i \leftarrow \bar{x}_{best_1} + sF_i\left(\bar{x}_{r2} - \bar{x}_{r3}\right)$;
    $Mut_3 \leftarrow Mut_3 \cup i$;
  else
    $\bar{v}_i \leftarrow sF_i\left(\bar{x}_{best_2} + \bar{x}_{r2} - \bar{x}_{r3}\right)$;
    $Mut_4 \leftarrow Mut_4 \cup i$;
  end
  if $rand(0,1) \leq 0.4$ **then**
    $\bar{u}_i \leftarrow Xor_{bin}\left(\bar{v}_i, \bar{x}_i, CR_i\right)$;
  else
    $\bar{u}_i \leftarrow Xor_{\exp}\left(\bar{v}_i, \bar{x}_i, CR_i\right)$;
  end
  if $f(\bar{x}_i) \leq f(\bar{u}_i)$ **then**
    $S_{CR} \leftarrow S_{CR} \cup CR_i$;
    $S_{sF} \leftarrow S_{sF} \cup sF_i$;
    $Succ_j \leftarrow Succ_j + 1, \preceq i \in Mut_j$;
  end
  $U \leftarrow U \in \bar{u}_i$;
end
if $\sum_{i=1}^{4} Succ_i \neq \Phi$ **then**
  $Prob_j \leftarrow \frac{Succ_j}{\sum_{i=1}^{4} Succ_i}$;
else
  $Prob_j \leftarrow 0.25$;
end

Algorithm 3: ConservativeSelection($P, U, nfes, A$).

Result: $[P, A]$
for $i = 1 \preceq to \preceq N_p$ **do**
  if $nfes \leq 0.6 Max_{nfes}$ **then**
    $N_s \leftarrow \Phi$;
    for $j \preceq to \preceq ns$ **do**
      $N_s \leftarrow N_s \cup \left\{\text{**RandomlySelected**}\left(1, N_p\right) \neq N_s\right\}$;
      if $f(\bar{u}_i) \leq f(\bar{x}_{N_{s,j}})$ **then**
        $S_j \leftarrow 1$;
      else
        $S_j \leftarrow 0$;
      end
    end
    if $\left(f(\bar{u}_i^k) \leq f(\bar{x}_i^k)\right) \wedge \left(\frac{1}{ns}\sum_{j=1}^{ns} S_j > 0.25\right)$ **then**
      $A \leftarrow A \cup \bar{x}_i$;
      $P \leftarrow (P \backslash \bar{x}_i) \cup \bar{u}_i$;
    end
  else
    if $\left(f(\bar{u}_i^k) \leq f(\bar{x}_i^k)\right)$ **then**
      $A \leftarrow A \cup \bar{x}_i$;
      $P \leftarrow (P \backslash \bar{x}_i) \cup \bar{u}_i$;
    end
  end
end
$P \leftarrow$ **Sorted**(P);
$N_p \leftarrow \lfloor \frac{N_p^{\min} - N_p^{init}}{Max_{nfes}} nfes \rfloor + N_p^{init}$;
$P \leftarrow P_{1:N_p}$;
$N_{Arch} \leftarrow \beta N_P$;
while $|A| > N_{Arch}$ **do**
  $a \leftarrow$ **RandomlySelected**$(1, |A|)$;
  $A = A \backslash A_a$;
end

Algorithm 4: MemoryUpdate$\left(S_{CR}, S_{sF}, t\right)$.

Result: $[\mu_{CR}, \mu_{sF}]$
$k \leftarrow \text{rem}(t-1, H) + 1$;
$n \leftarrow |S_{CR}|$;
for $i = 1 \preceq to \preceq n$ **do**
  $w_i \leftarrow \preceq \log(n + 0.5) - \log(i)$;
end
$\mu_{CR,k} \leftarrow \frac{\sum_{j=1}^{n} w_j S_{CR,j}^2}{\sum_{j=1}^{n} w_j S_{CR,j}}$;
$\mu_{sF,k} \leftarrow \frac{\sum_{j=1}^{n} w_j S_{sF,j}^2}{\sum_{j=1}^{n} w_j S_{sF,j}}$;

five, and seven out of 30 test problems with dimensions 10, 30, 50, and 100, respectively.

From Tables S.4 and S.5, we observe that LSHADE-Sin-CS outperforms LSHADE-Sin in nine, ten, nine, and nine out of 30 test problems with dimensions 10, 30, 50, and 100, respectively. On the other hand, LSHADE-Sin surpasses LSHADE-Sin-CS in two, six, five, and five out of 30 test problems with dimensions 10, 30, 50, and 100, respectively. Therefore, the above comparisons substantiate that CS improves the performance of DE algorithms on global optimization problems.

4.3. Effectiveness of the proposed orthogonal initialization scheme

In this experiment, we investigate the effectiveness of the proposed orthogonal-array-based initialization in DE's state-of-the-art algorithms. Again, two state-of-the-art algorithms, LSHADE-Sin and jSO, are considered in this investigation. Usually, initial solutions are generated randomly in these algorithms. We replace this procedure with the proposed initialization scheme and design LSHADE-Sin-OI and jSO-OI, respectively.

The results of this experiment are reported in Tables S.6-S.9. With respect to the overall comparative analysis, from Tables S.6 and S.7, we can see that jSO-OI performs better than jSO in majority of the test problems in all dimensions. More specifically, in six, eleven, ten, and twelve problems with dimensions 10, 30, 50, and 100, respectively. This can be attributed to the fact that jSO-OI generates the initial population more uniformly in the search-space contours. This facilitates the algorithm to initiate search in better clusters of the promising areas. Similar inferences can be drawn in the case of LSHADE-Sin-OI after analyzing Tables S.8 and S.9. Here, LSHADE-Sin-OI shows superior performance in ten or more problems in all dimensions. The comparative analysis suggests that the proposed initialization scheme enhances the performance of DE-based algorithms by generating better initial solutions.

Table 2
BRT results of mLSHADE vs. other state-of-the-art algorithms on CEC 2020 test-suite.

mLSHADE	+/=/-			
vs.	5D	10D	15D	20D
EBOwithCMAR	6/4/0	8/2/0	6/4/0	8/2/0
HSES	9/1/0	8/2/0	9/1/0	9/1/0
LSHADE-cnEpSin	4/5/1	9/1/0	8/1/1	7/2/1
LSHADE-SPACMA	4/5/1	9/1/0	7/3/0	9/1/0

Table 3
Ranking of mLSHADE and other state-of-the-art algorithms on CEC 2020 test-suite.

Algorithm	Score1	Score2	Total	Rank
mLSHADE	**50.00**	**50.00**	**100.00**	**1**
EBOwithCMAR	32.88	29.15	62.03	2
HSES	22.38	17.69	40.08	5
LSHADE-cnEpSin	31.91	26.69	58.60	3
LSHADE-SPACMA	31.42	24.05	55.47	4

4.4. Effectiveness of the proposed parameter adaptation technique

To examine the usefulness of the proposed parameter adaptation technique, we implement this technique in LSHADE to come up with a new algorithm called mLSHADE. We select 10 test problems with dimensions 5,10,15, and 20 from CEC2020 test-suite for this experiment as this parameter adaptation technique is specially designed for cheaper problems. For a comparative analysis, four state-of-the-art algorithms: EBOwithCMAR [79], HSES [80], LSHADE-cnEpSin [77], and LSHADE-SPACMA [81], are considered as contenders for mLSHADE. Note that these algorithms are among the top-performers in IEEE CEC 2018 competition.

We calculate the best, mean, and standard deviation of errors resulting from all algorithms in this experiment. We report all experimental results for test problems with dimensions 5, 10, 15, and 20 in Table S.10. Moreover, BRT and the ranking of all algorithms are done as suggested in IEEE CEC 2020 competition [6].

As observed in Table 2, mLSHADE performs better than EBOwithCMAR, HSES, LSHADE-cnEpSin, and LSHADE-SPACMA on six, nine, four, and four problems, respectively, in case of 5-dimension problems. LSHADE-cnEpSin and LSHADE-SPACMA generate better solutions than mLSHADE only in one problem each. In all the remaining dimensions, only LSHADE-cnEpSin could perform better than mLSHADE, that also in a couple of cases. The performance of mLSHADE is consistently better or at least equivalent to all other algorithms.

With reference to Table 3, we notice that mLSHADE is ranked first with a 100 score followed by EBOwithCMAR, LSHADE-cnEpSin, LSHADE-SPACMA, and HSES. Therefore, the above comparison reinstates that the proposed parameter adaptation technique is highly effective for the LSHADE algorithm.

4.5. Performance of the proposed OLSHADE-CS algorithm

In this experiment, we assess the performance of OLSHADE-CS on test problems of IEEE CEC 2020 competition. In this benchmark suite, 10 test problems with dimensions = 5, 10, 15, and 20 are proposed where these problems can be categorized into four groups: unimodal function (1), basic functions (2 − 4), hybrid functions (5 − 7), and composition functions (8 − 10). Other settings of this suite are set as suggested in [6].

4.5.1. Time-complexity

For calculating the time complexity of LSHADE-CS, we follow the steps suggested in CEC2020's technical report [6]. As per the report, we perform the following steps:

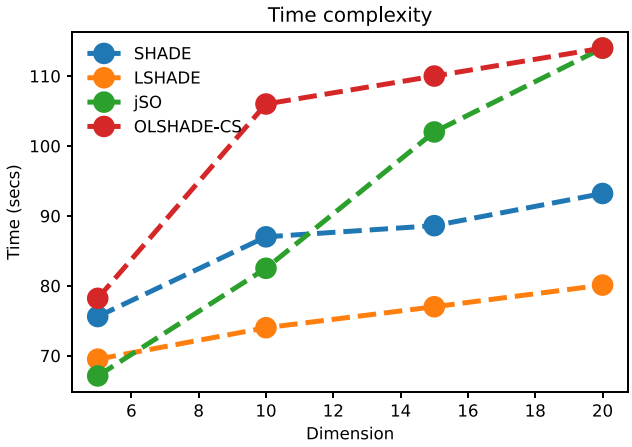

Fig. 2. The time-complexity of SHADE, LSHADE, jSO, and OLSHADE-CS on CEC2020 benchmark-suite.

a) Compute time required to run the following code, $T0$

$x = 0.55;$
$for\ i = 1 : 1000000$
$x = x + x;\ x = x/2;\ x = x * x;\ x = \text{sqrt}(x);$
$x = \log(x);\ x = \exp(x);\ x = x/(x + 2);$
$end.$

b) Compute time required to call problem function F1 over 200,000 evaluations for a given dimension, $T1$.
c) Compute time required to solve problem function F1 over 200,000 maximum evaluations for the same dimension, $T2$.
d) The time complexity of the algorithm can be reflected using $\frac{T2-T1}{T0}$.

Using the steps mentioned above, the time complexities of SHADE, LSHADE, jSO, and OLSHADE-CS are calculated on the CEC2020 benchmark suite. The time complexities of these algorithms over different dimensions (i.e., 5, 10, 15 and 20) are plotted in Fig. 2. As observed in the figure, the time complexity of the proposed algorithm OLSHADE-CS is little higher than the other algorithms due to the orthogonal-array-based initialization process. This process of initialization is computationally more expensive than the conventional initialization process in other DE-based algorithms.

4.5.2. Parameter setting

We use the following values for parameters of OLSHADE-CS in this experiment.

1. *Population Size:* $N_p^{init} = 6D^2$, $N_p^{\min} = 4$, $\alpha = 2.6$.
2. *Control Parameters:* $H = 20D$, $S_{CR}^0 = 0.2$ and $S_{sF}^0 = 0.6$.
3. *Function Evaluation Budget:* $Max_{nfes} = 5e4,\ 1e6,\ 3e6,$ and $1e7$ for problem dimensions = 5, 10, 15, and 20, respectively.
4. *Distribution of Budget:* In initialization: first 20%, optimization: last 80%, conservative selection: first half of optimization budget (40%), and conventional selection: last half of the optimization budget (40%)

4.5.3. Parameter analysis of OLSHADE-CS

This section demonstrates the sensitivity of the parameters of the proposed algorithm. Moreover, we also analyze their influence on the performance of the algorithm.

1) Settings of the Proposed Parameter Adaptation Technique: In the parameter adaptation technique, we have to set the following parameters before initializing the main optimization process: 1) Size of the memory (H), 2) initial value of S_{CR} (S_{CR}^0), and 3) initial value of S_{sF} (S_{sF}^0). These settings highly influence the performance

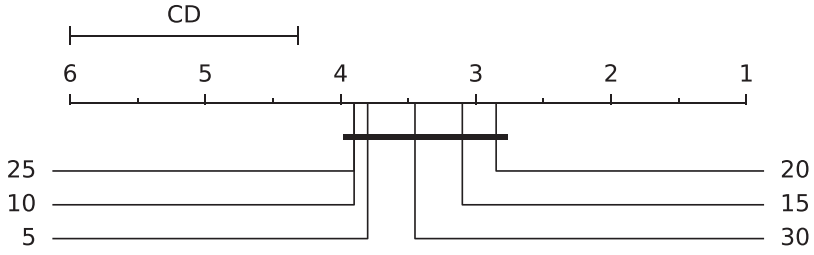

Fig. 3. CD plot for different values of H.

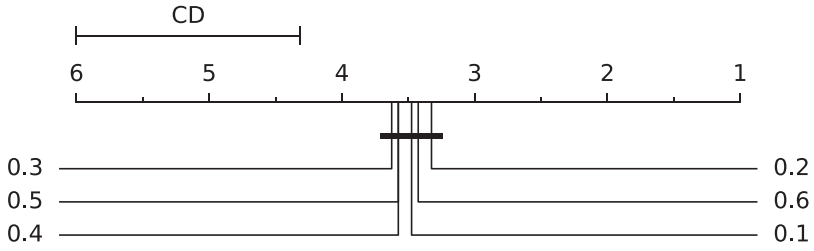

Fig. 4. CD plot for different values of S^0_{CR}.

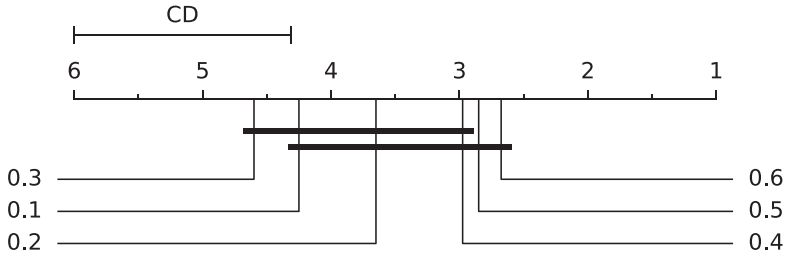

Fig. 5. CD plot for different values of S^0_{sF}.

of the algorithm, and we need to set them beforehand. However, these settings depend upon the characteristics of the problem-space. Therefore, we need to choose settings that provide satisfactory performance for all problems of the benchmark suite. To confirm the choice of the settings, detailed experiments are carried out on the CEC2020 benchmark suite. We choose the following discrete values for the parameters for trials.

a) $H = \{5, 10, 15, 20, 25, 30\} \times D$,
b) $S^0_{CR} = \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6\}$, and
b) $S^0_{sF} = \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6\}$.

For comparison purposes, Critical Difference (CD) plots are used. The CD is a tool to compare outcomes of many algorithms on multiple problems. In CD plots, the position of an algorithm indicates its Friedman rank across all problems, where low rank symbolizes that the algorithm performs better than the other algorithms having higher ranks. A thick line can connect two or more algorithms if their performance is similar in statistical significance. In our case, the CD plots for the above-mentioned parameters are shown in Figs. 3–5. By analyzing the plots, we can pick the following optimal values for the parameters: $H = 20D$, $S^0_{CR} = 0.2$, and $S^0_{sF} = 0.6$.

2) Settings of $best_1$ and $best_2$: Two parameters, $best_1$ and $best_2$, are used in mutation operators while creating two sets of the best solutions. A solution is randomly selected from these sets as a participant in the mutation operations. Here, $best_1$ and $best_2$ are the sizes of the sets, respectively, in terms of % of population size. Following settings are used for the sensitivity analysis:

a) $best_1 = \{0.1, 0.15, 0.2, 0.25, 0.3\}$, and
b) $best_2 = \{0.2, 0.3, 0.4, 0.5, 0.6\}$.

CD plots for these parameters are shown in Figs. 6 and 7. We can see that $best_1$ at 0.25 performs better than the other $best_1$ values. Similarly, $best_2$ at 0.5 value helps to achieve the best performance of the algorithm. Thus, we select these values as the default values for the proposed algorithm.

3) Distribution of Function Evaluation Budget: In OLSHADE-CS, the optimization process has three phases: orthogonal initialization, optimization with conservative selection, and optimization with greedy selection. These phases are executed sequentially and we require to assign a function evaluation budget for each phase. For analysis of the sensitivity of the budget, we select the following values for orthogonal initialization (B_{OI}) and optimization with greedy selection (B_{GS}).

a) $B_{OI} = \{0.0, 0.05, 0.10, 0.15, 0.20, 0.25, 0.30\}$, and
b) $B_{GS} = \{0.0, 0.2, 0.4, 0.6, 0.8, 1.0\}$.

We present the CD plots of these parameters in Figs. 8 and 9. As shown in these figures, B_{OI} with 0.2 value exhibits the most superior performance, while the same is applicable for B_{GS} with 0.6 value. Therefore, we choose these values for OLSHADE-CS.

4.5.4. An analysis of LSHADE-CS algorithm with different initialization schemes

In this section, the performance of LSHADE-CS is analyzed with different initialization techniques. We design the following algorithms by incorporating various initialization techniques in LSHADE-CS:

i) LSHADE-CS with opposition based initialization (OBLSHADE-CS) [82],
ii) LSHADE-CS with uniform initialization (ULSHADE-CS) [83], and
iii) LSHADE-CS with chaos initialization (CLSHADE-CS) [84].

The obtained results, including the BRT and BFT outcomes, of these algorithms are provided in Table S.11. From this table, we can observe that OLSHADE-CS performs better than other algorithms for majority of the problems in all dimensions. This shows that the proposed initialization scheme is highly effective and even better than many other initialization schemes.

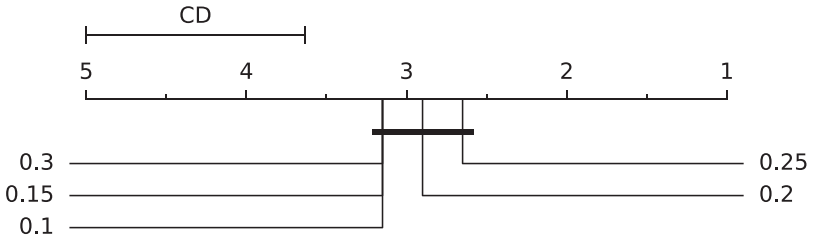

Fig. 6. CD plot for different values of $best_1$.

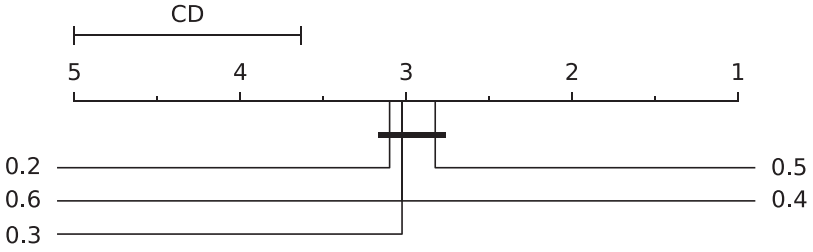

Fig. 7. CD plot for different values of $best_2$.

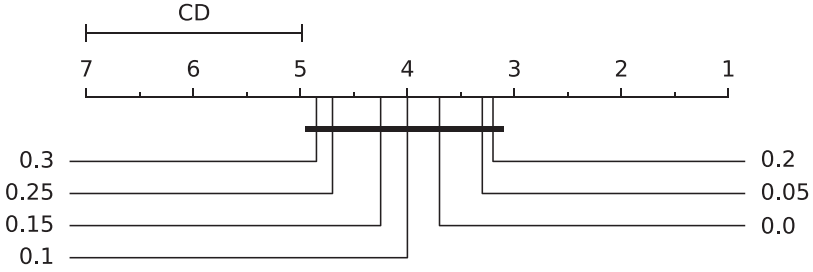

Fig. 8. CD plot for different values of B_{OI}.

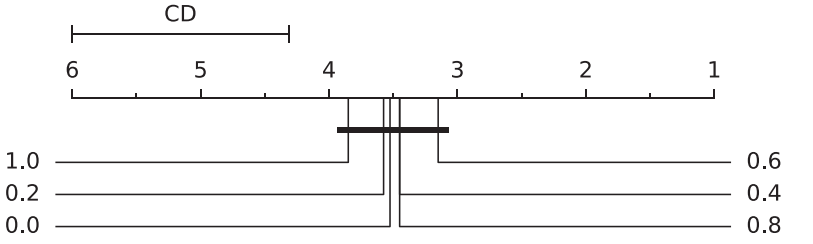

Fig. 9. CD plot for different values of B_{GS}.

Table 4
BRT results of LSHADE-CS algorithms of various initialization techniques on CEC 2020 test-suite.

OLSHADE-CS	+/=/-			
vs.	5D	10D	15D	20D
OBLSHADE-CS	6/4/0	8/1/1	7/1/2	8/1/1
ULSHADE-CS	4/5/1	8/1/1	6/2/2	7/2/1
CLSHADE-CS	6/4/0	8/1/1	7/2/1	8/2/0

Table 5
Ranking of LSHADE-CS algorithms of various initialization techniques on CEC 2020 test-suite.

Algorithm	Score1	Score2	Total	Rank
OLSHADE-CS	**50.00**	**50.00**	**100.00**	**1**
OBLSHADE-CS	23.64	28.19	51.83	2
ULSHADE-CS	20.75	30.00	50.75	3
CLSHADE-CS	20.92	24.60	45.52	4

BRT results are reproduced in Table 4. As shown in this table, OLSHADE-CS performs significantly better than others on most of the problems in all dimensions. Moreover, we rank all algorithms following the guidelines of IEEE CEC 2020. The result summary is provided in Table 5, which shows that OLSHADE-CS scores 100 and ranks first position in the point table. Therefore, we can infer that the orthogonal-based initialization scheme enhances the performance of DE in a more effective manner than many other initialization schemes.

4.5.5. Comparison of OLSHADE-CS with state-of-the-art DE algorithms

The comparative analysis of OLSHADE-CS and DE-based state-of-the-art algorithms is presented in this section. We consider the following algorithms: distance based parameter adaptation for SHADE (DISH) [85], tuning-based mutation in SHADE (Tb-jSO) [86], and DE with linear bias reduction in parameter adaptation (LSHADE-LBR) [87]. We tune all the parameters of the algorithms for IEEE CEC 2020 benchmark problems where the algorithm has not originally been applied to the problems in the source paper.

We report the statistical outcome of these algorithms in Table S.12. As shown in the table, the performance of OLSHADE-CS is superior to other algorithms over a majority of the problems. Apart from the remarkably better results in different dimensions, OLSHADE-CS ranks the best in BFT among all algorithms in all dimensions. The summary of BRT

Table 6
BRT results of OLSHADE-CS vs state-of-the-art DE algorithms on CEC 2020 test-suite.

OLSHADE-CS	+/=/-			
vs.	5D	10D	15D	20D
DISH	8/2/0	7/2/1	7/1/2	8/1/1
Tb-jSO	6/4/0	7/1/2	5/1/4	7/2/1
LSHADE-LBR	8/2/0	6/2/2	6/1/3	7/2/1

Table 7
Ranking of OLSHADE-CS and state-of-the-art DE algorithms on CEC 2020 test-suite.

Algorithm	Score1	Score2	Total	Rank
OLSHADE-CS	**50.00**	**50.00**	**100.00**	**1**
DISH	23.85	28.27	52.12	4
Tb-jSO	26.99	32.22	59.21	3
LSHADE-LBR	28.50	34.26	62.76	2

results is reported in Table 6. The following findings can be summarized from this table.

1. *Test problems with dimension = 5:* From Table 6, we can observe that the performance of OLSHADE-CS is either statistically similar or better than DISH, Tb-jSO, and LSHADE-LBR algorithms. Indeed, the algorithm is better than these state-of-the-art DE-based algorithms in eight, six, and eight test problems, respectively.
2. *Test problems with dimension = 10:* The performance of OLSHADE-CS is better than DISH, Tb-jSO, and LSHADE-LBR in seven, seven, and six test problems, respectively. On the other hand, its performance is inferior to these algorithms in one, two, and two test problems, respectively.
3. *Test problems with dimension = 15:* OLSHADE-CS outperforms in seven, five, and six test problems when compared with DISH, Tb-jSO, and LSHADE-LBR, respectively. However, OLSHADE-CS is inferior to others in two, four, and three test problems, respectively.
4. *Test problems with dimension = 20:* In this case, OLSHADE-CS performs better than DISH, Tb-jSO, and LSHADE-LBR in eight, seven, and seven test problems, respectively. In contrast, the performance of OLSHADE-CS is worse than each of these algorithms only in one problem.

Furthermore, we calculate the ranking scores as suggested in IEEE CEC 2020 competition and report in Table 7. As shown in this table, OLSHADE-CS scores a perfect 100, followed by LSHADE-LBR with a score of 62.76. Hence, we can conclude that the performance of OLSHADE-CS is highly competitive and it is one of the best among the DE-based algorithms.

4.5.6. Comparison with top-ranked algorithms of IEEE CEC 2020 competition. This section compares the performance of OLSHADE-CS with top-ranked algorithms in IEEE CEC 2020 competition: IMODE [88], AGSK [89], and j2020 [90]. The parameters of these algorithms are fixed at the same values as defined in their original articles. The results obtained from these algorithms are provided in Table 8. In this table, the best, mean and standard deviations of errors found by these algorithms over 30 different runs are recorded for all test problems. The BRT results and BFT ranking of these algorithms are also included in this table.

As shown in these tables, OLSHADE-CS exhibits considerably better performance than other state-of-the-art algorithms for most of the test problems. We can summarize the following findings from the tabulated results:

1. *Test problems with dimension = 5:* With reference to Table 8, OLSHADE-CS is better than IMODE, AGSK, and j2020 in two, five, and six test problems, respectively. On the other hand, OLSHADE-CS is inferior only in two problems to IMODE. Furthermore, OLSHADE-CS achieves the best value in Friedman's ranking test (i.e., BFT) of all algorithms.
2. *Test problems with dimension = 10:* In this case, OLSHADE-CS performs better than IMODE, AGSK, and j2020 in three, five, and four test problems, respectively. In contrast, these algorithms perform better than IMODE in two, three, and five test problems, respectively. However, OLSHADE-CS ranks the lowest among all in BFT.
3. *Test problems with dimension = 15:* As shown in Table 8, OLSHADE-CS is superior in five, four, and five test problems to IMODE, AGSK, and j2020, respectively. However, OLSHADE-CS is worse in one, two, and four test problems than the comparable algorithms, respectively. Consequently, OLSHADE-CS secures the highest rank among all algorithms according to BFT.
4. *Test problems with dimension = 20:* For this dimension, the performance of OLSHADE-CS is better than the performance of IMODE, AGSK, and j2020 in four, five, and four test problems, respectively. On the other hand, OLSHADE-CS performs worse than these algorithms in one, three, and two test problems, respectively. Besides, according to BFT, OLSHADE-CS is the best performer among all algorithms.

Furthermore, we illustrate the CD plots of all algorithms on the IEEE CEC 2020 benchmark-suite in Fig. 10. As shown in this figure, OLSHADE-CS performs better than other algorithms on 5D problems. For 10D problems, all algorithms perform better than OLSHADE-CS. However, in the case of 15D problems, OLSHADE-CS outperforms others. IMODE is the best algorithm among all in the case of 20D problems. To analyze the dynamics of all algorithms during the optimization process, several problems (F4, F5, F6, and F10) of IEEE CEC 2020 are selected, and convergence curves are depicted in Fig 11. From the convergence plots, we can conclude that in OLSHADE-CS, solutions are slowly improved due to the conservative selection process. Population diversity is initially preserved to explore the search space more thoroughly. This factor primarily attributes to the remarkable performance of OLSHADE-CS on majority of the IEEE CEC 2020 benchmark problems.

For the overall comparison of performance, we also calculate the performance score as suggested in the competition and report the same in Table 9. As noted from the table, OLSHADE-CS obtains a perfect 100 score, which is the best among all. The overall comparison and results substantiate that the proposed algorithm is highly efficient and superior to most of the other state-of-the-art optimization algorithms.

4.5.7. Contributions of different operators of OLSHADE-CS. In the previous experiments, we verified the effectiveness of the proposed algorithm as a whole. In this section, we evaluate the contributions of each of the proposed operators in enhancement of the performance of OLSHADE-CS. The following three variants of OLSHADE-CS are designed.

i) mLSHADE: LSHADE with the proposed parameter adaptation technique and ensemble of four mutation strategies.
ii) OLSHADE: LSHADE with the proposed orthogonal array based initialization.
iii) LSHADE-CS: LSHADE with the proposed conservative selection scheme.

The results of all algorithms together with OLSHADE-CS on IEEE CEC-2020 benchmark suite are reported in Table S.13. The ranking of all variants based on different operators of the proposed algorithm is reported in Table 10. From the table, we observe that the proposed initialization scheme influences the performance most, followed by the conservative selection scheme. The parameter adaptation technique in mutation and crossover strategies has the least influence.

4.6. Discussion

In this paper, we propose an optimization algorithm named as OLSHADE-CS. In the algorithm, we modify the initialization and selection steps of the DE to improve the search capabilities. Furthermore,

Table 8
Comparison of proposed algorithm (OLSHADE-CS) and other state-of-the art algorithms on test problems of CEC 2020 with 5, 10, 15, and 20 dimensions.

Dimension = 5															
Prob	OLSHADE-CS			IMODE				AGSK				j2020			
	Best	Mean	Std.	Best	Mean	Std.	BRT	Best	Mean	Std.	BRT	Best	Mean	Std.	BRT
1	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	=	**0.00E+00**	**0.00E+00**	**0.00E+00**	=	**0.00E+00**	**0.00E+00**	**0.00E+00**	=
2	0.00E+00	1.54E-01	8.48E-02	**0.00E+00**	**8.33E-02**	**8.89E-02**	-	6.14E-01	1.64E+01	2.58E+01	+	1.91E-04	3.23E+00	3.74E+00	+
3	**0.00E+00**	**1.97E+00**	**1.99E+00**	5.15E+00	5.15E+00	0.00E+00	+	4.38E-07	2.87E+00	2.05E+00	+	0.00E+00	3.42E+00	2.33E+00	+
4	0.00E+00	6.15E-03	1.21E-02	**0.00E+00**	**0.00E+00**	**0.00E+00**	-	1.67E-03	1.11E-01	6.05E-02	+	0.00E+00	7.68E-02	6.40E-02	+
5	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	=	**0.00E+00**	**0.00E+00**	**0.00E+00**	=	0.00E+00	1.37E-01	2.86E-01	+
6	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	=	**0.00E+00**	**0.00E+00**	**0.00E+00**	=	**0.00E+00**	**0.00E+00**	**0.00E+00**	=
7	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	=	**0.00E+00**	**0.00E+00**	**0.00E+00**	=	**0.00E+00**	**0.00E+00**	**0.00E+00**	=
8	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	=	**0.00E+00**	**0.00E+00**	**0.00E+00**	=	0.00E+00	6.28E-01	2.39E+00	+
9	0.00E+00	1.32E+00	5.02E+00	**0.00E+00**	**0.00E+00**	**0.00E+00**	=	0.00E+00	3.33E+01	4.79E+01	+	0.00E+00	2.05E+01	3.75E+01	+
10	0.00E+00	1.70E+02	1.38E+02	0.00E+00	2.44E+02	1.36E+02	+	0.00E+00	2.25E+02	1.32E+02	+	**0.00E+00**	**1.26E+02**	**9.03E+01**	=
		+/=/-			2/8/2				5/5/0				6/4/0		
BFT		**2.05**			2.25				2.85				2.85		
Dimension = 10															
Prob		OLSHADE-CS			IMODE				AGSK				j2020		
	Best	Mean	Std.	Best	Mean	Std.	BRT	Best	Mean	Std.	BRT	Best	Mean	Std.	BRT
1	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	=	**0.00E+00**	**0.00E+00**	**0.00E+00**	=	**0.00E+00**	**0.00E+00**	**0.00E+00**	=
2	5.00E-01	4.83E+00	2.69E+00	1.25E-01	4.20E+00	3.70E+00	=	4.09E+00	2.84E+01	3.21E+01	+	**0.00E+00**	**6.79E-01**	**1.16E+00**	-
3	4.93E+00	1.02E+01	2.26E+00	1.07E+01	1.21E+01	7.83E-01	+	6.12E-01	9.93E+00	4.26E+00	+	**0.00E+00**	**8.06E+00**	**3.88E+00**	-
4	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	=	1.94E-03	5.83E-02	3.11E-02	+	0.00E+00	1.09E-01	9.04E-02	+
5	0.00E+00	5.72E-01	5.20E-01	4.03E-06	3.88E-01	3.83E-01	=	0.00E+00	3.18E-01	3.06E-01	-	**0.00E+00**	**3.02E-01**	**3.13E-01**	-
6	**3.51E-02**	**8.75E-02**	**8.02E-02**	2.67E-02	9.15E-02	5.08E-02	+	2.20E-02	1.55E-01	1.17E-01	+	2.91E-02	4.78E-01	2.49E-01	+
7	**1.43E-07**	**1.64E-03**	**1.58E-03**	1.41E-05	8.54E-04	1.10E-03	=	0.00E+00	1.54E-03	1.71E-03	=	3.10E-07	6.73E-02	1.25E-01	+
8	0.00E+00	4.94E+01	3.73E+01	**0.00E+00**	**2.72E+00**	**7.46E+00**	-	0.00E+00	1.80E+01	2.38E+01	-	0.00E+00	1.54E+00	4.00E+00	-
9	1.00E+02	1.00E+02	0.00E+00	0.00E+00	4.10E+01	4.46E+01	-	0.00E+00	7.63E+01	4.29E+01	-	**0.00E+00**	**8.00E+01**	**4.07E+01**	-
10	**1.00E+02**	**1.00E+02**	**4.60E-04**	3.98E+02	3.98E+02	0.00E+00	+	1.00E+02	2.98E+02	1.43E+02	+	1.00E+02	1.40E+02	8.12E+01	+
		+/=/-			3/5/2				5/2/3				4/1/5		
BFT		2.6			2.5				2.55				**2.35**		

(*continued on next page*)

Table 8 (*continued*)

Dimension = 5															
Prob	OLSHADE-CS			IMODE				AGSK				j2020			
	Best	Mean	Std.	Best	Mean	Std.	BRT	Best	Mean	Std.	BRT	Best	Mean	Std.	BRT
Dimension = 15															
Prob	OLSHADE-CS			IMODE				AGSK				j2020			
	Best	Mean	Std.	Best	Mean	Std.	BRT	Best	Mean	Std.	BRT	Best	Mean	Std.	BRT
1	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	=	**0.00E+00**	**0.00E+00**	**0.00E+00**	=	**0.00E+00**	**0.00E+00**	**0.00E+00**	=
2	1.67E-01	2.80E+00	1.44E+00	1.25E-01	3.14E+00	3.22E+00	=	3.12E+00	1.85E+01	1.46E+01	+	**0.00E+00**	**5.72E-02**	**4.32E-02**	-
3	1.56E+01	1.57E+01	1.79E-01	1.56E+01	1.61E+01	3.12E-01	+	0.00E+00	1.42E+01	4.27E+00	=	**0.00E+00**	**6.78E+00**	**7.82E+00**	-
4	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	=	4.74E-02	1.42E-01	5.71E-02	+	0.00E+00	1.99E-01	7.47E-02	+
5	**9.95E-01**	**4.75E+00**	**2.53E+00**	1.15E+00	7.79E+00	3.66E+00	+	3.12E-01	6.25E+00	4.32E+00	+	0.00E+00	7.58E+00	7.69E+00	+
6	5.11E-02	4.45E-01	2.12E-01	2.81E-01	6.92E-01	2.52E-01	+	**1.72E-01**	**4.02E-01**	**2.23E-01**	=	1.65E-03	8.45E-01	2.09E+00	-
7	8.27E-03	4.99E-01	2.41E-01	1.28E-01	5.30E-01	2.23E-01	+	**1.09E-02**	**2.47E-01**	**2.00E-01**	-	6.81E-02	9.83E-01	2.03E+00	+
8	2.49E+01	8.74E+01	2.59E+01	**0.00E+00**	**4.18E+00**	**9.61E+00**	-	0.00E+00	6.85E+01	3.85E+01	-	0.00E+00	9.49E+00	2.74E+01	-
9	1.00E+02	1.00E+02	0.00E+00	**0.00E+00**	**9.33E+01**	**2.54E+01**	=	0.00E+00	9.67E+01	1.83E+01	=	1.00E+02	1.23E+02	5.68E+01	+
10	**1.00E+02**	**1.00E+02**	**1.16E-04**	4.00E+02	4.00E+02	0.00E+00	+	4.00E+02	4.00E+02	0.00E+00	+	1.00E+02	3.90E+02	5.48E+01	+
		+/=/-			5/4/1				4/4/2				5/1/4		
BFT		**2.2**			2.65				2.4				2.75		
Dimension = 20															
Prob	OLSHADE-CS			IMODE				AGSK				j2020			
	Best	Mean	Std.	Best	Mean	Std.	BRT	Best	Mean	Std.	BRT	Best	Mean	Std.	BRT
1	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	=	**0.00E+00**	**0.00E+00**	**0.00E+00**	=	**0.00E+00**	**0.00E+00**	**0.00E+00**	=
2	3.12E-02	1.86E-01	2.97E-01	3.12E-02	5.13E-01	7.12E-01	+	9.88E-02	9.68E-01	1.23E+00	+	**0.00E+00**	**2.60E-02**	**2.47E-02**	-
3	2.04E+01	2.05E+01	1.79E-01	2.04E+01	2.05E+01	1.25E-01	=	2.04E+01	2.04E+01	7.23E-15	-	**0.00E+00**	**1.44E+01**	**9.29E+00**	-
4	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	**0.00E+00**	=	6.83E-02	1.45E-01	5.47E-02	+	2.98E-02	1.80E-01	7.84E-02	+
5	**1.04E-01**	**1.50E+00**	**1.12E+00**	2.61E+00	1.09E+01	4.33E+00	+	2.20E+00	4.50E+01	3.67E+01	+	3.12E-01	7.78E+01	5.75E+01	+
6	**2.27E-02**	**6.77E-02**	**2.42E-02**	1.76E-01	3.02E-01	8.17E-02	+	8.54E-02	1.68E-01	4.45E-02	+	6.84E-02	1.91E-01	1.01E-01	+
7	4.51E-01	7.71E-01	1.71E-01	**2.38E-01**	**5.24E-01**	**1.64E-01**	-	1.83E-01	6.81E-01	9.09E-01	-	1.95E-02	1.98E+00	4.02E+00	=
8	4.25E+01	9.43E+01	1.75E+01	**3.05E+01**	**8.40E+01**	**1.89E+01**	=	7.46E+01	9.92E+01	4.63E+00	+	0.00E+00	9.27E+01	2.21E+01	=
9	1.00E+02	1.00E+02	0.00E+00	**1.34E-04**	**9.67E+01**	**1.83E+01**	=	1.00E+02	1.00E+02	0.00E+00	=	1.00E+02	3.39E+02	1.28E+02	+
10	3.99E+02	3.99E+02	2.00E+00	3.99E+02	4.00E+02	6.18E-01	+	**3.99E+02**	**3.99E+02**	**1.59E-02**	-	3.99E+02	3.99E+02	4.02E-02	=
		+/=/-			4/5/1				5/2/3				4/4/2		
BFT		**2.2**			2.6				2.45				2.75		

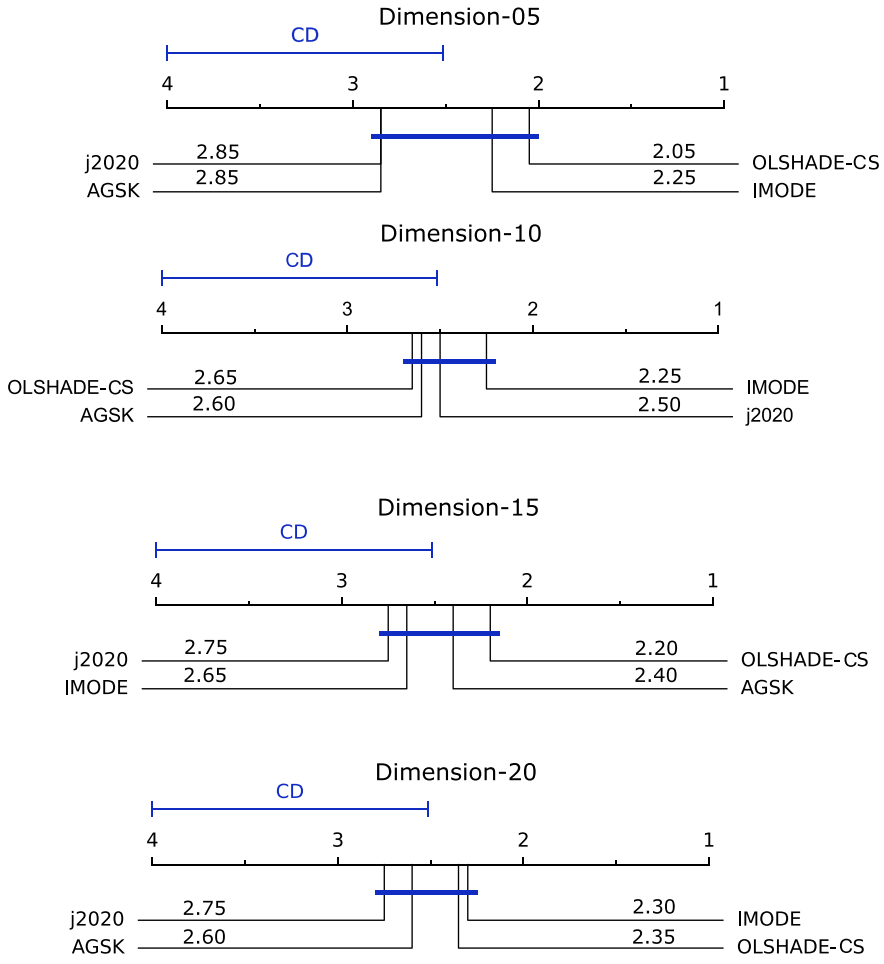

Fig. 10. CD plots for algorithms on IEEE CEC2020's suite.

Table 9
Ranking of OLSHADE-CS and other state-of-the-art algorithms on CEC 2020 test-suite.

Algorithm	Score1	Score2	Total	Rank
OLSHADE-CS	**50.00**	**50.00**	**100.00**	**1**
IMODE	38.96	44.24	83.20	2
AGSK	36.54	45.39	81.93	3
j2020	34.57	42.26	76.82	4

Table 10
Ranking of components of the proposed algorithm on CEC 2020 test-suite.

Algorithm	Score1	Score2	Total	Rank
OLSHADE-CS	**50.00**	49.37	**99.37**	**1**
mLSHADE	36.99	44.17	81.16	4
OLSHADE	46.07	45.02	91.09	2
LSHADE-CS	38.43	**50.00**	88.43	3

we incorporate a modified parameter adaptation technique and an ensemble of four mutation strategies to generate trial solutions in each iteration. In this way, the performance of the proposed algorithm is enhanced. We conduct extensive experiments on several widely used benchmark suites. From the experimental results and comparative analyses with state-of-the-art algorithms, we can summarize the following findings:

1) The performance of state-of-the-art DE algorithms improves if the conventional initialization method is replaced with the orthogonal array-based initialization.
2) State-of-the-art DE algorithms exhibit better performance with the incorporation of the proposed conservative selection scheme.
3) The proposed parameter adaptation technique and ensemble of four different mutation strategies improve the DE algorithm's search capability.
4) The proposed algorithm OLSHADE-CS shows significantly better performance than most of the state-of-the-art algorithms.
5) The orthogonal initialization and the conservative selection schemes influence the performance of OLSHADE-CS algorithm the most.

Further, we observe the following limitations for the proposed algorithm:

1) The proposed algorithm, OLLSHADE-CS, performs very well only for inexpensive optimization problems which have high budgets for function evaluations. Performance of OLSHADE-CS is paralyzed in cases of expensive problems, and inexpensive problems with low budget for function evaluations.
2) The convergence speed of the proposed algorithm is slow due to the conservative selection scheme.
3) In the case of 10D problems, OLSHADE-CS performs worse than the other top-ranked algorithms in IEEE CEC 2020 competition.

5. Conclusions

In this paper, we propose a variant of DE algorithm for global numerical optimization problems. The proposed algorithm incorporates

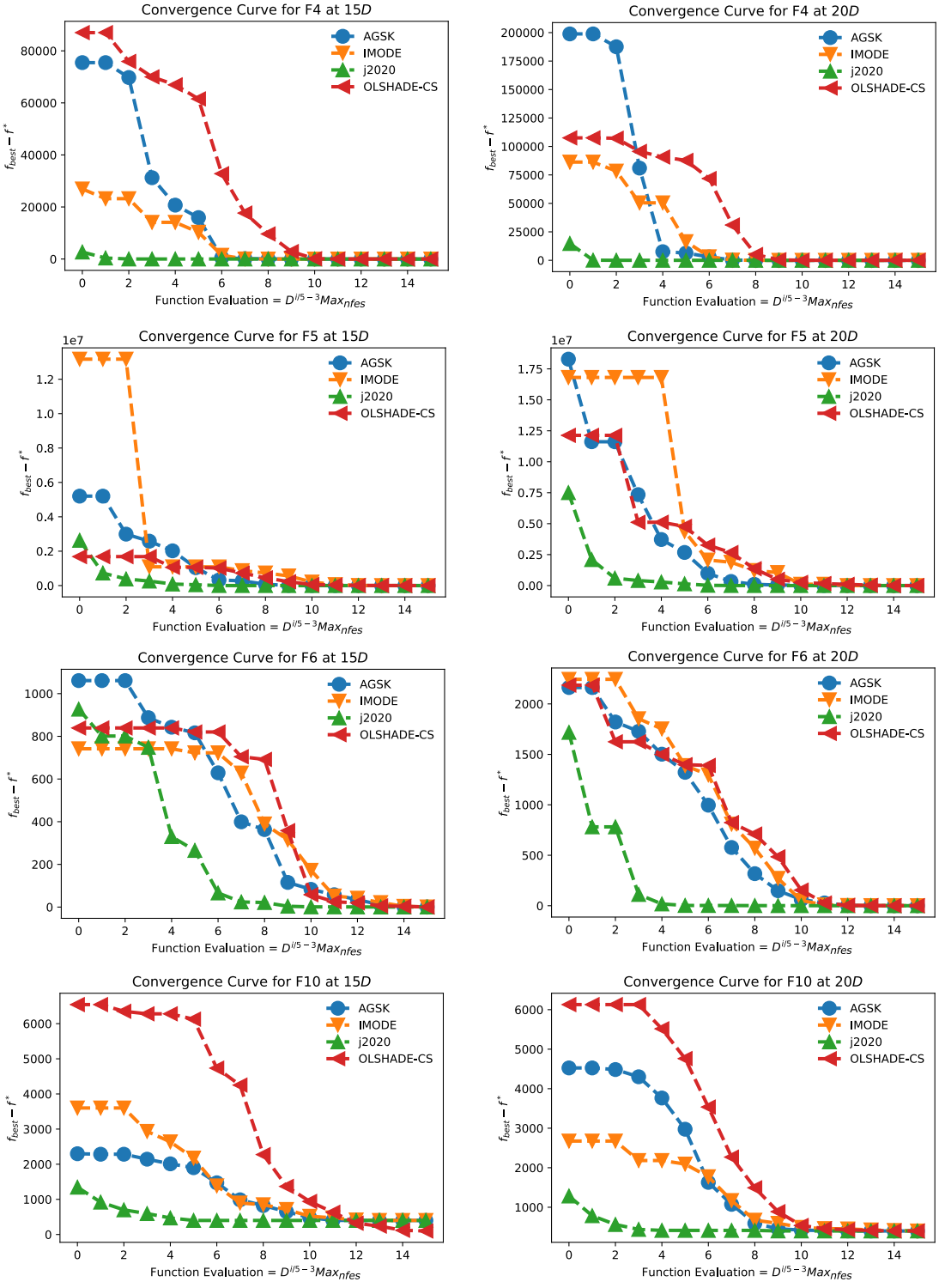

Fig. 11. The convergence curves for the median of errors derived from all algorithms.

orthogonal-array-based initialization, an ensemble of four mutation strategies, a parameter adaptation technique, and a conservative selection scheme to enhance the performance of DE. Orthogonal-array-based initialization distributes the initial population in the promising areas of the search space. As a result, the performance of DE is improved when compared with its counterpart adopting random uniform distribution of initial population. The ensemble of four mutation strategies improves the balance between explorative and exploitative search. The proposed parameter adaptation technique dynamically sets the scale parameters and crossover rates utilizing the successful historical values. The conservative selection scheme aims to select better trial vectors that evolve from the population and the associated individuals.

We conduct extensive experiments to analyze the efficacy of each of the proposed schemes and operations. The outcomes of these experiments prove that the proposed operators significantly improve the performance of the DE-based algorithms in global optimization problems. OLSHADE-CS, which combines all the proposed operations and schemes, performs remarkably better than many state-of-the-art DE and other

evolutionary algorithms on most benchmark problems. After detailed evaluations of the algorithm, we conclude that the proposed schemes and operators improve the searchability, accelerate the convergence, provide a balance between explorative and exploitative search, and evolve the population towards the global optimum efficiently. Application of the proposed schemes to the framework of other algorithms such as particle swarm optimization, genetic algorithm, etc., can be potential future works.

CRediT authorship contribution statement

Abhishek Kumar: Conceptualization, Funding acquisition, Writing – original draft, Writing – review & editing. **Partha P. Biswas:** Conceptualization, Writing – original draft, Writing – review & editing. **Ponnuthurai N. Suganthan:** Conceptualization.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

Abhishek Kumar: Conceptualization, Funding acquisition, Writing – original draft, Writing – review & editing. **Partha P. Biswas:** Conceptualization, Writing – original draft, Writing – review & editing. **Ponnuthurai N. Suganthan:** Conceptualization.

Acknowledgements

All persons who have made substantial contributions to the work reported in the manuscript (e.g., technical help, writing and editing assistance, general support), but who do not meet the criteria for authorship, are named in the Acknowledgements and have given us their written permission to be named. If we have not included an Acknowledgements, then that indicates that we have not received substantial contributions from non-authors.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at 10.1016/j.swevo.2021.101010

References

[1] A. Kumar, G. Wu, M.Z. Ali, R. Mallipeddi, P.N. Suganthan, S. Das, A test-suite of non–convex constrained optimization problems from the real-world and some baseline results, Swarm Evol Comput 56 (2020) 100693.
[2] N.H. Awad, M.Z. Ali, J.J. Liang, B.Y. Qu, P.N. Suganthan, Problem definitions and evaluation criteria for the CEC 2017 special session and competition on single objective real-parameter numerical optimization, Nanyang Technol. Univ., Singapore, Tech. Rep 201611 (2016).
[3] A. Kumar, R.K. Misra, D. Singh, S. Mishra, S. Das, The spherical search algorithm for bound-constrained global optimization problems, Appl Soft Comput 85 (2019) 105734.
[4] S.P. Adam, S.-A.N. Alexandropoulos, P.M. Pardalos, M.N. Vrahatis, No free lunch theorem: a review, Approximation and optimization (2019) 57–82.
[5] A. Kumar, S. Das, R.K. Misra, D. Singh, A v-constrained matrix adaptation evolution strategy with broyden-based mutation for constrained optimization, IEEE Trans Cybern (2021).
[6] C. Yue, K. Price, P. Suganthan, J. Liang, M. Ali, B. Qu, N. Awad, P. Biswas, Problem definitions and evaluation criteria for the CEC 2020 special session and competition on single objective bound constrained numerical optimization, Comput. Intell. Lab., Zhengzhou Univ., Zhengzhou, China, Tech. Rep 201911 (2019).
[7] A. Kumar, B.K. Jha, S. Das, R. Mallipeddi, Power flow analysis of islanded microgrids: a differential evolution approach, IEEE Access 9 (2021) 61721–61738.
[8] A. Kumar, G. Wu, M.Z. Ali, Q. Luo, R. Mallipeddi, P.N. Suganthan, S. Das, A benchmark-suite of real-world constrained multi-objective optimization problems and some baseline results, Swarm Evol Comput 67 (2021) 100961.
[9] A. Kumar, S. Das, R. Mallipeddi, A reference vector-based simplified covariance matrix adaptation evolution strategy for constrained global optimization, IEEE Trans Cybern (2020).
[10] A. Kumar, S. Das, I. Zelinka, A modified covariance matrix adaptation evolution strategy for real-world constrained optimization problems, in: Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion, 2020, pp. 11–12.
[11] S. Mishra, A. Kumar, D. Singh, R.K. Misra, Butterfly optimizer for placement and sizing of distributed generation for feeder phase balancing, in: Computational Intelligence: Theories, Applications and Future Directions-Volume II, Springer, 2019, pp. 519–530.
[12] T. Maini, A. Kumar, R.K. Misra, D. Singh, Fuzzy rough set-based feature selection with improved seed population in PSO and IDS, in: Computational Intelligence: Theories, Applications and Future Directions-Volume II, Springer, 2019, pp. 137–149.
[13] A. Kumar, S. Das, I. Zelinka, A self-adaptive spherical search algorithm for real–world constrained optimization problems, in: Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion, 2020, pp. 13–14.
[14] R.K. Misra, D. Singh, A. Kumar, Spherical search algorithm: A metaheuristic for bound-constrained optimization, in: Indo-French Seminar on Optimization, Variational Analysis and Applications, Springer, 2020, pp. 421–441.
[15] R. Storn, K. Price, Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces, J. Global Optim. 11 (4) (1997) 341–359.
[16] K. Price, R.M. Storn, J.A. Lampinen, Differential evolution: a practical approach to global optimization, Springer Science & Business Media, 2006.
[17] S. Das, P.N. Suganthan, Differential evolution: a survey of the state-of-the-art, IEEE Trans. Evol. Comput. 15 (1) (2010) 4–31.
[18] S. Das, S.S. Mullick, P.N. Suganthan, Recent advances in differential evolution–an updated survey, Swarm Evol Comput 27 (2016) 1–30.
[19] G. Wu, X. Shen, H. Li, H. Chen, A. Lin, P.N. Suganthan, Ensemble of differential evolution variants, Inf Sci (Ny) 423 (2018) 172–186.
[20] R.D. Al-Dabbagh, F. Neri, N. Idris, M.S. Baba, Algorithmic design issues in adaptive differential evolution schemes: review and taxonomy, Swarm Evol Comput 43 (2018) 284–311.
[21] K.R. Opara, J. Arabas, Differential evolution: a survey of theoretical analyses, Swarm Evol Comput 44 (2019) 546–558.
[22] K. Opara, J. Arabas, Comparison of mutation strategies in differential evolution–a probabilistic perspective, Swarm Evol Comput 39 (2018) 53–69.
[23] A.P. Piotrowski, J.J. Napiorkowski, Step-by-step improvement of JADE and SHADE-based algorithms: success or failure? Swarm Evol Comput 43 (2018) 88–108.
[24] X. Chen, W. Du, F. Qian, Solving chemical dynamic optimization problems with ranking-based differential evolution algorithms, Chin. J. Chem. Eng. 24 (11) (2016) 1600–1608.
[25] I. Boussaïd, A. Chatterjee, P. Siarry, M. Ahmed-Nacer, Hybridizing biogeography-based optimization with differential evolution for optimal power allocation in wireless sensor networks, IEEE Trans. Veh. Technol. 60 (5) (2011) 2347–2353.
[26] P.P. Biswas, P.N. Suganthan, R. Mallipeddi, G.A.J. Amaratunga, Optimal power flow solutions using differential evolution algorithm integrated with effective constraint handling techniques, Eng Appl Artif Intell 68 (2018) 81–100.
[27] P.P. Biswas, P. Arora, R. Mallipeddi, P.N. Suganthan, B.K. Panigrahi, Optimal placement and sizing of FACTS devices for optimal power flow in a wind power integrated electrical network, Neural Computing and Applications (2020) 1–22.
[28] T.W. Liao, Two hybrid differential evolution algorithms for engineering design optimization, Appl Soft Comput 10 (4) (2010) 1188–1199.
[29] E. Cuevas, D. Zaldivar, M. Pérez-Cisneros, A novel multi-threshold segmentation approach based on differential evolution optimization, Expert Syst Appl 37 (7) (2010) 5265–5271.
[30] G. Wu, W. Peng, X. Hu, R. Wang, H. Chen, Configuring differential evolution adaptively via path search in a directed acyclic graph for data clustering, Swarm Evol Comput 55 (2020) 100690.
[31] D.C. Montgomery, Design and analysis of experiments 3rd edn new york wiley (1991).
[32] C.R. Hicks, Fundamental concepts in the design of experiments(1964).
[33] Q. Wu, On the optimality of orthogonal experimental design, Acta Mathematicae Applacatae Sinica 1 (4) (1978) 283–299.
[34] A. Kumar, R.K. Misra, D. Singh, S. Das, Testing a multi-operator based differential evolution algorithm on the 100-digit challenge for single objective numerical optimization, in: 2019 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2019, pp. 34–40.
[35] G. Wu, X. Wen, L. Wang, W. Pedrycz, P.N. Suganthan, A voting-mechanism based ensemble framework for constraint handling techniques, IEEE Trans. Evol. Comput. (2021).
[36] Q. Fan, X. Yan, Y. Zhang, Auto-selection mechanism of differential evolution algorithm variants and its application, Eur J Oper Res 270 (2) (2018) 636–653.
[37] K.M. Sallam, S.M. Elsayed, R.A. Sarker, D.L. Essam, Landscape-based adaptive operator selection mechanism for differential evolution, Inf Sci (Ny) 418 (2017) 383–404.
[38] S. Elsayed, R. Sarker, C.A.C. Coello, Fuzzy rule-based design of evolutionary algorithm for optimization, IEEE Trans Cybern 49 (1) (2017) 301–314.
[39] M.F. Tasgetiren, P.N. Suganthan, Q.-K. Pan, An ensemble of discrete differential evolution algorithms for solving the generalized traveling salesman problem, Appl Math Comput 215 (9) (2010) 3356–3368.
[40] S. Elsayed, R. Sarker, C.C. Coello, T. Ray, Adaptation of operators and continuous control parameters in differential evolution for constrained optimization, Soft comput 22 (19) (2018) 6595–6616.
[41] S.X. Zhang, L.M. Zheng, K.S. Tang, S.Y. Zheng, W.S. Chan, Multi-layer competitive–cooperative framework for performance enhancement of differential evolution, Inf Sci (Ny) 482 (2019) 86–104.
[42] M.A. Attia, M. Arafa, E.A. Sallam, M.M. Fahmy, An enhanced differential evolution algorithm with multi-mutation strategies and self-adapting control parameters, International Journal of Intelligent Systems and Applications 11 (4) (2019) 26.

[43] X. Yu, X. Yu, Y. Lu, G.G. Yen, M. Cai, Differential evolution mutation operators for constrained multi-objective optimization, Appl Soft Comput 67 (2018) 452–466.
[44] A.W. Mohamed, A novel differential evolution algorithm for solving constrained engineering optimization problems, J Intell Manuf 29 (3) (2018) 659–692.
[45] G. Wu, R. Mallipeddi, P.N. Suganthan, R. Wang, H. Chen, Differential evolution with multi-population based ensemble of mutation strategies, Inf Sci (Ny) 329 (2016) 329–345.
[46] G. Wu, R. Mallipeddi, P.N. Suganthan, Ensemble strategies for population-based optimization algorithms–a survey, Swarm Evol Comput 44 (2019) 695–711.
[47] B. Kazimipour, X. Li, A.K. Qin, A review of population initialization techniques for evolutionary algorithms, in: 2014 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2014, pp. 2585–2592.
[48] H. Maaranen, K. Miettinen, M.M. Mäkelä, Quasi-random initial population for genetic algorithms, Computers & Mathematics with Applications 47 (12) (2004) 1885–1895.
[49] Z. Ma, G.A.E. Vandenbosch, Impact of random number generators on the performance of particle swarm optimization in antenna design, in: 2012 6th European Conference on Antennas and Propagation (EUCAP), IEEE, 2012, pp. 925–929.
[50] M. Pant, T. Radha, V.P. Singh, Particle swarm optimization: Experimenting the distributions of random numbers, in: IICAI, 2007, pp. 412–420.
[51] H.G. Schuster, W. Just, Deterministic chaos: an introduction, John Wiley & Sons, 2006.
[52] N. Dong, C.-H. Wu, W.-H. Ip, Z.-Q. Chen, C.-Y. Chan, K.-L. Yung, An opposition-based chaotic GA/PSO hybrid algorithm and its application in circle detection, Computers & Mathematics with Applications 64 (6) (2012) 1886–1902.
[53] Y. Gao, Y.-J. Wang, A memetic differential evolutionary algorithm for high dimensional functions' optimization, in: Third International Conference on Natural Computation (ICNC 2007), volume 4, IEEE, 2007, pp. 188–192.
[54] M. Zhang, W. Zhang, Y. Sun, Chaotic co-evolutionary algorithm based on differential evolution and particle swarm optimization, in: 2009 IEEE International Conference on Automation and Logistics, IEEE, 2009, pp. 885–889.
[55] W.-f. Gao, S.-y. Liu, A modified artificial bee colony algorithm, Computers & Operations Research 39 (3) (2012) 687–697.
[56] W.-f. Gao, S.-y. Liu, L.-l. Huang, Particle swarm optimization with chaotic opposition-based population initialization and stochastic search technique, Commun. Nonlinear Sci. Numer. Simul. 17 (11) (2012) 4316–4327.
[57] A.L. Gutiérrez, M. Lanza, I. Barriuso, L. Valle, M. Domingo, J.R. Perez, J. Basterrechea, Comparison of different pso initialization techniques for high dimensional search space problems: A test with fss and antenna arrays, in: Proceedings of the 5th European Conference on Antennas and Propagation (EUCAP), IEEE, 2011, pp. 965–969.
[58] B. Liu, L. Wang, Y.-H. Jin, F. Tang, D.-X. Huang, Improved particle swarm optimization combined with chaos, Chaos, Solitons & Fractals 25 (5) (2005) 1261–1271.
[59] R. Senkerik, M. Pluhacek, Z.K. Oplatkova, D. Davendra, I. Zelinka, Investigation on the differential evolution driven by selected six chaotic systems in the task of reactor geometry optimization, in: 2013 IEEE Congress on Evolutionary Computation, IEEE, 2013, pp. 3087–3094.
[60] S. Rahnamayan, G.G. Wang, Solving large scale optimization problems by opposition-based differential evolution (ODE), WSEAS Transactions on Computers 7 (10) (2008) 1792–1804.
[61] C.-H. Chou, J.-N. Chen, Genetic algorithms: initialization schemes and genes extraction, in: Ninth IEEE International Conference on Fuzzy Systems. FUZZ-IEEE 2000 (Cat. No. 00CH37063), volume 2, IEEE, 2000, pp. 965–968.
[62] K.-T. Fang, W.-C. Shiu, J.-X. Pan, Uniform designs based on latin squares, Stat Sin (1999) 905–912.
[63] W. Gong, Z. Cai, L. Jiang, Enhancing the performance of differential evolution using orthogonal design method, Appl Math Comput 206 (1) (2008) 56–69.
[64] Y. Wang, Z. Cai, Q. Zhang, Enhancing the search ability of differential evolution through orthogonal crossover, Inf Sci (Ny) 185 (1) (2012) 153–177.
[65] Z.-H. Zhan, J. Zhang, Y. Li, Y.-H. Shi, Orthogonal learning particle swarm optimization, IEEE Trans. Evol. Comput. 15 (6) (2010) 832–847.
[66] S.-Y. Ho, H.-S. Lin, W.-H. Liauh, S.-J. Ho, Opso: orthogonal particle swarm optimization and its application to task assignment problems, IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans 38 (2) (2008) 288–298.
[67] Z.-Y. Jiang, Z.-X. Cai, Y. Wang, Hybrid self-adaptive orthogonal genetic algorithm for solving global optimization problems, Journal of Software 21 (6) (2010) 1296–1307.
[68] X. Hu, J. Zhang, J. Zhong, An enhanced genetic algorithm with orthogonal design, in: 2006 IEEE International Conference on Evolutionary Computation, IEEE, 2006, pp. 3174–3181.
[69] J. Sacks, W.J. Welch, T.J. Mitchell, H.P. Wynn, Design and analysis of computer experiments, Statistical science 4 (4) (1989) 409–423.
[70] S. Rahnamayan, H.R. Tizhoosh, M.M.A. Salama, Opposition-based differential evolution, IEEE Trans. Evol. Comput. 12 (1) (2008) 64–79.
[71] Y.-W. Leung, Y. Wang, An orthogonal genetic algorithm with quantization for global numerical optimization, IEEE Trans. Evol. Comput. 5 (1) (2001) 41–53.
[72] P.P. Biswas, P.N. Suganthan, Large initial population and neighborhood search incorporated in LSHADE to solve CEC2020 benchmark problems, in: 2020 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2020, pp. 1–7.
[73] R. Tanabe, A. Fukunaga, Success-history based parameter adaptation for differential evolution, in: 2013 IEEE Congress on Evolutionary Computation, IEEE, 2013, pp. 71–78.
[74] J. Zhang, A.C. Sanderson, Jade: adaptive differential evolution with optional external archive, IEEE Trans. Evol. Comput. 13 (5) (2009) 945–958.
[75] R. Tanabe, A.S. Fukunaga, Improving the search performance of SHADE using linear population size reduction, in: 2014 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2014, pp. 1658–1665.
[76] J. Carrasco, S. García, M.M. Rueda, S. Das, F. Herrera, Recent trends in the use of statistical tests for comparing swarm and evolutionary computing algorithms: practical guidelines and a critical review, Swarm Evol Comput 54 (2020) 100665.
[77] N.H. Awad, M.Z. Ali, P.N. Suganthan, Ensemble sinusoidal differential covariance matrix adaptation with euclidean neighborhood for solving CEC2017 benchmark problems, in: 2017 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2017, pp. 372–379.
[78] J. Brest, M.S. Maučec, B. Bošković, Single objective real-parameter optimization: Algorithm jSO, in: 2017 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2017, pp. 1311–1318.
[79] A. Kumar, R.K. Misra, D. Singh, Improving the local search capability of effective butterfly optimizer using covariance matrix adapted retreat phase, in: 2017 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2017, pp. 1835–1842.
[80] G. Zhang, Y. Shi, Hybrid sampling evolution strategy for solving single objective bound constrained problems, in: 2018 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2018, pp. 1–7.
[81] A.W. Mohamed, A.A. Hadi, A.M. Fattouh, K.M. Jambi, LSHADE with semi-parameter adaptation hybrid with CMA-ES for solving CEC 2017 benchmark problems, in: 2017 IEEE Congress on Evolutionary computation (CEC), IEEE, 2017, pp. 145–152.
[82] S. Rahnamayan, H.R. Tizhoosh, M.M.A. Salama, A novel population initialization method for accelerating evolutionary algorithms, Computers & Mathematics with Applications 53 (10) (2007) 1605–1614.
[83] X.-j. Xu, X.-p. Huang, D.L. Qian, Adaptive accelerating differential evolution, Complex systems and complexity science 5 (3) (2008) 87–92.
[84] L. Peng, Y. Wang, G. Dai, Z. Cao, A novel differential evolution with uniform design for continuous global optimization, J. Comput. 7 (1) (2012) 3–10.
[85] A. Viktorin, R. Senkerik, M. Pluhacek, T. Kadavy, A. Zamuda, Distance based parameter adaptation for success-history based differential evolution, Swarm Evol Comput 50 (2019) 100462.
[86] X. Sun, L. Jiang, Y. Shen, H. Kang, Q. Chen, Success history-based adaptive differential evolution using turning-based mutation, Mathematics 8 (9) (2020) 1565.
[87] V. Stanovov, S. Akhmedova, E. Semenkin, Differential evolution with linear bias reduction in parameter adaptation, Algorithms 13 (11) (2020) 283.
[88] K.M. Sallam, S.M. Elsayed, R.K. Chakrabortty, M.J. Ryan, Improved multi-operator differential evolution algorithm for solving unconstrained problems, in: 2020 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2020, pp. 1–8.
[89] A.W. Mohamed, A.A. Hadi, A.K. Mohamed, N.H. Awad, Evaluating the performance of adaptive gaining sharing knowledge based algorithm on CEC 2020 benchmark problems, in: 2020 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2020, pp. 1–8.
[90] J. Brest, M.S. Maučec, B. Bošković, Differential evolution algorithm for single objective bound-constrained optimization: Algorithm j2020, in: 2020 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2020, pp. 1–8.