Information Sciences 372 (2016) 470–491

Contents lists available at ScienceDirect

Information Sciences

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

# A decremental stochastic fractal differential evolution for global numerical optimization

Noor H. Awad[a], Mostafa Z. Ali[b,c,*], Ponnuthurai N. Suganthan[a], Edward Jaser[b]

[a] Nanyang Technological University, School of Electrical & Electronic Engineering, 639798, Singapore
[b] Jordan University of Science & Technology, School of Computer Information Systems, 22110, Jordan
[c] Princess Sumayya University for Technology, King Hussein Faculty of Computing Sciences, Jordan

ARTICLE INFO

*Article history:*
Received 14 March 2016
Revised 5 August 2016
Accepted 11 August 2016
Available online 17 August 2016

*Keywords:*
Evolutionary algorithms
Fractals
Differential evolution
Exploration
Exploitation

ABSTRACT

The aim of hybridization in the context of evolutionary computation is to combine appropriate operators from different evolutionary computation paradigms to form a single technique that enjoys a statistically superior performance over a wide range of optimization problems. This paper introduces a novel hybridization between differential evolution and update processes of the stochastic fractal search algorithm. The diffusion property of the fractal search algorithm is applied in random fractals followed by two novel update processes to explore the search space more efficiently. In this algorithm, a diffusion process based on differential evolution algorithm is used instead of random fractals in the original stochastic fractal search algorithm. A new success-based scheme is used to utilize the update processes and to solve the burden of extra computations during the search. This new algorithm captures the strengths of both component algorithms and produces a greater explorative power as compared to the original algorithms. To verify the performance of our algorithm, a challenging test suite of 30 benchmark functions from the IEEE CEC2014 real parameter single objective competition is used. The results affirm the effectiveness and robustness of the proposed approach compared to the original stochastic fractal search and other recent state-of-the-art algorithms. The proposed algorithm enjoys a statistically superior performance over most of the tested benchmarks, especially hybrid and composition test functions compared to the other contestant algorithms.



## 1. Introduction

Engineers and scientists have to solve real-life problems formulated as global optimization problems. In these problems, the objective is to find a set of decision variables representing the globally optimal solution of the objective function. Mathematically, an optimization problem is expressed as either minimization or maximization of an objective function $f(\vec{X})$ $(f : \Omega \subseteq \Re^D \rightarrow \Re)$ where $\Omega$ is the domain of the problem with $D$ decision variables, represented as a vector $\vec{X} = (x_1, x_2, x_3, ..., x_D)^T$. As a result, the task of an optimization algorithm is to search for a vector $\vec{X}^*$ that represents an optimal solution for the objective function $f(\vec{X})$ where $f(\vec{X}^*) < f(\vec{X})$, $\forall \vec{X} \in \Omega$.

Due to the presence of optimization problems in many fields, there has been a growing interest in developing efficient optimization algorithms, in particular, population-based evolutionary algorithms which are capable of generating satisfactory

* Corresponding author at: Jordan University of Science & Technology, School of Computer Information Systems, 22110, Jordan.
*E-mail addresses:* noor0029@ntu.edu.sg (N.H. Awad), mzali.pn@gmail.com (M.Z. Ali), epnsugan@ntu.edu.sg (P.N. Suganthan), ejaser@psut.edu.jo (E. Jaser).

http://dx.doi.org/10.1016/j.ins.2016.08.032
0020-0255/

solutions in a reasonable time. Many research works have been done to demonstrate the effectiveness of using nature inspired methods [7,12,13,16,27,29,35,39,46]. Despite the fact that these algorithms have been shown as powerful techniques for solving real-valued optimization problems, they can suffer from stagnation and premature convergence easily when solving complex landscape problems. Researchers attempted to tackle this problem by developing novel algorithms and hybridizations that combine the strengths of different EAs or local search methods. These hybrid algorithms merge the search properties and capabilities to offer a new algorithm with new features that is superior in terms of robustness and resiliency to their constituent algorithms [1,3,5,7,8,14,22,33].

One of the well-known hybrid algorithms is AMALGAM-SO [42]. AMALGAM-SO hybridizes PSO, GA and Covariance Matrix Adaptation to develop a new powerful algorithm. A self-adaptive learning scheme is used to determine the participation ratio of each algorithm offspring in each generation. This algorithm showed promising results on solving high dimensional complex multimodal problems. On the other hand, the Covariance Matrix Adaptation is also used in Differential Evolution algorithm for crossover operation as in CoBiDE algorithm [45]. This concept helps the Differential Evolution algorithm to rely on a coordinate system to a certain extent along with increasing the robustness of the DE algorithm to solve complex optimization problems and a wide range of engineering optimization problems. The UMOEAs is another example that hybridizes three multi-operator evolutionary algorithms which are: Genetic Algorithm, Differential Evolution and Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [20]. The algorithm consists of three sub-populations and each sub-population is evolved using one of the aforementioned algorithms. The success rate of each of the used algorithms is recorded and the best algorithm is used to evolve its own sub-population while other sub-populations are kept on hold.

In [23], the authors introduced PSO6-Mtsls algorithm which merged PSO algorithm with multiple trajectory search in which each particle used local information from six neighbors to generate a new individual guided by trajectory search. Besides, Cultural Algorithm was also used as a search engine to develop new hybrid algorithms [7,22]. One recent example utilized the niching method, Tabu search and two selection methods to develop a novel Cultural Algorithm framework to solve continuous engineering optimization problems [3].

Although the above examples show the advantages of hybridization, it also comes with a potential cost. Firstly when more than one algorithm is used, the search engine requires more computational resources to organize the work between different algorithms. Secondly, an intermediate coordinate system is needed when more component algorithms are included to balance between exploration, exploitation and the participation ratio of such individual algorithms at each generation. It is beneficial thus to develop a simple hybrid algorithm which is compared to constituting component algorithms and other state-of-the-art hybrid algorithms.

The differential evolution algorithm is a powerful tool that successfully handled many optimization problems and applications [2,6,15,28,34,38,50,51]. It is a population-based technique with main operations which are mutation, crossover and selection. Although its effectiveness and competitive performance has been demonstrated, many experimental studies and theoretical analysis [53,21] show that it is sensitive and still quite dependent on the settings of control parameters such as scaling factor ($F$), crossover rate ($CR$), mutation/crossover strategy and population size ($NP$). The mechanisms proposed for control parameter calculation might have control parameters as well. Also, the rate of changing the control parameters might influence performance [49]. As a result, many studies proposed different adaptive and self-adaptive techniques to control parameter settings of DE such as: SaDE [36], jDE [9], JADE [52], SHADE [40], L-SHADE [41] and $m$DE-$b$ES [4]. SaDE for example used a learning period of 50 generations to adapt the probability of generating a new offspring using one of two mutation strategies, "DE/rand/1″ and "DE/current-to-best/1″. The scaling factor and crossover rate are generated using the Gaussian distribution [36]. Different from SaDE, jDE algorithm uses the uniform distribution to generate $F$ and $CR$ values and the new offspring with the best parameter values are most likely to survive and to propagate to the next generations [9]. On the other hand, the EPSDE algorithm uses an ensemble of mutation strategies with a pool of control parameter settings to generate a new offspring [32]. The CoDE algorithm also used a similar idea of using different mutation strategies and parameter settings [43].

Some other DE variants introduced new mutation strategies. In JADE [52], the authors proposed a novel mutation strategy called current-to-pbest/1 with an optional external archive to solve varying benchmark problems. In this mutation strategy, the pbest is chosen from the top best individuals in each generation. Adaptive schemes based on Cauchy and normal distributions with means $\mu_F$ and $\mu_{CR}$ and standard deviation of 0.1 are used to generate $F_i$ and $CR_i$. The successful values of $F_i$ and $CR_i$ are stored to update the mean values of $\mu_F$ and $\mu_{CR}$ for subsequent generations. The JADE algorithm is further developed as SHADE [40] and L-SHADE [41] algorithms. In SHADE [40], the greediness factor in current-to-pbest/1 is adapted and a new history-based scheme is used to generates $F_i$ and $CR_i$ values by selecting a random index $r_i$ from [1, $H$] to choose the mean values $\mu_F$ and $\mu_{CR}$ from memory $M$ of $H$ historical successful entries. The L-SHADE algorithm adds a linear population size reduction scheme to reduce the population size at each generation [41] inspired from [10,30]. Recently, L-SHADE is incorporated with a successful-parent selection scheme and an eigenvector-based crossover. This algorithm namely, SPS-L-SHADE-EIG, is tested on CEC 2015 benchmark set and showed competitive performance [26]. The dynNP-DE algorithm is also used a dynamic population reduction in classic rand/1/bin [10]. In this algorithm, $F_i$ and $CR_i$ are adapted to $F_l$+$rand$.$F_u$ and $rand$, respectively, if a random number is less than $t_1$ and $t_2$, and kept unchanged otherwise. $rand$ is a uniform random number that is generated within the interval [0, 1] and $t_1$, $t_2$, $F_l$ are set to 0.1 while $F_u$ is set to 0.9. The orthogonal crossover is used in OXDE algorithm to enhance the search ability of Differential Evolution [45]. In their work, they used $L_M(Q^k)$ orthogonal array to construct $M$ solutions as a crossover of DE/rand/1. $M$, $Q$, $k$ are set to 9, 3 and 4 respectively. Two novel operators were introduced in [12]. These two operators include a neighbor guided scheme for selecting involved

parents in mutation and the direction induced mutation strategy. In this algorithm namely NDi-DE, a direction information with neighborhood strategy is incorporated in DE to accelerate convergence [11]. Other works used Ranking-Based Mutation Operators and Gaussian Bare-Bones in Differential Evolution as found in [25] and [47], respectively.

On the other hand, stochastic fractal search (SFS) is a metaheuristic algorithm developed recently that imitate the natural phenomenon of growth [37]. It guides the search process by finding near optimal solutions. The aim of this algorithm was to develop a new optimization algorithm that overcomes the shortcomings of nature-inspired techniques such as premature convergence. However, the accuracy of SFS is not guaranteed when tackling complex optimization problems where stagnation during the search and premature convergence become apparent obstacles degrading its performance. This algorithm uses two phases based on fractal properties to satisfy the fast convergence and accuracy within the search process.

In this paper, we introduce a simple, yet powerful, hybrid evolutionary algorithm that attempts to overcome the aforementioned shortcomings and merge the capabilities and strengths of differential evolution and stochastic fractal search (SFS) [37]. SFS will be used to modify the update process in the evolution of the search in DE to affect the quality of newly generated solutions. The coherency between the differential evolution's exploitative and explorative powers and stochastic fractal search's local search powers will help to develop a new successful optimizer to guide the search towards more successful solutions The new proposed hybrid algorithm, which we call decremental differential stochastic fractal evolutionary algorithm (***d*DSF-EA**), presents a three-stage optimization algorithm: Differential Evolution Diffusion process, Success-based update process and Population size dynamic reduction. In the first stage, the L-SHADE engine is used as the Diffusion process of the Stochastic Fractal search instead of random fractals. The use of random fractals with fixed parameters values will slow down convergence. To tackle this problem, L-SHADE is used in the diffusion process with adaptive parameters. L-SHADE adapts parameters based on offspring that survive in the population. In the second stage, an adaptive success-based mechanism to call the update processes is also introduced. In the original SFS algorithm, the two update processes use fixed number of function evaluations at each generation. As a result, function evaluations may be lost if the update process is unable to generate good solutions. The use of success-based scheme of the update processes is proposed to solve this issue. The proposed success-based scheme will assign a suitable number of function evolutions to each update process based on its success in previous generations. Hence, function evaluations will be utilized in an effective manner. Population size reduction has been shown to be highly effective in improving evolutionary algorithm's performance [10,30]. As a result, a linear population size reduction is used in the last stage after the update process is performed to improve the whole performance of our amalgam. Using those three stages, the proposed algorithm has a guided synergy to efficiently select appropriate solution directions by balancing exploration and exploitation during the search. Challenging IEEE CEC2014 benchmark set consisting of 30 problems with a diverse range of complexity is used to test the performance of the proposed algorithm. The proposed work is characterized by how the diffusion process is handled in this amalgam via L-SHADE, and dynamic participation function that controls the assignment of function evaluations to the update process, in addition to dynamical control of the population size. Comparative studies show that the proposed algorithm can have a superior performance compared to its constituent algorithms, SFS and L-SHADE, in addition to many other well-known state-of-the-art algorithms.

The rest of the paper is organized as follows. Section 2 introduces background of DE and stochastic fractal search. In Section 3, The new algorithm, ***d*DSF-EA**, is elaborated with a detailed discussion on how to inject differential evolution in the diffusion process and the adaptive success-based mechanism to call the update processes. Simulation results are presented in Section 4 for the comparison of ***d*DSF-EA** with other evolutionary algorithms. Finally, concluding remarks are summarized in Section 5.

## 2. Preliminaries

### 2.1. Canonical differential evolution algorithm

To better understand the proposed algorithm in Section 3, an overview of classical differential evolution is presented in this section. The Differential Evolution algorithm is a population-based technique that consists of a population of $NP$ individuals where each individual is expressed as a vector of $D$-dimensional decision variables as follows:

$$X_{i,g} = (x^1_{i,g}, ..., x^D_{i,g}),\ i = 1, ..., NP \tag{1}$$

where $G$ is the generation number, $D$ is the dimension of the problem and $NP$ is the population size. The algorithm starts initially with randomly distributed individuals within the search space of the problem being solved as follows:

$$x^j_{i,g_0} = x^j_{\min} + rand(0,1).(x^j_{\max} - x^j_{\min}) \quad j = 1, 2, ..., D \tag{2}$$

where $j$ is the index of parameter value in the $i$th individual vector at generation $g = 0$, $rand(0,1)$ is a uniformly distributed random generator in the range [0,1] and $X_{\min} = (x^1_{\min}, ..., x^D_{\min})$, $X_{\max} = (x^1_{\max}, ..., x^D_{\max})$ are the lower and upper bounds of each decision variable $x^j_i$.

After the initialization step, a new offspring is generated using one of the following five mutation DE strategies as presented in Eqs. (3)–( 7) where $V_{i,g} = (v^1_{i,g}, v^2_{i,g}, ..., v^D_{i,g})$ is the mutant vector which is generated against each individual $X_{i,g}$ in the population space, $F$ is the scaling factor that usually takes values within the range [0,1] and $r^i_1, r^i_2, r^i_3, r^i_4, r^i_5$ are different

randomly selected population members [38].

$$\text{DE/rand/1}: \quad V_{i,g} = X_{r_1,g} + F \cdot (X_{r_2,g} - X_{r_3,g}) \tag{3}$$

$$\text{DE/best/1}: \quad V_{i,g} = X_{best,g} + F \cdot (X_{r_1,g} - X_{r_2,g}) \tag{4}$$

$$\text{DE/current} - \text{best/1}: \quad V_{i,g} = X_{i,g} + F \cdot (X_{best,g} - X_{i,g} + X_{r_1,g} - X_{r_2,g}) \tag{5}$$

$$\text{DE/best/2}: \quad V_{i,g} = X_{best,g} + F \cdot (X_{r_1,g} - X_{r_2,g} + X_{r_3,g} - X_{r_4,g}) \tag{6}$$

$$\text{DE/rand/2}: \quad V_i = X_{r_1,g} + F \cdot (X_{r_2,g} - X_{r_3,g} + X_{r_4,g} - X_{5,g}) \tag{7}$$

The crossover phase is applied after the mutation operation is completed to generate a new trial vector $U_{i,g} = (u^1_{i,g}, u^2_{i,g}, ..., u^D_{i,g})$ for each mutant vector $V_{i,G}$ as shown below using binomial crossover:

$$u^j_{i,g} = \begin{cases} v^j_{i,g}, \ if\ (with\ probability\ of\ CR)\ or\ (j = j_{rand}) \\ x^j_{i,g}, \ otherwise \end{cases} \Bigg\} j = 1, 2, ..., D \tag{8}$$

where $CR$ is the crossover rate which is a user-defined value within the range [0,1] to control the percentage of parameter values of mutant vectors that should be copied to form a new child. $j_{rand}$ is a random index of a position in the mutant vector within the range [1, $D$].

### 2.2. Stochastic fractal search

The Stochastic Fractal Search (SFS) is an improved version of Fractal Search in which it is introduced to overcome some of the shortcomings of its original version [37]. The basic Fractal Search has many parameters that need to be tuned and designed well to reflect good results. Besides, there is no information exchange between particles. This affects the convergence rate and makes the search of particles to be performed independently. Furthermore, the basic Fractal Search is a dynamic algorithm in terms of the modifications on the number of agents and this results in a trade-off between accuracy and time consumption.

The SFS consists of two main processes: the Diffusion process and the Update process. The diffusion process simulates the basic Fractal Search by satisfying the intensification or exploitation property in which each particle is diffused around its current position. This property is essential because it increases the chance of locating global optimum and reduces the chance of getting trapped in a local optimum. This Diffusion process is different from the basic one in Fractal Search as it is considered a static process and does not cause a dramatic increase in the number of participating points to generate a new particle. In other words, the best generated particle from the Diffusion process is considered and all other ones will be discarded. On the other hand, the update process is considered as a diversification or an exploration phase in which each particle updates its position based on the positions of other selected points in the group.

Unlike Diffusion process in the basic Fractal Search in which it uses only the Levy Flight, the Diffusion process in Stochastic Fractal Search uses Gaussian Walks also. The goal of using Gaussian Walks is it is better than Levy flight in locating global optima as the Levy Flight converges faster. The pseudo-code of Stochastic Fractal Search is given in Fig. 1. Initially, the procedure starts by initializing a population space $P$ which consists of a set of points uniformly distributed within the search space similar to Eq. (2) as shown in step 1 of Fig. 2. After that, one of the two Gaussian Walks is used to generate new points as shown in Eqs (9) and ( 10) as shown in lines 4–12. The number of generated points is determined using a predefined parameter denoted as $\gamma$ which represents the maximum diffusion number.

$$GW_1 = Gaussian(\mu_{BP}, \sigma) + (rand(0,1) \times X_{best} - rand(0,1) \times X_i) \tag{9}$$

$$GW_2 = Gaussian(\mu_P, \sigma) \tag{10}$$

where *Gaussian* is a vector function of two vector arguments which are $\mu$ and $\sigma$, *rand(0,1)* is a uniform random number in the range [0, 1], $X_{best}$ is the position of best point while $X_i$ is the position of the $i$th point in the group. The two means $\mu_{BP}$ and $\mu_P$ are equal to $X_{best}$ and $X_i$ respectively while the vector $\sigma$ represents the standard deviation which is computed as shown in Eq. (11) using the absolute value of difference between $X_{best}$ and $X_i$ multiplied by $\frac{\log(g)}{g}$ where $g$ is the generation number.

$$\sigma = \frac{\log(g)}{g} \times \left| (x^1_{i,g}, ..., x^D_{i,g}) - (x^1_{best,g}, ..., x^D_{best,g}) \right| \tag{11}$$

Next, the exploration phase starts by calling the two update processes. The first one is applied for each vector index in every point while the other one is applied for all the points after they are ranked according to Eq. (12):

$$Pa_i = \frac{rank(X_i)}{NP} \tag{12}$$

**Algorithm: Stochastic Fractal Search Algorithm (SFS)**

**Input:** Maximum diffusion number: $\gamma$

Lower and upper bounds of problem being solved $\left[X_{\min}, X_{\max}\right]$

**Output:** Optimal solution of an optimization problem

1. Initialize population at first generation $g_0$, $P_{g_0} = \left\langle X_{1,g_0}, \ldots, X_{NP,g_0} \right\rangle$ where $X_{i,g_0} = (x^1_{i,g_0}, x^2_{i,g_0}, \ldots, x^D_{i,g_0})$, $i$=1,2,…,$NP$, are uniformly distributed within $X_{\min} = (x^1_{\min}, \ldots, x^D_{\min})$ and $X_{\max} = (x^1_{\max}, \ldots, x^D_{\max})$.
2. **While termination criterion is not met**
3. **Call** Diffusion process to generate new points:
4. **For** $i$ = 1 to $NP$
5. **For** $j$ = 1 to $\gamma$
6. **If** user selects first Gaussian Walks **GW1**
7. **Use** Eq. 9 to generate new point
8. **If** user selects second Gaussian Walks **GW2**
9. **Use** Eq. 10 to generate new point
10. **EndIf**
11. **EndFor**
12. **EndFor**
13. **Call** update process 1
14. **For** $i$ = 1 to $NP$
15. **If** $rand(0,1) < Pa_i$
16. **Use** Eq. 13 for each index $j$ in a point $X_i$ to generate new point $\overline{X}_i$
17. **Evaluate** $\overline{X}_i$ by calling objective function
18. **If** $f(\overline{X}_i) < f(X_i)$
19. $\overline{X}_i$ **replaces** $X_i$
20. **EndIf**
21. **EndIf**
22. **EndFor**
23. **Call** update process 2
24. **For** $i$ = 1 to $NP$
25. **If** $rand < Pa_i$
26. **Use** Eq. 14 for each point $X_i$ to generate new point $\overline{\overline{X}}_i$
27. **Evaluate** $\overline{\overline{X}}_i$ by calling objective function
28. **If** $f(\overline{\overline{X}}_i) < f(X_i)$
29. $\overline{\overline{X}}_i$ **replaces** $X_i$
30. **EndIf**
31. **EndIf**
32. **EndFor**
33. **EndWhile**
34. **Terminate and return output**

**Fig. 1.** Pseudo-code of Stochastic Fractal Search.

where $rank(X_i)$ is the rank for each point based on its fitness and $NP$ is the number of points in the population space $P$.

If a random number, $rand(0, 1)$, is less than $Pa_i$, the following equation is used by the first update process to update the $j$th index for each point, otherwise it remains unchanged.

$$\bar{X}_i^j = X_{r1}^j - \varepsilon \times (X_{r2}^j - X_i^j) \tag{13}$$

where $\bar{X}_i$ is the new point, $X_{r1}$ and $X_{r2}$ are two different random points selected from the population space $P$. After that, the new point $\bar{X}_i$ is evaluated. If it is better or equal, $\bar{X}_i$ replaces $X_i$.

**Algorithm: Diffusion Process via Differential Evolution (DvDE)**

**Input:** Entire population at generation $g$ : $P_g = \left\langle X_{1,g}, X_{2,g}, \ldots, X_{NP,g} \right\rangle$
Maximum number of diffusion: $\gamma$
Used Differential Evolution parameters
**Output:** New enhanced population based on differential diffusion process

1. **For** $i$=1 to $NP$ **Do**
2. **For** j = 1 to $\gamma$
3. **Create** new individual based on **Differential Evolution**
4. Generate a random index $h = rand(1, M_{size})$
5. Generate $F = randc(\mu_{F_h}, 0.1)$, $CR = randn(\mu_{CR_h}, 0.1)$
6. The greediness factor $p$ is adapted: ($p = 0.1*NP$)
7. Apply Eq. 15 to generate trial vector $U_{i,g}$ with binomial crossover
8. **If** $f(U_{i,g}) \le f(X_{i,g})$
9. $X_{i,g+1} = U_{i,g}$
10. **If** $f(U_{i,g}) \ne f(X_{i,g})$
11. $F_i \rightarrow S_F, CR_i \rightarrow S_{CR}$
12. **Else**
13. $X_{i,g+1} = X_{i,g}$, Add $X_{i,g}$ to archive $A$
14. **End If**
15. **End If**
16. If size of archive $A$ exceeds $NP$ then remove a random individual to add $X_{i,g}$
17. **End If**
18. **End For**
19. **End For**
20. **Terminate and return output**

**Fig. 2.** Pseudo-code of Diffusion Process via Differential Evolution (DvDE).

The second update process uses two different equations to update the position of the current point. If *rand* is less than $Pa_i$, the following equations are used to update the current point $X_i$. Otherwise no update process takes place. The first equation is applied if a random number $rand(0,1)$ is less than or equal to 0.5 using the position of best point $X_{best}$ and a random point $X_{r1}$. If $rand(0,1)$ is greater than 0.5, the current position of $X_i$ is updated using two random points denoted as $X_{r1}$ and $X_{r2}$. Similar to first update process, if the objective value of $\bar{\bar{X}}_i$ is better than that of $X_i$, a replacement occurs.

$$\bar{\bar{X}}_i = \begin{cases} X_i - rand(0,1) \times (X_{r1} - X_{best}), & \text{if} \varepsilon' \le 0.5 \\ X_i + rand(0,1) \times (X_{r1} - X_{r2}), & \text{otherwise} \end{cases} \tag{14}$$

## 3. Stochastic fractal differential evolution

The efficacy of solving optimization problems using Evolutionary Algorithms relies on the choice of strategies that are used to generate new individuals and their associated parameters. When the problems have different characteristics such as non-separability, ill-conditioning, multimodality, in addition to the presence of the curse of dimensionality, these characteristics require an adaptive scheme to fine tune the parameters. Although the Stochastic Fractal Search is claimed to rectify the problem of having many parameters that need to be well designed in basic Fractal Search, it still has parameters that use static values all over the generations. Using static values will not be a suitable choice for all problems. Static values for the parameters will degrade the performance. Moreover, there is no knowledge propagation at different stages of the. The quality of new individuals can be enhanced when using an adaptive scheme to fine tune the parameters by benefiting from the best values of generating well-performing individuals. Furthermore, the update process in Stochastic Fractal Search is considered as exploration while the diffusion process is considered as exploitation. However, there is no balance between these two phases in terms of used function evaluations (FEs). The number of function evaluations that are needed by the diffusion process equals to $\gamma \times NP$ where $\gamma$ is the maximum diffusion number while update process for both procedures takes2 $\times NP$. This means that in each generation, the SFS algorithm needs at least 3 $\times NP$function evaluations to generate $NP$

new individuals if $\gamma$ is set to 1. As a result, if the population cannot maintain a balance between exploration and exploitation search phases, the SFS algorithm may suffer from stagnation and premature convergence scenarios. Motivated by these observations, the proposed algorithm in this work utilizes Differential Evolution in Stochastic Fractal Search and proposes a new success-based scheme for the update process. Furthermore, a population size reduction is used after the update process is performed. In other words, the three main phases in the proposed algorithm are:

- Differential Evolution Diffusion process
- Success-based update process
- Population size dynamic reduction

The first phase in the proposed algorithm is the Differential Evolution Diffusion process in which Differential Evolution is used instead of Gaussian Walks in the original Stochastic Fractal Search algorithm. This eliminates the need to fine-tune the parameters used by the basic Diffusion process and uses an adaptive approach for controlling the DE's parameters. In our proposed algorithm, the Fractals are generated by modifying the evolution process via Differential Evolution rules instead of stochastic rules such as Gaussian Walks in the original Stochastic Fractal Search. This will enhance the search process and guide the evolutionary search toward promising regions instead of using random Gaussian Fractals. The second phase is the use of an update process utilizing a success-based scheme in which the number of individuals that an update process uses to generate new individuals is based on the mean success rate in the previous generations. This provides a dynamic strategy for calculating the participation ratio that each update process can use and also saves computations instead of consuming $2 \times NP$for each generation. Population size reduction is introduced as one of the highly effective ways for improving an evolutionary algorithm [10,30]. Therefore, the last phase reduces the population size in each generation in a dynamic manner. The following sub-sections explain the three main phases in details.

### 3.1. Diffusion of information via differential evolution (DvDE)

The Diffusion process is carried out via Differential Evolution, named here as (DvDE), instead of Gaussian Walks Diffusion in SFS. Each particle diffuses around its current position to satisfy intensification (exploitation). The aim of this process is to increases the diversity of the population and allows information exchange by propagating the best values of parameters to the next generation to generate good offspring. The basic engine of L-SHADE algorithm is used in our diffusion process as given in Fig. 2. L-SHADE algorithm is an improved version of SHADE [40] and JADE algorithms [52]. In this algorithm, current-to-pbest is used as a mutation strategy as shown in Eq. (15) where the best individual is chosen randomly from the top $NP \times rand(0,1)$ best individuals where *rand(0,1)* represents the greediness factor of mutation operation.

$$v_i(t) = x_i(t) + F_i(t).(x_{pbest}(t) - x_i(t)) + F_i(t).(x_{r_1}(t) - x_{r2}(t)) \tag{15}$$

An adaptive history-based scheme is used to adapt the scaling factor $F_i(t)$ and the crossover rate $CR_i(t)$ in each generation. A memory $M$ is used of $M_{size}$ entries for both $\mu_F$ and $\mu_{CR}$ which are responsible to control DE parameters: $F$ and $CR$. In the beginning, the contents of $\mu_F$ and $\mu_{CR}$ are initialized to 0.5. At each generation, a random index $h$ is generated to select the mean values of $\mu_{F.h}$ and $\mu_{CR.h}$ that will be used to generate $F_i(t)$ and $CR_i(t)$ using Cauchy and Normal distributions *randn* and *randc*, respectively with a variance equal to 0.1. After generating the trial vectors by applying the binomial crossover, the selection operation starts to replace the parent with the new offspring if the fitness value of the offspring is not worse than its parent. Moreover, if the offspring survives, the successful values of $F_i(t)$ and $CR_i(t)$ are stored in $S_F$ and $S_{CR}$ respectively. An external archive $A$ is used to store the inferior individuals that will be used to select $r_2$ index in Eq. (15). Initially the archive is filled with initial population, $P_{g_0}$. At each generation $g$, if the size of $A$ exceeds the maximum size, $NP$, random individuals are eliminated to make free space for newly added individuals. At the end of each generation, $\mu_F$ and $\mu_{CR}$ will be regenerated using the following equations where $S_F$, $S_{CR}$ are two vectors which store the successful parameters for $F$ and $CR$ respectively, $w_k$ is a weighted value which is computed using Eq. (20). $U_{k,g}$ is the trial vector obtained after mutation and before applying the binomial crossover and $X_{k,g}$ is $k$th individual (parent) in generation $g$.

$$\mu F = mean_{WL}(S_F) \tag{16}$$

$$\mu CR = mean_{WL}(S_{CR}) \tag{17}$$

where $mean_{WL}$ is the Lehmer mean and it is computed as shown in the following equations:

$$mean_{WL}(S) = \frac{\sum_{i=1}^{|S|} w_k.S_k^2}{\sum_{i=1}^{|S|} w_k.S_k} \tag{18}$$

$$w_i = \frac{\Delta f_i}{\sum_{l=1}^{|S|} \Delta f_l} \tag{19}$$

$$\Delta f_i = |f(U_{i.g}) - f(X_{i.g})| \tag{20}$$

**Algorithm: Success-based Update Process 1 (S-UP1)**

**Input:** Entire population at generation $g$: $P_g = \langle X_{1,g}, X_{2,g}, \ldots, X_{NP,g} \rangle$

**Output:** New enhanced population based on modified update process 1

1. Initialize success rate of update process 1 at generation $g$, $SR_{1,g} = 0$
2. Rank all the individuals using Eq. 12 and generate $Pa_i$
3. Generate two different random integer numbers $r_1, r_2 \in [1, N]$
4. Use Eqs. 21-22 to calculate participation ratio $N$
5. **For** $i$=1 to $N$ **Do**
6. **For** $j$=1 to $D$ **Do**
20. **If** $rand(0,1) < Pa_i$
7. Update $j$ parameter $X^j_{i,g}$ using Eq. 13 which used $r_1, r_2$ to generate $\overline{X}^j_{i,g}$
8. **If** $\overline{X}^j_{i,g} < X^j_{\min}$ or $\overline{X}^j_{i,g} > X^j_{\max}$
9. regenerate $\overline{X}^j_{i,g}$ using Eq. 23
10. **End If**
11. **End If**
12. **End For**
13. Calculate function value for new individual $\overline{X}_{i,g}$
14. **If** $f(\overline{X}_{i,g}) < f(X_{i,g})$
15. **Replace** $X_{i,g}$ with $\overline{X}_{i,g}$
16. Increase success counter, $SR_{1,g} = SR_{1,g} + 1$
17. **End If**
18. **End For**
19. **Terminate and return output**

**Fig. 3.** Pseudo-code of the modified Update Process 1 (Success-based).

### 3.2. Success-based update processes

The update process simulates how each particle updates its position based on the position of other particles in the group. However, one of the shortcomings that Stochastic Fractal Search suffers from is that the update phase consumes double the computations that the Diffusion process needs if maximum diffusion, $\gamma$ is set to 1. On the other hand, the Diffusion process consumes at least double the computations of update process if $\gamma \geq 2$. In other words, there is no balance between exploration (update process) and exploitation (Diffusion process). To tackle this issue, a success-based scheme for the update process (S-UP) is introduced in this work. According to this scheme, the update process is considered as an adaptive process in which the participation ratio (i.e. the number of individuals assigned to each update process) is dynamically updated in each generation. To determine this participation ratio $\phi$, this tournament is decided via the mean success rate of the previous generation which is computed as follows:

$$\phi = \frac{\sum_{t=0}^{G} SR_g}{G} \tag{21}$$

$$N = ceil(\phi) \tag{22}$$

where $SR_g$ is the success counter of an update process at generation $g$, and $N$ is the number of individuals that will be assigned to an update process. The generated point of an update process is called successful if the function value of the new point $Y_{i,g}$ is better than its parent $X_{i,g}$, in other words $f(Y_{i,g}) < f(X_{i,g})$. If this happens, the success counter of an update process is increased, $SR_g = SR_g + 1$.

Using this success-based adaptive process, the two modified update processes of Stochastic Fractal search are used and their pseudo-codes are presented in Figs. 3 and 4. Differing from original Stochastic Fractal Search, for each dimension $j$ of the newly generated individual, if the new vector element $Y^i_{j,g}$ is outside of the search range $[X^j_{\min}, X^j_{\max}]$, the following reset

**Algorithm: Success-based Update Process 2 (S-UP2)**

**Input:** Entire population at generation $g$ : $P_g = \langle X_{1,g}, X_{2,g}, \ldots, X_{NP,g} \rangle$

**Output:** New enhanced population based on modified update process 2

1. Initialize success rate of update process 2 at generation $g$, $SR_{2,g} = 0$
2. Rank all the individuals using Eq. 12 and generate $Pa_i$
3. Select the best individual, $X_{best,g}$
4. Use Eqs. 21-22 to calculate participation ratio $N$
5. **For** $i$=1 to $N$ **Do**
6. **If** *rand(0,1)* < $Pa_i$
7. Generate two different random integer numbers $r_1, r_2 \in [1, N]$
8. Update individual $X_i$ $(t)$ based on Eq. 14 to generate $\overline{\overline{X}}_{i,g}$
9. Regenerate each parameter if $\overline{\overline{X}}^j_{i,g} < X^j_{\min}$ or $\overline{\overline{X}}^j_{i,g} > X^j_{\max}$
10. **Else**
11. Do nothing
12. **EndIf**
13. Calculate function value for new individual $\overline{\overline{X}}_{i,g}$
14. **If** $f(\overline{\overline{X}}_{i,g}) < f(X_{i,g})$
15. **Replace** $X_{i,g}$ with $\overline{\overline{X}}_{i,g}$
16. Increase success counter, $SR_{2,g} = SR_{2,g} + 1$
17. **EndIf**
18. **End For**
19. **Terminate and return output**

**Fig. 4.** Pseudo-code of the modified Update Process 2 (Success-based).

strategy is used to regenerate a new element:

$$Y^j_{i,g} = \begin{cases} \dfrac{X^j_{\min} + X^j_{i,g}}{2}, & if\ Y^j_{i,g} < X^j_{\min} \\ \dfrac{X^j_{\max} + X^j_{i,g}}{2}, & if\ Y^j_{i,g} > X^j_{\max} \end{cases} \tag{23}$$

### 3.3. Putting it all together

After diffusion using Differential Evolution (DvDE) and diversifying via the success-based update processes (S-UP1 and S-UP2), the last phase introduces a way to adapt the population size. The aim of using this adaptation is to use a dynamic scheme to change the population size *NP* for each generation based on logical search phases (exploration at the beginning and more exploitation at later stages of the search) and to improve the performance of the proposed algorithm. The pseudo-code of the whole proposed algorithm, *d*DSF-EA, with dynamic population size reduction, is presented in Fig. 5. The algorithm starts by uniformly distributed random individuals, as given in Step 1 of pseudo-code, using Eq. (2).

The next step is to call the Differential Diffusion process, DvDE as presented in Fig. 2. In the first generation, the two update processes are called with a participation ratio equal to $\frac{NP}{2}$. Subsequently, the success-based scheme is used to assign the participation ratios based on previous success as Figs 3 and 4 present. At the end of the optimization loop, the population size reduction routine is performed to determine the population size for the next generation $t+1$ as steps 9–12 show. A linear reduction is used to reduce the size of the population as shown in Eq. (24), favoring more exploitation as the time lapses. The $(NP_g - NP_{g+1})$ worst individuals according to fitness value are eliminated from the population.

$$NP_{g+1} = Round\left[\left(\frac{NP_{\min} - NP_{g_0}}{FEs_{\max}}\right).FEs + NP_g\right] \tag{24}$$

**Algorithm: Stochastic Fractal Search with Differential Evolution and Population size reduction**

**Input:** Input parameters of Differential Evolution Diffusion process
Input parameters of two update processes
Lower and upper bounds of problem being solved $\left[X_{\min}, X_{\max}\right]$

**Output:** Optimal solution of an optimization problem

1. Initialize population at first generation $g_0$, $P_{g_0} = \left\langle X_{1,g_0}, \ldots, X_{NP,g_0} \right\rangle$ where $X_{i,g} = (x^1_{i,g_0}, x^2_{i,g_0}, \ldots, x^D_{i,g_0})$, $i$=1,2,…,$NP$, are uniformly distributed within $X_{\min} = (x^1_{\min}, \ldots, x^D_{\min})$ and $X_{\max} = (x^1_{\max}, \ldots, x^D_{\max})$
2. Set $\mu_F = 0.5$, $\mu_{CR} = 0.5$ and $A = P_{g_0}$
3. **While termination criterion is not met**
4. **Call** Diffusion via Differential Evolution (DvDE) process to generate new offspring as shown in Fig. 2
5. **If** $g == 1$
6. **Set** participation ratio $N = NP / 2$
7. **Else**
8. **Set** participation ratio $N$ using Eqs. 21 and 22
9. **EndIf**
10. **Call modified success-based** update process, S-UP1, as shown in Fig. 3
11. **Call modified success-based** update process, S-UP2, as shown in Fig. 4
12. **Apply** Eq. 24 to calculate population size for next generation $NP_{g+1}$
13. Calculate $N_{diff} = NP_g - NP_{g+1}$
14. Sort individuals $P_g$ based on its fitness
15. Eliminate worst individuals $N_{diff}$ from $P_g$
16. **EndWhile**
17. **Terminate and return output**

**Fig. 5.** Pseudo-code of decremental differential stochastic fractal evolutionary algorithm (dDSF-EA).

where $NP_{g0}$ is the population size used at the first generation $g_0$, $FEs$ is current function evaluation, $FEs_{\max}$ is the maximum number of function evaluations. $NP_{\min}$ is the smallest possible value that is needed to perform mutation strategy which is 4 in our case and $NP(t)$ is the current population size at generation $t$.

## 4. Experiments and analysis

### 4.1. Numerical benchmarks

The *d*DSF-EA algorithm is tested using the benchmark functions from the special session and competition on single objective real-parameter numerical optimization held under the IEEE CEC 2014 [31]. Such benchmark functions span a various set of problem features and characteristics, including ruggedness, noise in fitness, multimodality, ill-conditioning, interdependence and non-separability. These benchmarks are based on classical functions such as Rastrigin's, Swefel's, Rosenbrick's, Ackley's, and Griewank's functions. More details on these functions are given in [31]. To summarize, functions $f_1$–$f_3$ are unimodal, functions $f_4$–$f_{16}$ are multimodal, functions $f_{17}$–$f_{22}$ are hybrid, and functions $f_{23}$–$f_{30}$ are composition functions.

### 4.2. Compared algorithms and parametric setup

In this section, the performance of *d*DSF-EA is compared with other state-of-the-art algorithms as well as other reported results from the literature. The compared algorithms can be distinguished into two different classes of algorithms. These algorithms were discussed in Section 1. The first seven algorithms (dynNP-jDE, OXDE, JADE, SaDE, SHADE, L-SHADE, and SFS) include pure homogeneous algorithms with single and/or adaptive mutation scheme algorithms. The last five algorithms (EPSDE, CoDE, CoBiDE, UMOEAs, and AMALGAM) include amalgam algorithms and/or an ensemble of mutations schemes algorithms. For all the contestant algorithms, unless otherwise stated, we employ the best suited parametric setup for the employed benchmark suite, chosen with guidelines from their respective literature. The rest of the parameters are adaptively set during the search.

**Table 1**
Sensitivity test for different values for maximum diffusion $\gamma$. Mean best entry is marked in bold.

| | $\gamma = 1$ | $\gamma = 2$ | $\gamma = 3$ | $\gamma = 4$ |
|---|---|---|---|---|
| F1 | **2.9831E+05** | 1.0754E+06 | 2.2760E+06 | 5.8142E+06 |
| F2 | **5.6482E+03** | 7.3480E+03 | 5.6540E+05 | 1.2550E+07 |
| F4 | **9.6694E+01** | 9.7744E+01 | 1.0215E+02 | 1.1489E+02 |
| F15 | **1.3166E+01** | 1.8242E+01 | 2.3456E+01 | 2.5747E+01 |
| F17 | **2.1987E+03** | 3.8976E+03 | 1.3498E+04 | 5.6101E+04 |
| F22 | **5.8325E+02** | 7.6706E+02 | 8.9113E+02 | 8.9036E+02 |
| F29 | **1.8267E+03** | 1.1575E+04 | 2.2932E+04 | 4.7798E+04 |
| F30 | **8.8380E+03** | 9.6810E+03 | 1.2306E+04 | 1.5976E+04 |

**Table 2**
Algorithm complexity.

| | $T_0$ | $T_1$ | $\hat{T}_2$ | $(\hat{T}_2 - T_1)/T_0$ |
|---|---|---|---|---|
| $D = 10$ | 0.1548 | 0.5651 | 1.0214 | 2.947674 |
| $D = 30$ | | 1.5412 | 2.4789 | 6.057494 |
| $D = 50$ | | 2.8745 | 4.5784 | 11.00711 |
| $D = 100$ | | 8.9547 | 11.6472 | 17.39341 |

1. dynNP-jDE [10] with a number of different population sizes ($pmax$)=4 and $NP$= 200.
2. OXDE [44] with the orthogonal array, $L_9(3^4)$ is used and DE/rand/1/bin is used as mutation strategy with $NP$=$D$, $F$=0.9, $CR$=0.9.
3. JADE with $NP$=100 [52].
4. SaDE with $NP$=50 [36].
5. SHADE with the archive rate=2.0 and $NP$=100 [40].
6. L-SHADE with the archive rate=2.0, a population that linearly decreases to 4.0 starting from 18×$D$ [41].
7. SFS where the first Gaussian walk (Eq. 10) is used, the maximum diffusion number ($\gamma$) is set to 1 and the population size is set to 100. [37].
8. EPSDE with $NP$=50 [53].
9. CoDE with $NP$ =30 [43].
10. CoBiDE with $NP$ is set to 60, the probability to execute DE based on covariance matrix, $pb$ is set to 0.4, and the proportion of the individuals chosen to calculate the covariance matrix, $ps$ is set to 0.5. [45].
11. UMOEAs with $PS_1 = PS_2 = PS_3 = 100$, $CS = 100$ and $\mu = 2$. $p = 0.1$, $arch = PS_1/2$, $Cr = 100\%$, mutation rate = 0.1, tournament selection is randomly 2 or 3, $\eta = 3$, and $b = 5$. $\mu_{\text{CMA}} = 0.5PS_3$, $\sigma = 1.5$ [20].
12. AMALGAM-SO [42] with the minimum number of individuals needs to contribute in GA, PCX, PSO and DE is set to 5% of the population size and 25% of the population size for the CMA-ES.

For parameters of $d$DSF-EA, the population size ($NP$) undergoes a linear reduction of $NP_{\max} = 18\times D$ to $NP_{\min} = 4$. Maximum diffusion $\gamma$ is set to 1. $\mu_{\text{F}}$ and $\mu_{\text{CR}}$ are initially set to 0.5. The memory size for storing successful $F$ and $CR$ are both set to 10. For L-SHADE engine, we used the same parameter settings that the authors mentioned after doing the parameters tuning. For SFS, the only parameter needs to be tested is maximum diffusion $\gamma$. Table 1 shows the mean values for different values of this parameter. For space limitation, we selected two functions from the four different sets of the used benchmark, (F1, F2) are from unimodal functions, (F4, F15) are from simple multimodal functions, (f17, f22) are from hybrid functions and (F29, F30) are from composition functions. This table shows that increasing the size of diffusion degrade the performance very clearly. As a result, the suitable value is set to 1.

### 4.3. Algorithm complexity

The algorithm complexity for $d$DSF-EA is shown in Table 2. The algorithm was coded using Matlab 2013a and was run on a PC with an Intel CPU (3.40 GHz) and 8GB RAM. The computational complexity of $d$DSF-EA algorithm is calculated as described in [31]. In Table 1, $T_0$ denotes the running time of the following program:

$$
\begin{aligned}
&for\ i = 1 : 1000000\\
&\quad x = 0.55 + (double)i;\ x = x + x;\ x = x/2;\ x = x * x;\\
&\quad x = sqrt(x);\quad x = \log(x);\ x = \exp(x);\ x = x/(x + 2);\\
&end
\end{aligned}
$$

$T_1$ is the computing time for $f_{18}$ for 200,000 evaluations while $T_2$ is the complete running time for the proposed algorithm of $f_{18}$ for 200,000 evaluations. $T_2$ is evaluated five times, and the mean for $T2$ is denoted as $\widehat{T}_2$. Finally, the algorithm complexity is reflected by $\widehat{T}_2, T_1$ and $(\widehat{T}_2 - T_1)/T_0$).

### 4.4. Simulation schemes

The 30 benchmark problems are tested in 50$D$. In order to evaluate how the performance of the proposed algorithm adapts with scaling the search space, a scalability test at 100$D$ is also provided for all the functions. Following the guidelines provided in the IEEE CEC 2014 special session and competition [31], the maximum number of function evaluations (FEs) was set to $5\times10^5$ for 50$D$ and $1\times10^6$ for 100$D$. The simulations were done on an Intel® Core™ i7 CPU at 2.3 GHz speed. All of the DE variant algorithms were started from the same initial population in each run to attribute any differences in the performance to internal search operators of the algorithms.

### 4.5. Results

The mean and the standard deviation of the best-of-run errors for 51 independent runs of all the contestant algorithms on the 30 benchmark problems for $D = 50$ are given in Tables 3 and 4. The error is the absolute value of the change between the real optimum value of the objective function $f_{\text{opt}}$ and the best result $f(\vec{X}_{best})$, i.e., $|f_{opt} - f(\vec{X}_{best})|$. The comparison has been divided into two tables based on the algorithms. Table 3 presents a comparison against a set of algorithms that contain canonical and/or adaptive DE variants with single mutation. On the other hand, Table 4 shows the results of comparison against a set of algorithms that are either hybrid (as in UMOEA, CoBiDE and Amalgam algorithms) or with an ensemble of mutation strategies (as in CoDE and EPSDE algorithms).

A two-sided Wilcoxon rank sum test [48] is used to evaluate the statistical significance of the observed differences between $d$DSF-EA and every contestant algorithm. At the 5% significance level, the tested hypothesis ($H_0$) was that the compared results are independent ones from identical continuous distributions. As indicated in the results tables (Tables 3), a "–" sign signifies the case where the compared algorithm shows an inferior performance, and a "+" sign identifies when it exhibits superior performance. In such cases $H_0$ is rejected. When the performance difference is not statistically significant, these cases are marked with a "=" sign. We also show the total number of the aforementioned cases at the end of the second table of each dimension, for each of the competitor algorithms as (+/=/–).

As indicated in Table 3, and considering the mean of the error values for 50$D$ problems and as noted from results of the Wilcoxon test, $d$DSF-EA showed a statistically significant performance when compared with all the contestant algorithms in most of the functions. The last row of each of Tables 3–6 indicates a summary of the score of each algorithm in terms of wins "$w$", ties "$t$" and losses "$l$" over all the 30 functions, as indicated by the Wilcoxon test, compared to $d$DSF-EA. The comparison of $d$DSF-EA against the first category of algorithms indicates that the proposed algorithm was at least as good as the other competing algorithms in 18 functions. The second performing algorithm was L-SHADE where it performs better than $d$DSF-EA in 4 functions, shows a similar performance in 12 functions, and shows an inferior performance in 14 functions. A careful scrutiny of the results of the comparison with the state-of-the-art algorithms in the second category of algorithms (Table 4) shows that $d$DSF-EA obtained the smallest best-of-the-run errors over the majority of the functions at 50$D$. Compared to the second best performing algorithm (CoBiDE), $d$DSF-EA shows a superior performance in 21 functions, equal performance in 6 functions and performed worse in 3 functions. According to Tables 3 and 4, one can notice that the proposed algorithm performs well in unimodal functions (F1–F3) on both dimensions except that the UMOEA algorithm is better in F1 on $50D$. For simple multimodal functions, F4–F16, the proposed algorithm was the best algorithm in F6–F11, F15 and F16 and was the second best in F12–F14, being outperformed only by SFS in F12 and L-SHADE in F14. For the hybrid (F17–F22) and composition functions (F23–F30), the proposed algorithm shows superior performance compared to all other algorithms. Exceptions are noted when proposed algorithm ranked second after CoBiDE when tested on F18–F19 from the hybrid category and being ranked second after EPSDE when tested on F29–F30 from the composition category.

A scalability study using 100$D$ was performed following the specifications of the IEEE CEC2014 special session and competition as can be seen in Tables 5 and 6. The relative performance of $d$DSF-EA was not substantially decreased with increasing of the dimensionality to 100$D$. As revealed by Table 5, $d$DSF-EA obtained the best performance over the majority of these functions when compared to the other contestant algorithms. It was able to obtain better results in 18 functions compared to the second best performing algorithm (SHADE), equal performance in 5 functions, and a lower performance in 7 functions. $d$DSF-EA obtained better results than L-SHADE, the announced winner of the IEEE CEC special session and competition [31], in 13 functions and an inferior performance in only 4 functions. Inspecting the results of the second category of compared algorithms at 100$D$ as shown in Table 6 shows that $d$DSF-EA was able to obtain statistically significant results compared to the UMOEAs in 16 functions, equal performance in 4 functions and an inferior performance in 10 functions. Similar as 50$D$, $d$DSF-EA shows competitive performance in solving most of the hybrid and composition functions with clear difference in comparison to the contestant algorithms.

Furthermore, an extensive statistical analysis [17] was added for the purpose of evaluating the statistical significance of observed performance differences. For an input of $n$ algorithms, this analysis employs a statistical test procedure to rank the performance of each compared algorithm. The test highlights whether there are statistical significant differences in the

**Table 3**
Mean and standard deviation of the error values for functions f1-f30 @ 50D. Best entries are marked in boldface.

| | dynNP-jDE | OXDE | JADE | SaDE | SHADE | L-SHADE | SFS | *d*DSF-EA |
|---|---|---|---|---|---|---|---|---|
| F1 | 3.2264E+05 (1.1734E+05) – | 1.2846E+06 (7.0510E+05) – | 1.5275E+04 (1.3422E+04) – | 9.3147E+05 (3.1419E+05) – | 1.8614E+04 (1.3638E+04) – | 1.2000E+03 (1.5154E+03) – | 1.0380E+06 (3.4650E+05) – | **4.0970E+02 (5.1974E+02)** |
| F2 | 3.3251E−07 (1.4702E−06) – | 2.7074E+01 (3.5785E+01) – | **0.0000E+00 (0.0000E+00)** = | 3.7564E+03 (3.8115E+03) – | **0.0000E+00 (0.0000E+00)** = | **0.0000E+00 (0.0000E+00)** = | 7.9599E+03 (7.7780E+03) – | **0.0000E+00 (0.0000E+00)** |
| F3 | 7.0324E−06 (4.7809E−05) – | 1.7754E+01 (3.3507E+01) – | 4.5134E+03 (2.3441E+03) – | 3.4631E+03 (2.2424E+03) – | **0.0000E+00 (0.0000E+00)** = | **0.0000E+00 (0.0000E+00)** = | 1.6841E+00 (2.4293E+00) – | **0.0000E+00 (0.0000E+00)** |
| F4 | 8.1654E+01 (2.1947E+01) – | 5.6840E+01 (3.5706E+01) – | 1.9346E+01 (3.9290E+01) + | 8.2104E+01 (4.4460E+01) – | **9.7224E+00 (2.9435E+01)** + | 5.8866E+01 (4.5608E+01) – | 9.6504E+01 (4.8574E+01) – | 4.6038E+01 (4.7699E+01) |
| F5 | 2.0384E+01 (2.9280E−02) = | 2.1138E+01 (3.3878E−02) – | 2.0356E+01 (3.6845E−02) = | 2.0733E+01 (4.1549E−02) = | **2.0140E+01 (1.9410E−02)** = | 2.0249E+01 (4.5920E−02) = | 2.0429E+01 (1.7556E−01) = | 2.0258E+01 (2.8409E−02) |
| F6 | 7.3352E+00 (5.6880E+00) – | 4.3734E+00 (1.8132E+00) – | 1.6589E+01 (6.6285E+00) – | 1.8148E+01 (3.2634E+00) – | 5.1707E+00 (2.4284E+00) – | **2.6408E−01 (5.2278E−01)** = | 2.4186E+01 (3.8939E+00) – | 4.2789E−01 (6.2441E−01) |
| F7 | **0.0000E+00 (0.0000E+00)** = | 1.7391E−03 (4.5755E−03) – | 2.0769E−03 (4.9084E−03) – | 1.3887E−02 (1.5352E−02) – | 3.9093E−03 (7.4077E−03) – | **0.0000E+00 (0.0000E+00)** = | 1.0624E−03 (3.5592E−03) – | **0.0000E+00 (0.0000E+00)** |
| F8 | **0.0000E+00 (0.0000E+00)** = | 2.7683E+01 (6.4809E+00) – | **0.0000E+00 (0.0000E+00)** = | 1.0925E+00 (1.0945E+00) – | **0.0000E+00 (0.0000E+00)** = | 2.5817E−09 (7.4842E−09) – | 6.1843E+01 (3.0497E+01) – | **0.0000E+00 (0.0000E+00)** |
| F9 | 7.4286E+01 (9.3923E+00) – | 1.7698E+02 (1.1903E+02) – | 5.1867E+01 (8.6673E+00) – | 9.1673E+01 (1.4268E+01) – | 3.4182E+01 (6.1592E+00) – | 1.1636E+01 (2.1338E+00) = | 2.0178E+02 (4.0628E+01) – | **1.1611E+01 (2.0545E+00)** |
| F10 | 6.8582E−03 (9.7767E−03) – | 2.8633E+02 (2.8898E+02) – | 1.2247E−02 (1.3099E−02) + | 1.2284E+00 (1.0759E+00) – | **1.0287E−02 (1.1368E−02)** + | 1.2188E−01 (4.1286E−02) – | 1.6675E+02 (1.5591E+02) – | 6.7828E−02 (2.6934E−02) |
| F11 | 4.3297E+03 (3.8575E+02) – | 1.3195E+04 (3.8675E+02) – | 3.8695E+03 (2.7785E+02) – | 6.8299E+03 (1.5764E+03) – | 3.5552E+03 (3.2722E+02) – | **3.2219E+03 (3.2983E+02)** + | 4.7934E+03 (6.9382E+02) – | 3.3280E+03 (3.4911E+02) |
| F12 | 3.6401E−01 (4.8652E−02) = | 3.2787E+00 (3.5656E−01) – | 2.5535E−01 (3.1321E−02) = | 1.0986E+00 (1.0537E−01) – | 1.6366E−01 (2.1190E−02) = | 2.1891E−01 (2.8176E−02) = | **1.4197E−01 (8.4787E−02)** = | 2.2020E−01 (2.6991E−02) |
| F13 | 3.4874E−01 (4.1091E−02) – | 3.4547E−01 (5.5859E−02) – | 3.3223E−01 (4.9590E−02) – | 4.3681E−01 (5.4721E−02) – | 3.1034E−01 (4.5483E−02) – | **1.6043E−01 (1.8317E−02)** = | 5.2075E−01 (8.3969E−02) – | 1.6252E−01 (2.2208E−02) |
| F14 | 3.0009E−01 (2.6630E−02) = | 2.6537E−01 (4.5895E−02) = | 3.1348E−01 (9.1988E−02) = | 3.1170E−01 (3.2642E−02) = | **2.9453E−01 (5.7219E−02)** = | 3.0808E−01 (2.4695E−02) = | 3.4798E−01 (1.3415E−-01) = | 3.0843E-−01 (2.8188E−02) |
| F15 | 1.0077E+01 (1.1616E+00) – | 2.9733E+01 (1.6981E+00) – | 7.0826E+00 (9.4236E−01) – | 1.7092E+01 (5.8042E+00) – | 5.7376E+00 (6.1481E−01) – | 5.2056E+00 (5.0768E−01) = | 1.3024E+01 (3.2486E+00) – | **5.1771E+00 (4.4868E−01)** |
| F16 | 1.7551E+01 (4.6595E−01) = | 2.2114E+01 (2.7946E−01) – | 1.7720E+01 (4.3341E−01) – | 2.0222E+01 (2.7357E−01) – | 1.7444E+01 (4.2728E−01) – | 1.7014E+01 (4.8120E−01) = | 1.8089E+01 (7.0462E−01) – | **1.6997E+01 (4.1221E−01)** |
| F17 | 1.2583E+04 (8.3862E+03) – | 2.7195E+04 (2.2236E+04) – | 2.3000E+03 (5.9502E+02) – | 5.9118E+04 (3.9220E+04) – | 2.2347E+03 (7.8109E+02) – | 1.4543E+03 (5.1303E+02) – | 1.1219E+04 (1.0968E+04) – | **1.4317E+03 (3.6679E+02)** |
| F18 | 2.5984E+02 (2.4692E+02) – | 1.4232E+02 (1.2845E+01) – | 1.8203E+02 (4.7150E+01) – | 7.5884E+02 (5.1263E+02) – | 1.5375E+02 (3.6617E+01) – | 1.0213E+02 (1.3841E+01) – | 8.6313E+02 (1.2040E+03) – | **1.0014E+02 (1.3119E+01)** |
| F19 | 1.0832E+01 (8.4404E−01) – | 1.1660E+01 (1.4518E+00) – | 1.3421E+01 (5.2199E+00) – | 1.6741E+01 (8.7552E+00) – | 9.3810E+00 (3.2466E+00) – | **8.2960E+00 (1.8136E+00)** = | 1.6699E+01 (1.3771E+01) – | 8.3073E+00 (1.9160E+00) |
| F20 | 3.9163E+01 (1.3337E+01) – | 1.8574E+02 (4.5768E+01) – | 7.9529E+03 (6.9640E+03) – | 9.0611E+02 (7.3143E+02) – | 1.9816E+02 (5.7311E+01) – | **1.3914E+01 (4.5644E+00)** + | 1.0889E+02 (3.5584E+01) – | 1.4126E+01 (4.3694E+00) |
| F21 | 2.7281E+03 (2.5200E+03) – | 7.3588E+03 (6.3316E+03) – | 2.4322E+04 (1.6455E+05) – | 8.5745E+04 (4.5410E+04) – | 1.2297E+03 (4.0604E+02) – | **5.2593E+02 (1.4909E+02)** + | 2.3415E+03 (2.2556E+03) – | 5.3548E+02 (1.3736E+02) |
| F22 | 4.2383E+02 (1.2826E+02) – | 3.8417E+02 (2.4172E+02) – | 5.1982E+02 (1.3024E+02) – | 5.1001E+02 (1.7759E+02) – | 3.6769E+02 (1.5946E+02) – | **1.1716E+02 (7.5047E+01)** + | 6.6906E+02 (2.1832E+02) – | 1.2436E+02 (7.2157E+01) |
| F23 | 3.4400E+02 (1.1482E−13) – | 3.4400E+02 (2.8705E−13) – | 3.4400E+02 (3.7646E−13) – | 3.4400E+02 (1.1482E−13) – | 3.4400E+02 (1.1482E−13) – | 3.4400E+02 (4.4592E−13) – | **2.0000E+02 (0.0000E+00)** = | **2.0000E+02 (0.0000E+00)** |

(*continued on next page*)

**Table 3** (*continued*)

| | dynNP-jDE | OXDE | JADE | SaDE | SHADE | L-SHADE | SFS | *d*DSF-EA |
|---|---|---|---|---|---|---|---|---|
| F24 | 2.6546E+02 (1.6705E+00) – | 2.8552E+02 (3.1975E+00) – | 2.7482E+02 (1.9487E+00) – | 2.7596E+02 (3.0905E+00) – | 2.7478E+02 (1.7334E+00) – | 2.7522E+02 (6.6170E−01) – | **2.0000E+02 (0.0000E+00) =** | **2.0000E+02 (0.0000E+00)** |
| F25 | 2.0691E+02 (1.3449E+00) – | 2.0584E+02 (7.1385E−01) – | 2.1699E+02 (6.7636E+00) – | 2.1665E+02 (9.7720E+00) – | 2.1217E+02 (6.9165E+00) – | 2.0715E+02 (3.6488E−01) – | **2.0000E+02 (0.0000E+00) =** | **2.0000E+02 (0.0000E+00)** |
| F26 | 1.0035E+02 (4.1976E−02) = | 1.0033E+02 (4.9095E−02) = | 1.0237E+02 (1.3950E+01) – | 1.9030E+02 (2.9941E+01) – | 1.0431E+02 (1.9529E+01) – | 1.0212E+02 (1.3980E+01) – | 1.9414E+02 (2.3667E+01) – | **1.0016E+02 (1.7444E−02)** |
| F27 | 3.8452E+02 (4.8217E+01) – | 4.1152E+02 (4.7756E+01) – | 4.4058E+02 (5.6802E+01) – | 7.8213E+02 (6.2606E+01) – | 4.5329E+02 (5.6995E+01) – | 3.3486E+02 (3.0280E+01) – | **2.0000E+02 (0.0000E+00) =** | **2.0000E+02 (0.0000E+00)** |
| F28 | 1.1052E+03 (3.8147E+01) – | 9.1392E+02 (3.1211E+01) – | 1.1289E+03 (4.1317E+01) – | 1.4354E+03 (9.8737E+01) – | 1.1413E+03 (5.2616E+01) – | 1.1124E+03 (2.9101E+01) – | **2.0000E+02 (0.0000E+00) =** | **2.0000E+02 (0.0000E+00)** |
| F29 | 9.9798E+02 (1.4277E+02) – | 1.3458E+03 (4.8515E+02) – | 9.0482E+02 (1.2689E+02) – | 1.4224E+03 (3.1715E+02) – | 8.9071E+02 (5.5375E+01) – | 8.0091E+02 (2.4010E+01) – | 5.0385E+02 (5.5204E+02) – | **3.0823E+02 (3.5311E+02)** |
| F30 | 8.4860E+03 (4.2315E+02) – | 1.1986E+04 (4.4610E+02) – | 9.7197E+03 (8.1792E+02) – | 1.1613E+04 (1.8321E+03) – | 9.3841E+03 (7.8714E+02) – | 8.7599E+03 (4.1306E+02) – | 8.7778E+03 (4.0000E+03) – | **8.7135E+03 (3.8407E+02)** |
| *w*/*t*/*l* | 0/7/23 | 0/2/28 | 2/5/23 | 0/2/28 | 2/6/22 | 4/12/14 | 0/8/22 | |

**Table 4**
Mean and standard deviation of the error values for functions f1-f30 @ 50D. Best entries are marked in boldface.

| | EPSDE | CoDE | CoBiDE | UMOEAs | AMALGAM | *d*DSF-EA |
|---|---|---|---|---|---|---|
| F1 | 1.7374E+06 (5.5064E+06) – | 2.3214E+05 (1.0747E+05) – | 2.7108E+05 (1.1421E+05) – | **0.0000E+00 (0.0000E+00)** + | 1.8272E-02 (2.3181E-02) + | 4.0970E+02 (5.1974E+02) |
| F2 | 1.4999E−08 (3.4609E−08) – | 6.5010E+01 (1.9284E+02) – | 2.1271E+00 (9.3876E+00) – | **0.0000E+00 (0.0000E+00)** = | 1.0773E+04 (1.0150E+04) – | **0.0000E+00 (0.0000E+00)** |
| F3 | 3.7908E−04 (1.8292E−03) – | 2.5387E+01 (4.1188E+01) – | 1.2074E−02 (3.4056E−02) – | **0.0000E+00 (0.0000E+00)** = | 1.1576E−03 (4.6341E−04) – | **0.0000E+00 (0.0000E+00)** |
| F4 | 4.1596E+01 (2.2985E+01) + | 2.6067E+01 (3.3789E+01) + | **3.1804E+01 (3.2589E+01)** + | 8.0872E+01 (3.7809E+01) – | 3.8392E+01 (4.7627E+01) + | 4.6038E+01 (4.7699E+01) |
| F5 | 2.0596E+01 (3.2400E−02) = | 2.0032E+01 (6.8626E−02) **=** | 2.0207E+01 (3.3036E−01) = | 2.0155E+01 (2.0579E−01) = | **2.0019E+01 (2.9245E−02)** = | 2.0258E+01 (2.8409E−02) |
| F6 | 4.5596E+01 (2.6277E+00) – | 8.4983E+00 (3.3570E+00) – | 4.4657E+00 (2.7394E+00) – | 1.4615E+00 (1.5197E+00) – | 9.5012E+00 (4.2842E+00) – | **4.2789E−01 (6.2441E−01)** |
| F7 | 5.1192E−03 (7.5766E−03) – | 1.5467E−03 (3.1935E−03) – | **0.0000E+00 (0.0000E+00)** = | **0.0000E+00 (0.0000E+00)** = | 2.4382E−03 (1.4124E−03) – | **0.0000E+00 (0.0000E+00)** |
| F8 | 1.9509E−02 (1.3932E−01) – | 4.8773E−01 (8.2941E−01) – | **0.0000E+00 (0.0000E+00)** = | 7.6281E+00 (5.2067E+00) – | 4.2638E+01 (1.6509E+01) – | **0.0000E+00 (0.0000E+00)** |
| F9 | 1.4593E+02 (1.6322E+01) – | 8.2582E+01 (1.7230E+01) – | 8.3200E+01 (1.8512E+01) – | 2.0965E+01 (5.0632E+00) – | 9.0612E+01 (1.7436E+01) – | **1.1611E+01 (2.0545E+00)** |
| F10 | 5.5334E+02 (6.0708E+02) – | 5.1856E+00 (3.0648E+00) – | 2.5472E+02 (5.4541E+01) – | 8.7297E+01 (1.2338E+02) – | 1.8827E+03 (5.5230E+02) – | **6.7828E−02 (2.6934E−02)** |
| F11 | 8.9455E+03 (5.3473E+02) – | 4.3391E+03 (9.4713E+02) – | 4.1635E+03 (6.1871E+02) – | 3.4497E+03 (1.5371E+03) – | 4.3470E+03 (8.0502E+02) – | **3.3280E+03 (3.4911E+02)** |
| F12 | 8.5469E−01 (6.9658E−02) – | 8.7329E−02 (4.5691E−02) + | 1.2484E−01 (2.3175E−01) = | **1.3021E−03 (8.7544E−04)** + | 2.9995E−02 (8.6099E−03) + | 2.2020E−01 (2.6991E−02) |

(*continued on next page*)

**Table 4** (*continued*)

| | EPSDE | CoDE | CoBiDE | UMOEAs | AMALGAM | *d*DSF-EA |
|---|---|---|---|---|---|---|
| F13 | 3.6268E−01 (5.9205E−02) − | 3.3636E−01 (4.9736E−02) − | 3.3022E−01 (4.7479E−02) − | **1.1041E−01 (3.2770E−02)** = | 2.9013E−01 (5.9834E−02) = | 1.6252E−01 (2.2208E−02) |
| F14 | 3.5209E−01 (9.0136E−02) = | 2.7652E−01 (2.9839E−02) = | **2.4902E−01 (3.0336E−02)** = | 2.7956E−01 (3.2886E−02) = | 5.2434E−01 (2.4602E−01) − | 3.0843E−01 (2.8188E−02) |
| F15 | 1.8616E+01 (2.6294E+00) − | 7.4043E+00 (1.3955E+00) − | 6.4988E+00 (1.3866E+00) − | 5.5483E+00 (9.4556E−01) = | 6.2105E+00 (1.0093E+00) − | **5.1771E+00 (4.4868E−01)** |
| F16 | 2.0695E+01 (4.1566E−01) − | 1.8290E+01 (8.9113E−01( − | 1.8442E+01 (8.6659E−01) − | 1.9333E+01 (7.9549E−01) − | 2.0606E+01 (7.0027E−01) − | **1.6997E+01 (4.1221E−01)** |
| F17 | 2.2167E+05 (1.4896E+05) − | 1.5714E+04 (1.3291E+04) − | 1.2534E+04 (1.1574E+04) − | 2.5196E+03 (5.4530E+02) − | 2.7677E+03 (5.5196E+02) − | **1.4317E+03 (3.6679E+02)** |
| F18 | 3.6879E+03 (8.1501E+03) − | 3.2916E+02 (3.4839E+02) − | **2.8929E+01 (9.8444E+00)** + | 1.1394E+02 (6.6103E+01) − | 2.2486E+02 (5.9791E+01) − | 1.0014E+02 (1.3119E+01) |
| F19 | 2.4584E+01 (1.6861E+00) − | **6.1691E+00 (9.8380E−01)** + | 6.7325E+00 (1.2317E+00) + | 1.0376E+01 (1.5351E+00) − | 2.1313E+01 (7.6461E+00) − | 8.3073E+00 (1.9160E+00) |
| F20 | 4.5080E+02 (5.6348E+02) − | 2.1861E+02 (2.3515E+02) − | 3.9996E+01 (1.6285E+01) − | 8.1586E+01 (3.2670E+01) − | 3.7317E+02 (1.0774E+02) − | **1.4126E+01 (4.3694E+00)** |
| F21 | 7.4370E+04 (8.5331E+04) − | 6.4912E+03 (5.5372E+03) − | 3.7929E+03 (3.3289E+03) − | 1.5664E+03 (3.6130E+02) − | 1.7085E+03 (3.9523E+02) − | **5.3548E+02 (1.3736E+02)** |
| F22 | 8.0373E+02 (2.0458E+02) − | 6.0223E+02 (2.2061E+02) − | 5.6048E+02 (2.2255E+02) − | 2.9400E+02 (1.6073E+02) − | 8.1481E+02 (2.7265E+02) − | **1.2436E+02 (7.2157E+01)** |
| F23 | 3.3705E+02 (3.4523E−12) − | 3.4400E+02 (1.1482E−13) − | 3.4400E+02 (4.1053E−13) − | 3.4401E+02 (3.5179E−13) − | 3.4401E+02 (3.1354E−03) − | **2.0000E+02 (0.0000E+00)** |
| F24 | 2.7312E+02 (5.7937E+00) − | 2.7139E+02 (2.9312E+00) − | 2.6971E+02 (4.9786E+00) − | 2.7158E+02 (4.4989E+00) − | 2.7370E+02 (5.3595E+00) − | **2.0000E+02 (0.0000E+00)** |
| F25 | 2.0116E+02 (2.0229E+00) − | 2.0792E+02 (4.7505E+00) − | 2.0042E+02 (1.0850E−01) = | 2.0011E+02 (7.8507E−01) = | 2.0121E+02 (3.9002E−01) = | **2.0000E+02 (0.0000E+00)** |
| F26 | 1.0035E+02 (4.7521E−02) − | 1.0618E+02 (2.3701E+01) − | 1.0227E+02 (1.3965E+01) − | 1.0210E+02 (1.3983E+01) − | 1.5358E+02 (5.0166E+01) − | **1.0016E+02 (1.7444E−02)** |
| F27 | 1.5566E+03 (5.8870E+01) − | 5.3680E+02 (6.5540E+01) − | 3.7451E+02 (4.8366E+01) − | 3.9515E+02 (5.1030E+01) − | 5.9863E+02 (7.8780E+01) − | **2.0000E+02 (0.0000E+00)** |
| F28 | 3.8746E+02 (1.1591E+01) − | 1.1734E+03 (4.4835E+01) − | 1.0176E+03 (1.5766E+02) − | 1.2523E+03 (7.4633E+01) − | 1.7041E+03 (3.0824E+02) − | **2.0000E+02 (0.0000E+00)** |
| F29 | **2.2556E+02 (1.1299E+01)** + | 9.2726E+02 (1.3813E+02) − | 9.2212E+02 (2.8885E+02) − | 1.0795E+03 (2.7595E+02) − | 7.1994E+02 (6.3409E+01) − | 3.0823E+02 (3.5311E+02) |
| F30 | **1.0618E+03 (2.0931E+02)** + | 9.0213E+03 (5.3860E+02) − | 1.1969E+04 (4.6813E+02) − | 8.8415E+03 (9.7809E+02) − | 1.4822E+04 (1.5294E+03) − | 8.7135E+03 (3.8407E+02) |
| *w*/*t*/*l* | 3/2/25 | 3/2/25 | 3/6/21 | 2/8/20 | 3/3/24 | |

performance ranking of at least one pair of these compared algorithms. For such purpose, the Aligned Friedman test [18] is a non-parametric multiple comparison test that was used to test the differences between the 13 compared algorithms (including *d*DSF-EA), and judge the performance of the different samples. Table 7 presents the rankings using the Aligned Friedman test. The last two rows in Table 7 show the test-statistic for this test and the corresponding measured *p*-value, respectively. These *p*-values suggest that there are significant differences among the compared algorithms for dimensions 50*D* and 100*D* at the 0.05 significance level. Friedman test assigns the lowest rank to the best performing algorithm. As can be seen in this table, the lowest rank score was obtained by *d*DSF-EA (rank@50*D* = 120.8209, rank@100*D* = 106.2615) with a clear difference compared to L-SHADE as the second-best performing algorithm (rank@50*D* = 151.4667, rank@100*D* = 150.4833).

For further illustration of the differences, the final performance results of the following chosen algorithms (SFS, CoDE, EPSDE, JADE, L-SHADE, SaDE, SHADE, AMALGAM-SO, and *d*DSF-EA) among the contestant algorithms, for some selected func-

**Table 5**
Mean and standard deviation of the error values for functions f1-f30 @ 100D. Best entries are marked in boldface.

| | dynNP-jDE | OXDE | JADE | SaDE | SHADE | L-SHADE | SFS | *d*DSF-EA |
|---|---|---|---|---|---|---|---|---|
| F1 | 1.5304E+06 (4.7144E+05) − | 1.5639E+07 (3.7771E+06) − | **1.1644E+05 (6.1455E+04)** + | 4.2708E+06 (8.9182E+05) − | 1.4280E+05 (1.0527E+05) + | 1.7188E+05 (5.6543E+04) − | 3.4063E+06 (8.1004E+05) − | 1.5481E+05 (4.4904E+04) |
| F2 | 2.1887E+02 (1.5629E+03) − | 1.1752E+04 (1.4936E+04) − | **0.0000E+00 (0.0000E+00)** = | 1.3273E+04 (9.9812E+03) − | **0.0000E+00 (0.0000E+00)** = | **0.0000E+00 (0.0000E+00)** = | 2.1043E+04 (1.5808E+04) − | **0.0000E+00 (0.0000E+00)** |
| F3 | 4.6258E−05 (1.6794E−04) − | 2.5071E+03 (1.1942E+03) − | 5.9129E+03 (3.7758E+03) − | 6.3462E+03 (3.7037E+03) − | 5.6315E−02 (2.1099E−01) − | **0.0000E+00 (0.0000E+00)** = | 1.2151E+03 (9.4582E+02) − | **0.0000E+00 (0.0000E+00)** |
| F4 | 1.6598E+02 (3.3829E+01) = | 2.1052E+02 (4.3079E+01) − | **7.7261E+01 (5.5558E+01)** + | 2.1284E+02 (6.0547E+01) − | 9.2526E+01 (4.6796E+01) + | 1.7035E+02 (3.0634E+01) − | 2.3361E+02 (4.9480E+01) − | 1.6063E+02 (2.6181E+01) |
| F5 | 2.0601E+01 (3.2027E−02) = | 2.1316E+01 (2.6776E−02) − | 2.0482E+01 (7.4319E−02) = | 2.1013E+01 (3.1972E−02) = | 2.0200E+01 (1.4669E−02) = | 2.0554E+01 (3.1110E−02) = | **2.0155E+01 (2.0534E−01)** = | 2.0564E+01 (3.6034E−02) |
| F6 | 4.0142E+01 (1.5199E+01) − | 1.9845E+01 (3.8338E+00) − | 4.4605E+01 (1.4840E+01) − | 7.5561E+01 (5.8575E+00) − | 3.3509E+01 (5.2490E+00) − | **8.6915E+00 (2.3336E+00)** + | 8.2195E+01 (8.9640E+00) − | 9.6903E+00 (2.7112E+00) |
| F7 | **0.0000E+00 (0.0000E+00)** + | 8.9344E−04 (2.7358E−03) = | 2.0760E−03 (6.0154E−03) − | 1.1012E−02 (3.3793E−02) − | 1.9787E−03 (5.3391E−03) − | **0.0000E+00 (0.0000E+00)** + | 2.0243E−03 (6.7272E−03) − | 1.4502E−04 (1.0357E−03) |
| F8 | **0.0000E+00 (0.0000E+00)** + | 2.0041E+02 (6.0449E+01) − | **0.0000E+00 (0.0000E+00)** + | 2.0016E+01 (6.4531E+00) − | **0.0000E+00 (0.0000E+00)** + | 1.1000E−02 (7.4351E−03) − | 3.8161E+02 (8.0468E+01) − | 4.1692E−03 (3.6511E−03) |
| F9 | 1.9410E+02 (2.2115E+01) − | 8.2835E+02 (2.4724E+01) − | 1.4425E+02 (2.0417E+01) − | 2.9217E+02 (3.5870E+01) − | 1.1400E+02 (1.5505E+01) − | **3.4079E+01 (5.0045E+00)** + | 6.4295E+02 (8.1853E+01) − | 3.8309E+01 (6.2988E+00) |
| F10 | **9.0627E−03 (8.0292E−03)** + | 1.5929E+04 (1.3511E+03) − | 1.5553E−02 (9.2955E−03) + | 3.5785E+01 (5.2364E+01) − | 1.0287E−02 (7.5803E−03) + | 2.6429E+01 (5.7878E+00) = | 1.2109E+03 (4.8627E+02) − | 2.5243E+01 (6.1027E+00) |
| F11 | 1.1520E+04 (6.8003E+02) − | 3.0360E+04 (5.3262E+02) − | 1.0730E+04 (5.4689E+02) − | 1.2648E+04 (2.6268E+03) − | **9.8049E+03 (4.9663E+02)** + | 1.1015E+04 (5.6250E+02) − | 1.2617E+04 (1.3411E+03) − | 1.0963E+04 (5.7809E+02) |
| F12 | 5.2194E−01 (3.9318E−02) = | 3.9774E+00 (2.5601E−01) − | 3.5887E−01 (1.2847E−01) = | 1.6530E+00 (1.2847E−01) − | 2.2939E−01 (2.4409E−02) + | 4.3723E−01 (4.6523E−02) = | **1.9954E−01 (6.6936E−02)** + | 4.5112E−01 (4.6843E−02) |
| F13 | 4.6015E−01 (5.0347E−02) − | 5.0080E−01 (4.7086E−02) − | 4.0910E−01 (4.0001E−02) − | 4.9867E−01 (4.2591E−02) − | 3.9960E−01 (4.2796E−02) = | 2.4083E−01 (2.0956E−02) = | (6.0664E−01 7.1825E−02) − | **2.3687E−01 (2.0001E−02)** |
| F14 | 1.3808E−01 (1.2602E−02) + | 3.1491E−01 (5.5916E-−02) = | 2.9028E+02 (2.8002E+02) − | 1.6850E−01 (1.3967E−02) + | 1.2012E−01 (9.7789E−03) + | **1.1781E−01 (7.3457E-−03)** + | 3.2705E−01 (6.5182E-−02) = | 3.3490E−01 (1.5611E−02) |
| F15 | 2.5898E+01 (2.3195E+00) − | 7.4016E+01 (2.8653E+00) − | 2.9730E+01 (3.9865E+00) − | 5.8790E+01 (1.1862E+01) − | 2.2126E+01 (2.8739E+00) − | 1.6182E+01 (1.1988E+00) = | 4.7043E+01 (1.0807E+01) − | **1.6573E+01 (1.2374E+00)** |
| F16 | 3.9314E+01 (5.1608E−01) = | 4.6636E+01 (2.3747E−01) − | 4.0091E+01 (5.6076E−01) − | 4.3865E+01 (3.3444E−01) − | 3.9591E+01 (4.9269E−01) = | 3.9146E+01 (4.7973E−01) = | 4.1295E+01 (9.6696E−01) − | **3.9258E+01 (5.2096E−01)** |
| F17 | 1.2110E+05 (4.9013E+04) − | 2.5058E+06 (7.3410E+05) − | 1.2039E+04 (4.8820E+03) − | 4.2373E+05 (1.6403E+05) − | 1.2354E+04 (4.7972E+03) − | **4.4345E+03 (7.1340E+02)** = | 3.9591E+05 (1.6848E+05) − | 4.4332E+03 (6.7563E+02) |
| F18 | 3.2104E+02 (3.8450E+02) − | 2.1438E+03 (2.0508E+03) − | 1.0748E+03 (9.5074E+02) − | 7.1544E+02 (4.3296E+02) − | 5.0701E+02 (3.8870E+02) − | 2.2364E+02 (1.6791E+01) = | 2.9481E+03 (2.6760E+03) − | **2.2252E+02 (1.8799E+01)** |
| F19 | 9.0449E+01 (1.2663E+00) + | 9.7603E+01 (6.0625E+00) = | 9.9450E+01 (8.3109E+00) − | **7.0697E+01 (2.8473E+01)** + | 9.7832E+01 (1.5491E+01) = | 9.5740E+01 (2.2879E+00) = | 7.4273E+01 (3.2266E+01) + | 9.6531E+01 (2.4790E+00) |
| F20 | 2.6591E+02 (5.6307E+01) − | 4.3229E+03 (2.1289E+03) − | 9.5316E+03 (1.5166E+04) − | 6.5476E+03 (3.1507E+03) − | 4.9152E+02 (1.3438E+02) − | 1.4992E+02 (5.2469E+01) − | 7.6721E+02 (3.2262E+02) − | **1.3970E+02 (5.0419E+01)** |
| F21 | 3.6871E+04 (1.5383E+04) − | 8.5478E+05 (2.8650E+05) − | 3.6989E+03 (1.1894E+03) − | 2.9005E+05 (1.1541E+05) − | 3.6938E+03 (1.6153E+03) − | 2.3163E+03 (5.3084E+02) − | 1.5657E+05 (7.3558E+04) − | **2.2415E+03 (5.2845E+02)** |
| F22 | 1.5195E+03 (1.8003E+02) − | 4.5779E+03 (5.3409E+02) − | 1.5963E+03 (2.3319E+02) − | 1.5644E+03 (3.6481E+02) − | 1.4005E+03 (2.1322E+02) − | 1.1061E+03 (1.8788E+02) − | 2.0722E+03 (4.5647E+02) − | **1.0542E+03 (1.9937E+02)** |
| F23 | 3.4823E+02 (1.1482E−13) − | 3.4824E+02 (1.1556E−03) − | 3.4823E+02 (9.0668E−13) − | 3.4823E+02 (1.5592E−07) − | 3.4823E+02 (4.5610E−13) − | 3.4823E+02 (3.6628E−13) − | **2.0000E+02 (0.0000E+00)** = | **2.0000E+02 (0.0000E+00)** |

(*continued on next page*)

**Table 5** (*continued*)

| | dynNP-jDE | OXDE | JADE | SaDE | SHADE | L-SHADE | SFS | *d*DSF-EA |
|---|---|---|---|---|---|---|---|---|
| F24 | 3.6749E+02 | 3.8902E+02 | 3.9879E+02 | 3.9636E+02 | 3.9436E+02 | 3.9440E+02 | **2.0000E+02** | **2.0000E+02** |
| | (2.6976E+00) | (4.9622E+00) | (5.2302E+00) | (8.1820E+00) | (4.4474E+00) | (2.8738E+00) | **(0.0000E+00)** | **(0.0000E+00)** |
| | − | − | − | − | − | − | = | |
| F25 | 2.5804E+02 | 2.1707E+02 | 2.7524E+02 | 2.4720E+02 | 2.5836E+02 | **2.0000E+02** | **2.0000E+02** | **2.0000E+02** |
| | (9.2705E+00) | (2.0482E+01) | (6.8524E+00) | (1.5625E+01) | (5.8654E+00) | **(4.0892E-13)** | **(0.0000E+00)** | **(0.0000E+00)** |
| | − | − | − | − | − | = | = | |
| F26 | **1.9233E+02** | 1.9246E+02 | 2.0009E+02 | 2.0016E+02 | 2.0008E+02 | 2.0000E+02 | 2.0000E+02 | 2.0000E+02 |
| | **(2.7060E+01)** | (2.7104E+01) | (7.4145E−03) | (2.3015E−02) | (1.2627E−02) | (6.2672E−13) | (0.0000E+00) | (0.0000E+00) |
| | + | + | − | − | − | = | = | |
| F27 | 4.9295E+02 | 6.8084E+02 | 1.0447E+03 | 1.8135E+03 | 9.5033E+02 | 3.7711E+02 | **2.0000E+02** | **2.0000E+02** |
| | (1.5533E+02) | (8.2246E+01) | (1.1466E+02) | (1.4439E+02) | (8.8368E+01) | (3.2793E+01) | **(0.0000E+00)** | **(0.0000E+00)** |
| | − | − | − | − | − | − | = | |
| F28 | 2.1182E+03 | 2.6173E+03 | 2.3307E+03 | 3.1541E+03 | 2.3170E+03 | 2.3086E+03 | **2.0000E+02** | **2.0000E+02** |
| | (1.4790E+02) | (3.9633E+02) | (2.7172E+02) | (3.4967E+02) | (2.5071E+02) | (4.6454E+01) | **(0.0000E+00)** | **(0.0000E+00)** |
| | − | − | − | − | − | − | = | |
| F29 | 1.4094E+03 | 3.2438E+03 | 1.3375E+03 | 2.2867E+03 | 1.3451E+03 | 7.9838E+02 | 2.4228E+02 | **2.0000E+02** |
| | (2.2823E+02) | (3.8777E+02) | (1.9888E+02) | (3.2957E+02) | (1.4872E+02) | (7.5828E+01) | (3.0195E+02) | **(0.0000E+00)** |
| | − | − | − | − | − | − | − | |
| F30 | **7.4961E+03** | 6.8645E+03 | 8.7953E+03 | 1.1546E+04 | 8.4953E+03 | 8.3501E+03 | 1.1499E+04 | 8.2952E+03 |
| | **(8.8393E+02)** | (9.5410E+02) | (1.4283E+03) | (3.2443E+03) | (9.2680E+02) | (9.6337E+02) | (9.3061E+03) | (7.7419E+02) |
| | + | + | − | − | − | − | − | |
| *w*/*t*/*l* | 7/4/19 | 2/3/25 | 4/3/23 | 2/1/27 | 7/5/18 | 4/13/13 | 2/8/20 | |

**Table 6**
Mean and standard deviation of the error values for functions f1-f30 @ 100D. Best entries are marked in boldface.

| | EPSDE | CoDE | CoBiDE | UMOEAs | AMALGAM | *d*DSF-EA |
|---|---|---|---|---|---|---|
| F1 | 3.5283E+05 | 7.8408E+05 | 9.8393E+05 | **0.0000E+00** | 6.9266E-02 | 1.5481E+05 |
| | (1.7736E+05) | (2.3046E+05) | (2.4263E+05) | **(0.0000E+00)** | (4.3500E-02) | (4.4904E+04) |
| | − | − | − | + | + | |
| F2 | 2.1917E+03 | 1.2940E+04 | 1.8623E+04 | **0.0000E+00** | 3.1652E+04 | **0.0000E+00** |
| | (6.9317E+03) | (1.1735E+04) | (1.7428E+04) | **(0.0000E+00)** | (4.0997E+04) | **(0.0000E+00)** |
| | − | − | − | = | − | |
| F3 | 2.0477E−03 | 4.8685E+02 | 1.1202E+00 | 1.1807E+01 | 1.5927E−03 | **0.0000E+00** |
| | (4.2699E−03) | (5.4667E+02) | (9.2619E−01) | (3.2061E+01) | (5.0506E−04) | **(0.0000E+00)** |
| | − | − | − | − | − | |
| F4 | 1.3580E+02 | 1.5142E+02 | 1.6769E+02 | 1.9503E+02 | **1.3985E+02** | 1.6063E+02 |
| | (4.2348E+01) | (3.3456E+01) | (4.0238E+01) | (2.7193E+01) | **(6.6484E+01)** | (2.6181E+01) |
| | + | + | − | − | + | |
| F5 | 2.1069E+01 | 2.0028E+01 | 2.0081E+01 | **2.0002E+01** | 2.0012E+01 | 2.0564E+01 |
| | (5.4879E−02) | (5.4426E−02) | (2.7763E−01) | **(8.2903E−03)** | (1.2485E−02) | (3.6034E−02) |
| | = | = | = | = | = | |
| F6 | 1.2968E+02 | 4.4880E+01 | 2.6775E+01 | **7.9300E+00** | 2.3796E+01 | 9.6903E+00 |
| | (4.3802E+00) | (6.5725E+00) | (7.1179E+00) | **(3.4878E+00)** | (5.4931E+00) | (2.7112E+00) |
| | − | − | − | + | − | |
| F7 | 8.1619E−03 | 9.6577E−04 | **0.0000E+00** | **0.0000E+00** | 5.7385E−03 | 1.4502E−04 |
| | (1.7601E−02) | (3.5870E−03) | **(0.0000E+00)** | **(0.0000E+00)** | (2.0378E−03) | (1.0357E−03) |
| | − | = | + | + | = | |
| F8 | 9.6584E+01 | 1.6212E+01 | 3.4396E+00 | 2.1635E+01 | 1.1004E+02 | **4.1692E−03** |
| | (4.7485E+01) | (4.3565E+00) | (6.7426E+00) | (4.8899E+00) | (1.9434E+01) | **(3.6511E−03)** |
| | − | − | − | − | − | |
| F9 | 6.4329E+02 | 2.1084E+02 | 2.1196E+02 | 6.0413E+01 | 2.0848E+02 | **3.8309E+01** |
| | (4.0261E+01) | (3.3882E+01) | (3.5375E+01) | (1.1975E+01) | (2.6494E+01) | **(6.2988E+00)** |
| | − | − | − | − | − | |
| F10 | 1.0356E+04 | 8.4659E+01 | 7.1638E+02 | 1.6645E+03 | 4.3857E+03 | **2.5243E+01** |
| | (1.7444E+03) | (8.9816E+01) | (1.7306E+02) | (1.6473E+03) | (1.5402E+03) | **(6.1027E+00)** |
| | − | − | − | − | − | |
| F11 | 2.6528E+04 | 1.2002E+04 | 1.1516E+04 | **1.0064E+04** | 1.0112E+04 | 1.0963E+04 |
| | (1.0668E+03) | (1.3890E+03) | (1.2288E+03) | **(2.0414E+03)** | (1.1047E+03) | (5.7809E+02) |
| | − | − | − | + | + | |
| F12 | 2.0291E+00 | 2.8319E−01 | 1.4308E−01 | **8.1971E−04** | 2.4501E−02 | 4.5112E−01 |
| | (3.2436E−01) | (1.8429E−01) | (7.1062E−02) | **(4.3929E−04)** | (4.2577E−03) | (4.6843E−02) |
| | − | + | + | + | + | |
| F13 | 4.7972E−01 | 4.2927E−01 | 3.8634E−01 | **2.1885E−01** | 4.3276E−01 | 2.3687E−01 |
| | (5.8966E−02) | (5.7280E−02) | (4.0985E−02) | **(3.5681E−02)** | (7.3589E−02) | (2.0001E−02) |
| | − | − | − | = | − | |

(*continued on next page*)

**Table 6** (*continued*)

| | EPSDE | CoDE | CoBiDE | UMOEAs | AMALGAM | *d*DSF-EA |
|---|---|---|---|---|---|---|
| F14 | 1.3908E+04 (9.3252E+03) − | **1.2626E−01 (1.4810E−02)** + | 2.8290E−01 (2.2862E−02) = | 1.3959E−01 (1.3247E−02) + | 5.2100E−01 (3.0672E−01) − | 3.3490E−01 (1.5611E−02) |
| F15 | 9.0087E+01 (1.3536E+01) − | 2.3732E+01 (4.5910E+00) − | 1.6840E+01 (2.7218E+00) = | **1.1855E+01 (1.3898E+00)** + | 1.2823E+01 (1.7087E+00) + | 1.6573E+01 (1.2374E+00) |
| F16 | 4.5921E+01 (4.3113E−01) − | 4.2189E+01 (1.1668E+00) − | 4.1725E+01 (1.1105E+00) − | 4.2364E+01 (7.2103E−01) − | 4.3815E+01 (9.6319E−01) − | **3.9258E+01 (5.2096E−01)** |
| F17 | 2.5556E+06 (4.4312E+06) − | 1.2409E+05 (4.0440E+04) − | 1.6151E+05 (5.8385E+04) − | 5.6608E+03 (3.3516E+03) − | 5.4718E+03 (8.0722E+02) − | **4.4332E+03 (6.7563E+02)** |
| F18 | 2.9169E+03 (3.4706E+03) − | 9.9298E+02 (1.0156E+03) − | 1.2933E+03 (1.3824E+03) − | 3.6067E+02 (1.0087E+02) − | 5.1931E+02 (1.3373E+02) − | **2.2252E+02 (1.8799E+01)** |
| F19 | **5.2023E+01 (2.0511E+01)** + | 9.2009E+01 (1.7679E+01) + | **9.0398E+01 (1.4401E+01)** + | 9.2573E+01 (2.1159E+01) = | 1.1267E+02 (2.9695E+00) − | 9.6531E+01 (2.4790E+00) |
| F20 | 1.2954E+03 (2.6087E+03) − | 1.9187E+03 (9.4802E+02) − | 2.3555E+02 (6.5119E+01) − | 3.9241E+02 (1.4710E+02) − | 7.2751E+02 (1.5524E+02) − | **1.3970E+02 (5.0419E+01)** |
| F21 | 5.2754E+05 (8.8753E+05) − | 5.8445E+04 (2.5505E+04) − | 9.4917E+04 (4.5601E+04) − | 4.9535E+03 (3.4595E+03) − | 3.2955E+03 (5.5289E+02) − | **2.2415E+03 (5.2845E+02)** |
| F22 | 2.1206E+03 (3.2906E+02) − | 1.8394E+03 (3.9059E+02) − | 1.6458E+03 (3.5314E+02) − | **7.0711E+02 (3.5708E+02)** + | 1.1670E+03 (3.5096E+02) − | 1.0542E+03 (1.9937E+02) |
| F23 | 3.4504E+02 (4.8172E−10) − | 3.4823E+02 (3.3301E−13) − | 3.4824E+02 (1.2933E−12) − | 3.4823E+02 (7.7310E−12) − | 3.4828E+02 (2.0800E−02) − | **2.0000E+02 (0.0000E+00)** |
| F24 | 4.0616E+02 (8.3431E+00) − | 3.8531E+02 (4.7593E+00) − | 3.6903E+02 (5.1155E+00) − | 3.8417E+02 (4.8468E+00) − | 3.7444E+02 (7.0067E+00) − | **2.0000E+02 (0.0000E+00)** |
| F25 | 2.6567E+02 (2.6963E+01) − | 2.5231E+02 (1.8111E+01) − | 2.1173E+02 (2.2893E+01) − | 2.2991E+02 (8.1664E+00) − | 2.0295E+02 (4.8880E−01) − | **2.0000E+02 (0.0000E+00)** |
| F26 | **1.3373E+02 (4.7403E+01)** + | 2.0011E+02 (1.8248E−02) - | 2.0011E+02 (1.4368E−02) = | 1.7855E+02 (4.1451E+01) + | 1.9871E+02 (1.4030E+01) + | 2.0000E+02 (0.0000E+00) |
| F27 | 3.7424E+03 (8.0297E+01) − | 1.1731E+03 (1.3449E+02) − | 7.3664E+02 (1.0165E+02) − | 4.4794E+02 (6.8288E+01) − | 9.3889E+02 (1.4165E+02) − | **2.0000E+02 (0.0000E+00)** |
| F28 | 8.3725E+02 (1.9107E+02) − | 2.3304E+03 (2.4505E+02) − | 1.9500E+03 (3.0927E+02) − | 2.3447E+03 (1.3583E+02) − | 2.8183E+03 (4.9983E+02) − | **2.0000E+02 (0.0000E+00)** |
| F29 | 2.7066E+02 (3.8804E+01) − | 1.5277E+03 (1.7129E+02) − | 2.1319E+03 (2.4346E+02) − | 1.4776E+03 (2.4731E+02) − | 9.6632E+02 (1.0596E+02) − | **2.0000E+02 (0.0000E+00)** |
| F30 | **2.6387E+03 (4.3839E+02)** + | 8.6182E+03 (9.8207E+02) − | 9.4799E+03 (1.2831E+03) − | 7.6225E+03 (1.1389E+03) + | 1.0557E+04 (1.5913E+03) − | 8.2952E+03 (7.7419E+02) |
| *w*/*t*/*l* | 4/1/25 | 4/2/24 | 3/4/23 | 10/4/16 | 6/2/22 | |

tions ($f_2$, $f_9$, $f_{15}$, $f_{16}$, $f_{17}$, $f_{18}$, $f_{26}$, and $f_{29}$) from the different categories of problems in the benchmark suite, are presented in Figs. 6 and 7 as box plots of performance data. The box plots represent the final performance of each of the top performing algorithms at 50*D*. Insets next to the original ones are provided for some of the algorithms to highlight the difference in results between these algorithms. Results in these figures illustrate the differences in terms of the different tallies of the performances of compared algorithms, which reflect the effectiveness and reliability of the proposed algorithm.

To better analyze the behavior of our method with respect to other set of algorithms, the contrast estimation is used which is a nonparametric procedure for estimating the contrast between algorithms medians [19]. This procedure is considered a safer metric in multiple-problem environments [24]. Tables 8 and 9, show the contrast estimation among all compared algorithms on 50*D* and 100*D*, respectively. As can be seen in these tables, *s*TDE-*d*R algorithm always obtains a positive difference value with respect to the other 12 algorithms. This indicates that the *s*TDE-*d*R is the best performing algorithm. Referring to the Holm, Holland, Rom, Finner and Li tests in Tables 10 and 11 with a $p$-value $\leq 0.05$, there is a statistically significant difference between the proposed work and each of the contestant algorithms.

**Table 7**
Average ranking of competitor algorithms, achieved by the Aligned Friedman test at dimensions $D$=50 and $D$=100.

| Algorithm | Ranking ($D$=50) | Ranking ($D$=100) |
|---|---|---|
| CoDE | 178.5500 | 203.6833 |
| EPSDE | 246.9167 | 244.4667 |
| JADE | 216.4667 | 201.2667 |
| SaDE | 308.7833 | 258.2333 |
| SHADE | 159.9500 | 165.7333 |
| L-SHADE | 151.4667 | 150.4833 |
| SFS | 206.2833 | 207.7333 |
| OXDE | 222.4167 | 291.4833 |
| CoBiDE | 181.0500 | 182.7667 |
| AMALGAM | 220.1167 | 188.6667 |
| UMOEAs | 161.9333 | 157.9333 |
| dynNP-jDE | 167.8500 | 163.1667 |
| *d*DSF-EA | 119.7167 | 125.8833 |
| Statistic | 26.0537 | 26.0190 |
| *p*-value | 0.01055 | 0.01067 |

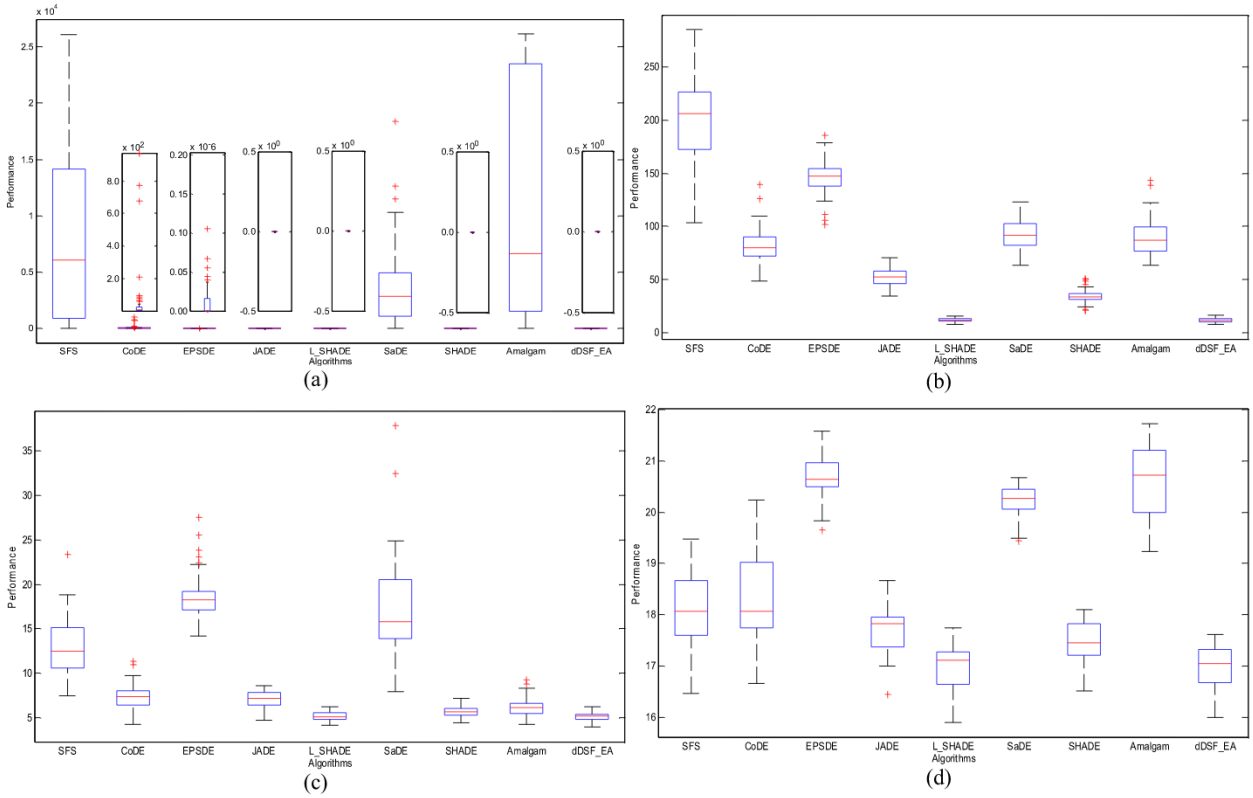


**Fig. 6.** Final performance results of the top performing algorithms for functions (a) $f_2$, (b) $f_9$, (c) $f_{15}$, (d) $f_{16}$ at 50$D$. Results are shown as box plots of performance data grouped by the contestant algorithms. Insets next to the results of any are provided for some of the algorithms to highlight the difference in results between these algorithms at their search scope, when there is a difference in scale of the obtained results.

## 5. Conclusions

In this paper, a decremental differential stochastic fractal evolutionary algorithm (*d*DSF-EA) is proposed to tackle complex optimization problems with balancing the exploration/exploitation features of the search. The proposed work presents a three-stage optimization algorithm: Differential Evolution Diffusion process, Success-based update process and Population size dynamic reduction. The L-SHADE algorithm is adopted in the first stage and is used as the Diffusion process of the Stochastic Fractal search instead of random fractals. The use of random fractals with fixed values for the parameters will slow down convergence especially when there is no knowledge propagation at different stages of the search. This problem

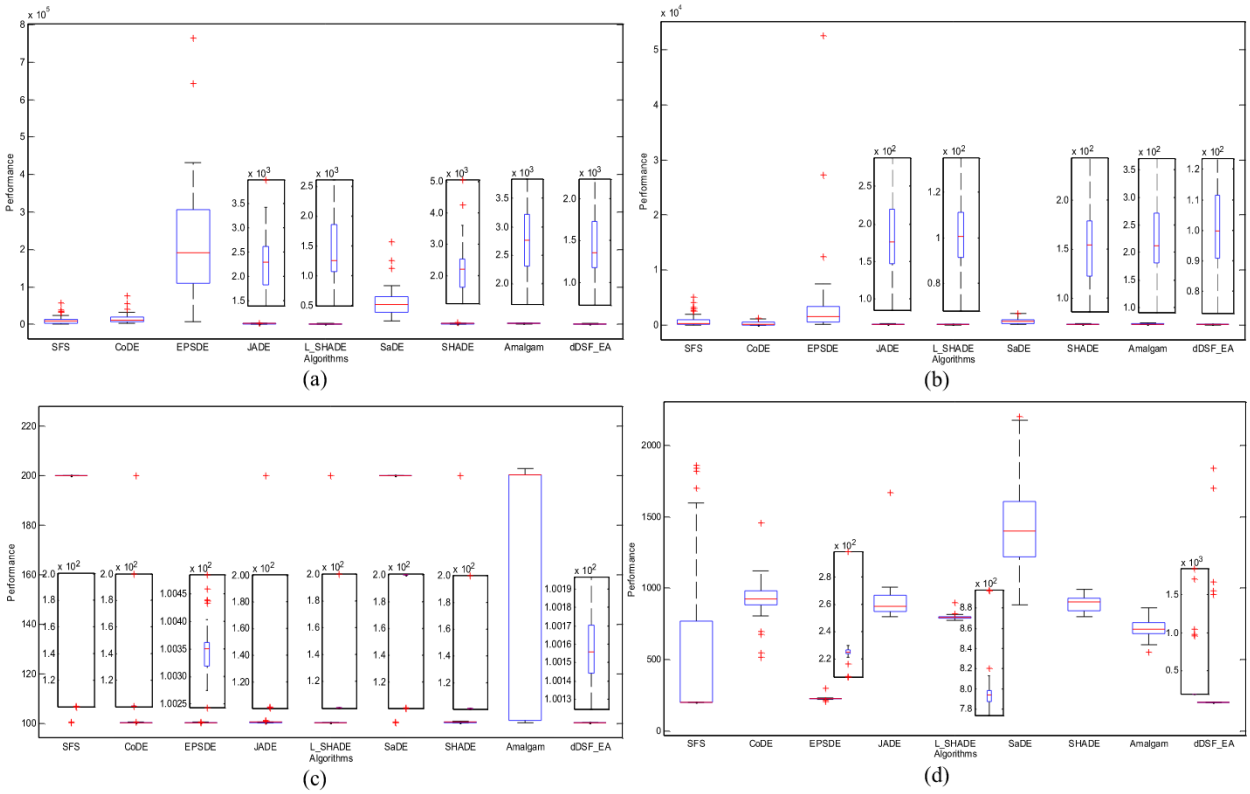


**Fig. 7.** Final performance results of the top performing algorithms for functions (a) $f_{17}$, (b) $f_{18}$, (c) $f_{26}$, (d) $f_{29}$ at 50$D$. Results are shown as box plots of performance data grouped by the contestant algorithms. Insets next to the results of any are provided for some of the algorithms to highlight the difference in results between these algorithms at their search scope, when there is a difference in scale of the obtained results.

**Table 8**
The contrast estimation among all algorithms on 50$D$.

| | CoDE | EPSDE | JADE | SaDE | SHADE | L-SHADE | SFS | OXDE | CoBiDE | Amalgam | UMOEAs | dynNP-jDE | $d$DSF-EA |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CoDE | 0.00 | −1.47 | −1.23 | 18.60 | −10.94 | −12.65 | 0.55 | 5.19 | −6.33 | 4.90 | −9.21 | −5.70 | −29.54 |
| EPSDE | 1.47 | 0.00 | 0.24 | 20.07 | −9.47 | −11.18 | 2.02 | 6.66 | −4.87 | 6.37 | −7.74 | −4.23 | −28.08 |
| JADE | 1.23 | −0.24 | 0.00 | 19.83 | −9.71 | −11.42 | 1.78 | 6.42 | −5.11 | 6.13 | −7.98 | −4.47 | −28.32 |
| SaDE | −18.60 | −20.07 | −19.83 | 0.00 | −29.54 | −31.25 | −18.05 | −13.41 | −24.93 | −13.70 | −27.81 | −24.30 | −48.14 |
| SHADE | 10.94 | 9.47 | 9.71 | 29.54 | 0.00 | −1.71 | 11.49 | 16.13 | 4.61 | 15.84 | 1.73 | 5.24 | −18.61 |
| L-SHADE | 12.65 | 11.18 | 11.42 | 31.25 | 1.71 | 0.00 | 13.20 | 17.84 | 6.32 | 17.55 | 3.44 | 6.95 | −16.90 |
| SFS | −0.55 | −2.02 | −1.78 | 18.05 | −11.49 | −13.20 | 0.00 | 4.64 | −6.88 | 4.35 | −9.76 | −6.25 | −30.09 |
| OXDE | −5.19 | −6.66 | −6.42 | 13.41 | −16.13 | −17.84 | −4.64 | 0.00 | −11.52 | −0.29 | −14.39 | −10.89 | −34.73 |
| CoBiDE | 6.33 | 4.87 | 5.11 | 24.93 | −4.61 | −6.32 | 6.88 | 11.52 | 0.00 | 11.24 | −2.87 | 0.63 | −23.21 |
| Amalgam | −4.90 | −6.37 | −6.13 | 13.70 | −15.84 | −17.55 | −4.35 | 0.29 | −11.24 | 0.00 | −14.11 | −10.60 | −34.45 |
| UMOEAs | 9.21 | 7.74 | 7.98 | 27.81 | −1.73 | −3.44 | 9.76 | 14.39 | 2.87 | 14.11 | 0.00 | 3.51 | −20.34 |
| dynNP-jDE | 5.70 | 4.23 | 4.47 | 24.30 | −5.24 | −6.95 | 6.25 | 10.89 | −0.63 | 10.60 | −3.51 | 0.00 | −23.85 |
| $d$DSF-EA | 29.54 | 28.08 | 28.32 | 48.14 | 18.61 | 16.90 | 30.09 | 34.73 | 23.21 | 34.45 | 20.34 | 23.85 | 0.00 |

was curtailed by utilizing the L-SHADE algorithm in the Diffusion process where it uses an adaptive manner of fine-tuning the parameters that can benefit from the best values of generating well-performing individuals. The second stage of the proposed work presents an adaptive success-based mechanism to call the update processes in this new amalgam. The use of success-based scheme is proposed to solve this issue. The proposed success-based scheme will assign the suitable number of function evolutions to each update process based on its previous success and will also utilize the allowed function evaluations in an effective manner which reflects the quality of new individuals. The performance of the proposed algorithm is tested on a challenging set of 30 benchmark problems taken from the IEEE CEC 2014 single objective optimization special session and competition. Simulation results affirm the fact that the new hybrid algorithm significantly advances the efficiency over any of its SFS or L-SHADE component algorithms, and other state-of-the-art algorithms in terms of quality of found solutions. Future works will include testing other types of individual success-based parameter adaptation and stagnation alerts to flag a need for changing the course of the search.

**Table 9**
The contrast estimation among all algorithms on 100*D*.

| | CoDE | EPSDE | JADE | SaDE | SHADE | L-SHADE | SFS | OXDE | CoBiDE | Amalgam | UMOEAs | dynNP-jDE | *d*DSF-EA |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| CoDE | 0.00 | 11.64 | −14.96 | 23.88 | −39.99 | −45.97 | −0.81 | 44.58 | −16.81 | −18.96 | −39.84 | −31.60 | −95.17 |
| EPSDE | −11.64 | 0.00 | −26.59 | 12.24 | −51.62 | −57.61 | −12.44 | 32.95 | −28.44 | −30.60 | −51.47 | −43.23 | −106.80 |
| JADE | 14.96 | 26.59 | 0.00 | 38.83 | −25.03 | −31.01 | 14.15 | 59.54 | −1.85 | −4.00 | −24.88 | −16.64 | −80.21 |
| SaDE | −23.88 | −12.24 | −38.83 | 0.00 | −63.86 | −69.85 | −24.68 | 20.71 | −40.68 | −42.84 | −63.71 | −55.47 | −119.00 |
| SHADE | 39.99 | 51.62 | 25.03 | 63.86 | 0.00 | −5.99 | 39.18 | 84.57 | 23.18 | 21.03 | 0.15 | 8.39 | −55.18 |
| L-SHADE | 45.97 | 57.61 | 31.01 | 69.85 | 5.99 | 0.00 | 45.16 | 90.55 | 29.17 | 27.01 | 6.13 | 14.38 | −49.20 |
| SFS | 0.81 | 12.44 | −14.15 | 24.68 | −39.18 | −45.16 | 0.00 | 45.39 | −16.00 | −18.15 | −39.03 | −30.79 | −94.36 |
| OXDE | −44.58 | −32.95 | −59.54 | −20.71 | −84.57 | −90.55 | −45.39 | 0.00 | −61.39 | −63.54 | −84.42 | −76.18 | −139.80 |
| CoBiDE | 16.81 | 28.44 | 1.85 | 40.68 | −23.18 | −29.17 | 16.00 | 61.39 | 0.00 | −2.16 | −23.03 | −14.79 | −78.36 |
| Amalgam | 18.96 | 30.60 | 4.00 | 42.84 | −21.03 | −27.01 | 18.15 | 63.54 | 2.16 | 0.00 | −20.88 | −12.64 | −76.21 |
| UMOEAs | 39.84 | 51.47 | 24.88 | 63.71 | −0.15 | −6.13 | 39.03 | 84.42 | 23.03 | 20.88 | 0.00 | 8.24 | −55.33 |
| dynNP-jDE | 31.60 | 43.23 | 16.64 | 55.47 | −8.39 | −14.38 | 30.79 | 76.18 | 14.79 | 12.64 | −8.24 | 0.00 | −63.57 |
| *d*DSF-EA | 95.17 | 106.80 | 80.21 | 119.00 | 55.18 | 49.20 | 94.36 | 139.80 | 78.36 | 76.21 | 55.33 | 63.57 | 0.00 |

**Table 10**
A comparison of adjusted *p*-values when $D=50$ with $\alpha = 0.05$ (Control method: *s*TDE-*d*R).

| i | hypothesis | $p_{Holm}$ | $P_{Holland}$ | $P_{Rom}$ | $P_{Finner}$ | $P_{Li}$ |
|---|---|---|---|---|---|---|
| 1 | SaDE | 0.004167 | 0.004265 | 0.004383 | 0.004265 | 0.038140 |
| 2 | EPSDE | 0.004545 | 0.004652 | 0.004782 | 0.008512 | 0.038140 |
| 3 | OXDE | 0.005000 | 0.005116 | 0.005260 | 0.012741 | 0.038140 |
| 4 | Amalgam | 0.005556 | 0.005683 | 0.005844 | 0.016952 | 0.038140 |
| 5 | JADE | 0.006250 | 0.006391 | 0.006574 | 0.021145 | 0.038140 |
| 6 | SFS | 0.007143 | 0.007301 | 0.007513 | 0.025321 | 0.038140 |
| 7 | CoBiDE | 0.008333 | 0.008512 | 0.008764 | 0.029478 | 0.038140 |
| 8 | CoDE | 0.010000 | 0.010206 | 0.010515 | 0.033617 | 0.038140 |
| 9 | dynNP-jDE | 0.012500 | 0.012741 | 0.013109 | 0.037739 | 0.038140 |
| 10 | UMOEAs | 0.016667 | 0.016952 | 0.016667 | 0.041844 | 0.038140 |
| 11 | SHADE | 0.025000 | 0.025321 | 0.025000 | 0.042425 | 0.038140 |
| 12 | L-SHADE | 0.031140 | 0.035247 | 0.035124 | 0.045931 | 0.038140 |

**Table 11**
A comparison of adjusted *p*-values when $D=100$ with $\alpha = 0.05$ (Control method: *s*TDE-*d*R).

| i | hypothesis | $p_{Holm}$ | $P_{Holland}$ | $P_{Rom}$ | $P_{Finner}$ | $P_{Li}$ |
|---|---|---|---|---|---|---|
| 1 | SaDE | 0.004167 | 0.004265 | 0.004383 | 0.004265 | 0.038140 |
| 2 | EPSDE | 0.004545 | 0.004652 | 0.004782 | 0.008512 | 0.038140 |
| 3 | OXDE | 0.005000 | 0.005116 | 0.005260 | 0.012741 | 0.038140 |
| 4 | Amalgam | 0.005556 | 0.005683 | 0.005844 | 0.016952 | 0.038140 |
| 5 | JADE | 0.006250 | 0.006391 | 0.006574 | 0.021145 | 0.038140 |
| 6 | SFS | 0.007143 | 0.007301 | 0.007513 | 0.025321 | 0.038140 |
| 7 | CoBiDE | 0.008333 | 0.008512 | 0.008764 | 0.029478 | 0.038140 |
| 8 | CoDE | 0.010000 | 0.010206 | 0.010515 | 0.033617 | 0.038140 |
| 9 | dynNP-jDE | 0.012500 | 0.012741 | 0.013109 | 0.037739 | 0.038140 |
| 10 | UMOEAs | 0.016667 | 0.016952 | 0.016667 | 0.041844 | 0.038140 |
| 11 | SHADE | 0.025000 | 0.025321 | 0.025000 | 0.042425 | 0.038140 |
| 12 | L-SHADE | 0.031140 | 0.035247 | 0.035124 | 0.045931 | 0.038140 |

## References


[1] M.Z. Ali, K. Alkhatib, Y. Tashtoush, Cultural algorithms: emerging social structures for the solution of complex optimization problems, Int. J. Artif. Intel., IJAI 11 (2013) 20–43.
[2] M.Z. Ali, N.H. Awad, R. Duwairi, J. Albadarneh, R.G. Reynolds, P.N. Suganthan, Cluster-based differential evolution with heterogeneous influence for numerical optimization, in: Proceedings of the IEEE Congress of Evolutionary Computation, 2015, pp. 393–400.
[3] M.Z. Ali, N.H. Awad, R.G. Reynolds, Hybrid niche cultural algorithm for numerical global optimization, in: Proceedings of the IEEE Congress of Evolutionary Computations, Cancun, Mexico, June 2013, pp. 309–316.
[4] M.Z. Ali, N.H. Awad, P.N. Suganthan, Multi-population differential evolution with balanced ensemble of mutation strategies for large-scale global optimization, Appl. Soft Comput. 33 (2015) 304–327.
[5] N.H. Awad, M.Z. Ali, R.M. Duwairi, Cultural algorithm with improved local search for optimization problems, in: Proceedings of the IEEE Congress of Evolutionary Computation, Cancun, Mexico, June 2013, pp. 284–291.
[6] N.H. Awad, M.Z. Ali, R.G. Reynolds, A differential evolution algorithm with success-based parameter adaptation for CEC2015 learning-based optimization, in: Proceedings of the IEEE Congress of Evolutionary Computation, 2015, pp. 1098–1105.
[7] R.L. Becerra, C.A.C. Coello, Culturizing differential evolution for constrained optimization, in: Proceedings of the Mexican International Conference on Computer Science, 2004, pp. 304–311.
[8] R.L. Becerra, C.A.C. Coello, Cultured differential evolution for constrained optimization, Comput. Methods Appl. Mech. Eng. 195 (2006) 4303–4322.

[9] J. Brest, S. Greiner, B. Boskovic, M. Mernik, V. Zumer, Self adapting control parameters in differential evolution: a comparative study on numerical benchmark problems, IEEE Trans. Evol. Comput. 10 (2006) 646–657.
[10] J. Brest, M.S. Maucec, Population size reduction for the differential evolution algorithm, Appl. Intell. 29 (2008) 228–247.
[11] Y. Cai, J. Wang, Differential evolution with neighborhood and direction information for numerical optimization, IEEE Trans. Cybern. 43 (2013) 2202–2215.
[12] C.J. Chung, R.G. Reynolds, Function optimization using evolutionary programming with self-adaptive cultural algorithms, in: Proceedings of the Asia-Pacific Conference on Simulated Evolution and Learning, 1996, pp. 17–26.
[13] C.J. Chung, R.G. Reynolds, A testbed for solving optimization problems using cultural algorithms, in: Proceedings of the Annual Conference on Evolution Programming, 1996, pp. 225–236.
[14] L.S. Coelho, V.C. Mariani, An efficient particle swarm optimization approach based on cultural algorithm applied to mechanical design, in: Proceedings of the IEEE Congress on Evolutionary Computing, Canada, 2006, pp. 1099–1104.
[15] S. Das, S.S. Mullick, P.N. Suganthan, Recent advances in differential evolution – An updated survey, Swarm Evol. Comput. 27 (2016) 1–30.
[16] K. Deb, P.N. Suganthan, Special Session on Real Parameter Optimization at CEC 05, Retrieved from: www.ntu.edu.sg/home/EPNSugan/, April 25, 2005.
[17] J. Demsar, Statistical comparisons of classifiers over multiple data sets, J. Mach. Learn. Res. 7 (2006) 1–30.
[18] J. Derrac, S. Garcia, D. Molina, F. Herrera, A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms, Swarm Evol. Comput. 1 (2011) 3–18.
[19] K. Doksum, Robust procedures for some linear models with one observation per cell, Ann. Math. Stat. 38 (1967) 878–883.
[20] S.M. Elsayed, R.A. Sarker, D.L. Essam, N.M. Hamza, Testing united multi-operator evolutionary algorithms on the CEC2014 real-parameter numerical optimization, in: Proceedings of the IEEE Congress on Evolutionary Computing, 2014, pp. 1650–1657.
[21] R. Gamperle, S.D. Muller, P. Koumoutsakos, A parameter study for differential evolution, in: Proceedings of the Advances Intelligent Systems, Fuzzy Systems,Evolutionary Computing, Crete, Greece, 2002, pp. 293–298.
[22] X.Z. Gao, X. Wang, T. Jokinen, S.J. Ovaska, A. Arkkio, K. Zenger, A hybrid optimization method for wind generator design, Int. J. Innov. Comput. 8 (2012) 4347–4373.
[23] J.M. Garcia-Nieto, E. Alba, Hybrid PSO6 for hard continuous optimization, Soft Comput. 19 (2015) 1843–1861.
[24] S. Garcia, A. Fernandez, J Luengo, F. Herrera, Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: experimental analysis of power, Inf. Sci. 180 (2010) 2044–2064.
[25] W. Gong, Z. Cai, Differential evolution with ranking-based mutation operators, IEEE Trans. Cybern. (43) (2013) 2066–2081.
[26] S.M. Guo, J.S.H. Tsai, C.C. Yang, P.H. Hsu, A self-optimization approach for L-SHADE incorporated with eigenvector-based crossover and successful-parent-selecting framework on CEC 2015 benchmark set, in: Proceedings of the IEEE Congress on Evolutionary Computing, 2015, pp. 1003–1010.
[27] J.H. Holland, Adaption in Natural and Artificial Systems, University of Michigan Press, Ann Arbor, MI, 1975.
[28] R. Joshi, A.C. Sanderson, Minimal representation multisensor fusion using differential evolution, IEEE Trans. Syst., Man Cybern. Part A 29 (1999) 63–76.
[29] Z. Kobti, R.G. Reynolds, T. Kohler, A multi-agent simulation using cultural algorithms: the effect of culture on the resilience of social systems, in: Proceedings of the IEEE Congress on Evolutionary Computing, 2003, pp. 1988–1995.
[30] J.L.J. Laredo, C. Fernandes, J.J.M. Guervos, C. Gagne, Improving genetic algorithms performance via deterministic population shrinkage, in: Proceedings of the GECCO, 2009, pp. 819–826.
[31] J.J. Liang, B.-Y. Qu, P.N. Suganthan, Problem Definitions and Evaluation Criteria for the CEC 2014 Special Session and Competition on Single Objective Real-Parameter Numerical Optimization, Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou, China and Technical Report, Nanyang Technological University, Singapore, December 2013 Technical Report 201311.
[32] R. Mallipeddi, P.N. Suganthan, Q.K. Pan, M.F. Tasgetiren, Differential evolution algorithm with ensemble of parameters and mutation strategies, Appl. Soft Comput. 11 (2011) 1679–1696.
[33] T.T. Nguyen, X. Yao, An experimental study of hybridizing cultural algorithms and local search, Int. J. Neural Systems 18 (2008) 1–18.
[34] K.V. Price, R.M. Storn, J.A. Lampinen, Differential Evolution: A Practical Approach to Global Optimization, 1st ed., Springer-Verlag, New York, Dec. 2005.
[35] A.K. Qin, V.L. Huang, P.N. Suganthan, Differential evolution algorithm with strategy adaptation for global numerical optimization, IEEE Trans. Evol. Comput. 13 (2009) 398–417.
[36] A.K. Qin, P.N. Suganthan, Self-adaptive differential evolution algorithm for numerical optimization, in: Proceedings of the IEEE Congress on Evolutionary Computing, vol. 2, Sep. 2005, pp. 1785–1791.
[37] H. Salimi, Stochastic fractal search: a powerful metaheuristic algorithm, Knowl. Based Syst. 75 (2015) 1–18.
[38] R. Storn, K. Price, Differential evolution a simple and efficient heuristic for global optimization over continuous spaces, J. Global Opt. 11 (1997) 341–359.
[39] R. Storn, K.V. Price, Differential evolution—A simple and efficient adaptive scheme for global optimization over continuous spaces, ICSI, Berkeley, CA, 1995 Tech. Rep. TR-95-012[Online]. Available: http://http.icsi.berkeley.edu/~storn/litera.html.
[40] R. Tanabe, A. Fukunaga, Success-history based parameter adaptation for differential evolution, in: Proceedings of the IEEE CEC, 2013, pp. 71–78.
[41] R. Tanabe, A. Fukunaga, Improving the search performance of SHADE using linear population size reduction, in: Proceedings of the IEEE CEC, 2014, pp. 1658–1665.
[42] J.A. Vrugt, B.A. Robinson, J.M. Hyman, Self-adaptive multimethod search for global optimization in real-parameter spaces, IEEE Trans. Evol. Comput. 13 (2009) 243–259.
[43] Y. Wang, Z. Cai, Q. Zhang, Differential evolution with composite trial vector generation strategies and control parameters, IEEE Tran. Evol. Comput. 15 (2011) 55–66.
[44] Y. Wang, Z. Cai, Q. Zhang, Enhancing the search ability of differential evolution through orthogonal crossover, Inf. Sci. 185 (2012) 153–177.
[45] Y. Wang, H.-X. Li, T. Huang, L. Li, Differential evolution based on covariance matrix learning andbimodal distribution parameter setting, Appl. Soft Comput. 18 (2014) 232–247.
[46] H. Wang, I. Moon, S. Yang, D. Wang, A memetic particle swarm optimization algorithm for multimodal optimization problems, Inf. Sci. 197 (2012) 38–52.
[47] H. Wang, S. Rahnamayan, H. Sun, M.G.H. Omran, Gaussian bare-bones differential evolution, IEEE Trans. Cybern. (43) (2013) 634–647.
[48] F. Wilcoxon, Individual comparisons by ranking methods, Biometrics 1 (1945) 80–83.
[49] A. Zamuda, J. Brest, Self-adaptive control parameters' randomization frequency and propagations in differential evolution, Swarm Evol. Comput. 25 (2015) 72–99.
[50] J. Zhang, V. Avasarala, A.C. Sanderson, T. Mullen, Differential evolution for discrete optimization: an experimental study on combinatorial auction problems, in: Proceedings of the IEEE World Congress on Computational Intelligence, Hong Kong, China, Jun. 2008, pp. 2794–2800.
[51] J. Zhang, V. Avasarala, R. Subbu, Evolutionary optimization of transition probability matrices for credit decision-making, Eur. J. Oper. Res. 200 (2010) 557–567.
[52] J. Zhang, A.C. Sanderson, JADE: adaptive differential evolution with optional external archive, IEEE Trans. Evol. Comput. 13 (2009) 945–958.
[53] J. Zhang, A.C. Sanderson, An approximate Gaussian model of differential evolution with spherical fitness functions, in: Proceedings of the IEEE Congress on Evolutionary Computing, Singapore, Sep. 2007, pp. 2220–2228.