Differential Evolution with Stochastic Fractal Search Algorithm for Global Numerical Optimization

Noor H. Awad[1], Mostafa Z. Ali[2], Ponnuthurai N. Suganthan[1], Edward Jaser[2]
[1]Nanyang Technological University, Singapore 639798
School of Electrical & Electronic Engineering
[2]Princess Sumayya University for Technology
King Hussein Faculty of Computing Sciences
Emails: noor0029@e.ntu.edu.sg, mzali.pn@gmail.com, epnsugan@ntu.edu.sg, ejaser@psut.edu.jo

***Abstract*— Evolutionary algorithms are successfully developed to handle the challenges in solving optimization problems with complex landscapes. Differential evolution proves its efficiency as a powerful evolutionary algorithm to solve complex optimization problems with diverse characteristics. In this paper, we aim at designing an enhanced evolutionary algorithm that embeds Differential Evolution in Stochastic Fractal Search. Stochastic Fractal Search is developed recently as a powerful metaheuristic algorithm that imitates the natural phenomenon of growth and uses the diffusion process based on random fractals. In this paper, we introduce a new adjustment to the Diffusion Process of Stochastic Fractal Search. The proposed algorithm namely, SFS-DP$_{DE-GW}$, uses Differential Evolution in the Diffusion Process along with the Gaussian Walks to enhance the search. To validate 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 proposed combination clearly enhances the performance of Stochastic Fractal Search and increases the efficiency of the update process which was incorporated after Diffusion process. Comparative studies show that the new algorithm has a superior performance compared to the original Stochastic Fractal Search and other recent state-of-the-art algorithms.**

Keywords*— *Evolutionary Algorithms, Optimization Problem, Differential Evolution, Stochastic Fractal Search

I. INTRODUCTION

Optimization algorithms are needed in countless applications of different areas of study and real-life problem solving. Mathematically, any optimization problem can be expressed as minimizing or maximizing an objective function of the form $f(\vec{X})(f:\Omega\subseteq\Re^{D}\rightarrow\Re)$ where Ω is the domain range of the problem being solved which involves D decision variables, represented as a vector like $\vec{X}=[x_1,x_2,x_3,...,x_D]^T$. Developing efficient evolutionary algorithms for solving such problems is essential and challenging task for solving optimization problems. Over the last few decades, evolutionary algorithms proved their effectiveness over classical methods such as Nelder-Mead's simplex method [1], Tabu search [2] and quasi-Newton [3]. Despite this fact, evolutionary algorithms often become trapped in stagnation scenarios and premature convergence especially when solving problems with complex landscapes. To solve this issue, a potential effort has been made to develop new efficient algorithms and/or hybridizing the strengths of multiple evolutionary algorithms or local search methods. This hybridization offers efficient synergistic style to develop new algorithms that are superior in terms of robustness and resiliency to their constituent algorithms [4, 5, 6, 7, 8].

Differential Evolution (DE) is one of the most effective evolutionary algorithms due to its simplicity and powerful search ability to solve a wide range of optimization problems with different characteristics [9, 10, 11]. It consists of three main steps to generate new individuals at each optimization loop which are: mutation, crossover and selection operator. Although the competitive performance of Differential Evolution has been demonstrated, many research works show that it is quite dependent on the settings of control parameters such as scaling factor (*F*), crossover rate (*CR*) and population size (*NP*) and may get trapped in local optimal solutions [12, 13]. As a result, many researchers have developed self-adaptive Differential Evolution algorithms and hybridized Differential Evolution with other algorithms or local search methods to increase its search ability.

One of the early works that adjust the control parameters and mutation strategies during the search process of Differential Evolution is SaDE algorithm [14]. This algorithm uses a learning period to adapt *F* and *CR* based on past search behavior. It uses a pool of strategies to select one of them randomly based on previous success times. This randomized assignment is biased with a higher number of successes of a strategy. The scaling factor and crossover rate are generated using the Gaussian distribution. Another algorithm that seeks to automate the control parameter settings of DE is jDE [15]. In this algorithm, different values are assigned to each individual at each optimization loop based on some heuristic rules. The uniform distribution is used to generate *F* and *CR* values and the new offspring with the best parameter values are most likely to survive and propagate to the next generation. 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 [16]. The CoDE algorithm also used a similar idea of using different mutation strategies and parameter settings [17]. Unlike the aforementioned algorithms which use one of the existing mutation strategies, JADE algorithm introduces a new effective mutation strategy called "current-to-pbest/1" with an

978-1-5090-0623-6/16/$31.00 ©2016 IEEE

optional external archive [18]. This mutation strategy is an extension to the basic current-to-best/1 but the best individual is chosen from the top best individuals in each generation. *F* and *CR* are adapted using the Cauchy and Normal distributions with successful mean values of previous generations. SHADE is an algorithm developed based on JADE mutation [19]. In this algorithm, a history-based scheme is used to enhance parameter settings of JADE. They used a random index within the range of used memory to select one of the successful mean values. SHADE algorithm is further enhanced by L-SHADE algorithm by incorporating a linear population size reduction scheme [20].

Recently, Stochastic Fractal Search (SFS) is introduced by Salimi as a new powerful metaheuristic algorithm [21]. This algorithm imitates the natural phenomenon of growth and uses two main processes which are: the diffusion process and the update process. The diffusion process uses random fractals using Gaussian Walks and Levy flights in which each particle is diffused around its current position. The diffusion process is considered the exploitation phase in Stochastic Fractal Search while the update process is the exploration phase. In this later phase, two update processes are called in an attempt to increase the search ability of random fractals. Although the Stochastic Fractal Search overcomes the shortcomings of other nature-inspired algorithms, a competitive performance is not guaranteed when solving complex optimization problems due to stagnation and premature convergence scenarios. In this paper, we aim at mitigating such issues and introduce an enhanced and powerful evolutionary algorithm that embeds Differential Evolution in Stochastic Fractal Search. Instead of just using the random fractals in the diffusion process that may suffer from low convergence, a Differential Evolution algorithm based on SHADE mutation is also used. The new adjustment of the diffusion process increases the search ability of Stochastic Fractal Search and balances the exploitation and exploration phases. The strength of the update process is also enhanced by providing fitter search agents to start the exploration phase.

The remaining sections complete this paper as follows. Section 2 briefly introduces Differential Evolution and Stochastic Fractal Search. In Section 3, the proposed method is elaborated. Section 4 describes the used benchmark problems, parameter settings and simulation results. Finally, section 5 summarizes the conclusion of this paper.

II. SCIENTIFIC BACKGROUND

A. Differential Evolution

Differential Evolution is a population-based evolutionary algorithm proposed by Storn [22]. It is considered one of the most promising evolutionary algorithms due to its simplicity with a powerful stochastic direct search technique [23]. The classical Differential Evolution consists of a population of *NP* individuals and each of them is represented as a vector of *D*-dimensional parameters as $X_{i,G} = \{x^1_{i,G}, ..., x^D_{i,G}\}, i = 1, ..., NP$ where *G* is the generation number and *NP* is the population size. The algorithm starts by initializing random individuals that are uniformly distributed within the search space of the problem being solved as follows:

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

where *j* is the index of parameter value in the i^{th} individual, and $rand(0,1)$ is a uniformly distributed random generator in the range [0,1], $X_{\min} = \{x^1_{\min}, ..., x^D_{\min}\}$ and $X_{\max} = \{x^1_{\max}, ..., x^D_{\max}\}$ are the lower and upper bounds of the search range. Five mutation strategies were suggested in the classical DE [22, 24] as described below:

"DE/rand/1": $V_i = X_{r_1} + F \cdot (X_{r_2} - X_{r_3})$ (2)

"DE/best/1": $V_i = X_{best} + F \cdot (X_{r_1} - X_{r_2})$ (3)

"DE/current-best/1":

$$V_i = X_i + F \cdot (X_{best} - X_i + X_{r_1} - X_{r_2}) \qquad (4)$$

"DE/best/2":

$$V_i = X_{best} + F \cdot (X_{r_1} - X_{r_2} + X_{r_3} - X_{r_4}) \qquad (5)$$

"DE/rand/2":

$$V_i = X_{r_1} + F \cdot (X_{r_2} - X_{r_3} + X_{r_4} - X_{r_5}) \qquad (6)$$

The $V_{i,G} = \{v^1_{i,G}, v^2_{i,G}, ..., v^D_{i,G}\}$ is the mutant vector which is generated against each individual $X_{i,G}$ in the population space. $r^i_1, r^i_2, r^i_3, r^i_4, r^i_5$ are distinct random integer numbers between 1 and *NP*.

After mutation is performed to generate new offspring, the crossover is applied to trial vector $U_{i,G} = \{u^1_{i,G}, u^2_{i,G}, ..., u^D_{i,G}\}$. The original DE has defined a binomial crossover as follows [22, 24]:

$$u^j_{i,G} = \begin{cases} v^j_{i,G}, if\ (with\ probability\ of\ CR)\ or\ (j = j_{rand}) \\ x^j_{i,G}, otherwise \end{cases} \quad j = 1, 2, ..., D \qquad (7)$$

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*. After crossover, the DE algorithm must check the generated trial vector to be within the search range. Otherwise, re-initialization takes place for the violated dimensions. Finally, the selection operation is started to fill the next generation population with new individuals as shown in Eq. 8.

$$X_{i,G+1} = \begin{cases} U_{i,G}, if\ f(U_{i,G}) \le f(X_{i,G}) \\ X_{i,G}, otherwise \end{cases} \qquad (8)$$

The above DE steps including mutation, crossover and selection will be repeated until a termination criterion is met.

B. Stochastic Fractal Search

Stochastic Fractal Search was introduced recently by Salimi as a new powerful metaheuristic [21]. The aim of this

algorithm is to overcome the shortcomings of basic Fractal Search. Firstly, the parameter setting of basic Fractal Search is time consuming. Besides that, an appropriate settings for too many parameters need to be fine-tuned to deduce a good performance. Secondly, there is no information exchange between particles. This affects the search ability to converge to promising regions and force the particles to perform their search independently. Stochastic Fractal Search tackles the mentioned shortcomings by performing two main processes: Diffusion process and Update process [21].

The Diffusion process is considered as the exploitation phase in which each particle is diffused around its current position to generate a new offspring. In this process, the intensification property is satisfied to prevent the scenario of getting trapped in a local optima and increases the chance of locating the global optimum. This Diffusion process is different from the basic one in Fractal Search due to: it does not cause a dramatic increase in the number of participating points to generate a new particle and it uses Gaussian Walks and Levy Flight to locate the global optimum faster. After initializing a set of points uniformly distributed within the search range, the Diffusion process is started to generate new particles using one of the two Gaussian Walks as shown in Eq. 9 and 10.

$$GW_1 = Gaussian(\mu_{BP}, \sigma) + (\varepsilon \times BP - \varepsilon' \times P_i) \quad (9)$$

$$GW_2 = Gaussian(\mu_P, \sigma) \quad (10)$$

where ε and ε' are two uniform random numbers in the range [0, 1], BP is the position of best point while P_i is the position of the *i*th point in the group. The two means μ_{BP} and μ_P are equal to BP and P_i respectively while standard deviation σ is computed as shown in Eq. 11 where t is the generation number.

$$\sigma = \left| \frac{\log(t)}{t} \times (P_i - BP) \right| \quad (11)$$

After that, the Update process is started which 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. This process consists of two update procedures. The first one uses Eq. 12 to update the j^{th} index of each point while second update procedure uses Eq. 13 to update the point.

$$P_i'(j) = P_r(j) - \varepsilon \times (P_t(j) - P_i(j)) \quad (12)$$

$$P_i' = \begin{cases} P_i - \varepsilon \times (P_t - BP), & \varepsilon \le 0.5 \\ P_i + \varepsilon \times (P_t - P_r), & \varepsilon > 0.5 \end{cases} \quad (13)$$

where P_i' is the new point, P_r and P_t are two different random points selected from the group, BP is the best point and ε is a random number within the range [0, 1].

III. PROPOSED ALGORITHM

Developing an efficient evolutionary algorithm is a challenging task especially when solving optimization problems which have different characteristics such as non-separability, ill-conditioning, multimodality, in addition to the presence of the curse of dimensionality. Although the Stochastic Fractal Search is claimed to rectify the shortcomings of basic Fractal Search, it may get trapped in premature convergence and stagnation scenarios when solving problems with the aforementioned characteristics. Moreover, the use of static parameter settings will not be a suitable choice for all the problems and hence degrade the performance and cause low convergence for the evolutionary search. Furthermore, there is no knowledge propagation at the different stages of the Diffusion process. The quality of solutions can be enhanced if the best knowledge of previous generations can be boosted to the next generation. This paper aims at solving these issues by introducing a new adjustment on Stochastic Fractal Search. The proposed algorithm uses a differential-Gaussian Diffusion process, as the following sub-sections present.

A. Differential-Gaussian Diffusion process

Instead of using Gaussian Walks in the original Stochastic Fractal Search, the Diffusion process is also carried out via Differential Evolution. Our algorithm used SHADE mutation [19] in the Diffusion process besides the Gaussian Walks. In other words, the proposed algorithm hybridizes Differential Evolution and Gaussian Walks in the Diffusion process. Hence, the proposed Diffusion process is named: Differential-Gaussian Diffusion process. This provides an ensemble pool consisting of two schemes to guide the evolutionary search of the exploitation phase in the Diffusion process of Stochastic Fractal Search. In our algorithm, the selection of those schemes is done randomly. Our proposed Diffusion process is started by generating a random number, *rand*. If *rand* is less than 0.5 the SHADE mutation is used as presented in Sub-section 1. Otherwise, a mixture of two Gaussian Walks is used as presented in Sub-section 2.

1) SHADE mutation in the Diffusion process

SHADE mutation is mainly chosen due to its efficiency that enhances the "current-to-pbest/1" mutation strategy which is originally developed by JADE algorithm [19]. Besides that, SHADE introduced effective adaptive settings of *F* and *CR* parameters using a history based scheme. In current-to-pbest/1 as shown in Eq. 14, the best individual is chosen from the top *p* best individuals.

$$V_{i,G} = X_{i,G} + F_{i,G}.(X_{pbest,G} - X_{i,G}) + F_{i,G}.(X_{r_1,G} - X_{r2,G}) \quad (14)$$

where $x_{pbest,G}$ is a random selected individual from the top $NP \times p$ ($p \in [0,1]$) best members of the G^{th} generation, where *NP* is the population size. x_{r1} and x_{r2} are two different random individuals. F_i is an adapted scaling factor associated with individual x_i. In this equation, parameter *p* is the greediness factor and is used to balance the exploitation and exploration search phases in which small *p* values induce a greedier behavior. SHADE assigns an associated p_i for each individual according to Eq. 15 where $p_{\min}$ is set to $2/NP$.

$$p_i = rand[p_{\min}, 0.2] \tag{15}$$

SHADE uses a history-based scheme to adapt the parameter settings. *F* and *CR* are adapted using Cauchy and Normal distributions of successful means same as JADE algorithm except that a random index is selected to choose one of the stored successful means, μF and μCR, in the used memory, *M*, as the following equations show.

$$F_i = randc(\mu F_{r_i}, 0.1) \tag{16}$$

$$CR_i = randn(\mu CR_{r_i}, 0.1) \tag{17}$$

where *randc* and *randn* are the Cauchy and normal distributions of mean μF_{r_i} and μCR_{r_i} respectively with a variance of 0.1. r_i is a random number chosen from $[1, H]$ and *H* is the size of memory *M*. Initially the μF, μCR are both set to 0.5 and are updated at next generations. When $F_i > 1$, it is truncated to 0 and when $F_i \leq 1$, the sampling is repeated until it finds a valid value.

The successful F_i and CR_i are stored in S_F and S_{CR}. The contents of memory of at the end of each generation, μF and μCR, are updated as shown in the following equations:

$$\mu F_{k,g_{s1}+1} = mean_L(S_F) \tag{18}$$

$$\mu CR_{k,g_{s1}+1} = mean_A(S_{CR}) \tag{19}$$

where $mean_L$ is the weighted Lehmer mean and it is computed as shown in Eq. 20 - 22.

$$mean_L(S_F) = \frac{\sum_{k=1}^{|S_F|} w_k . S_{F,k}^2}{\sum_{k=1}^{|S_F|} w_k . S_{F,k}} \tag{20}$$

$$w_k = \frac{\Delta f_k}{\sum_{k=1}^{|S_F|} \Delta f_k} \tag{21}$$

$$\Delta f_k = |f(u_{k,G}) - f(x_{k,G})| \tag{22}$$

and $mean_A$ is the weighted arithmetic mean as shown in Eq. 23.

$$mean_A(S_{CR}) = \sum_{k=1}^{|S_{CR}|} w_k . S_{CR,k} \tag{23}$$

2) Gaussian Walks in Diffusion process

As recommended by SFS authors [21], the first Gaussian Walk (Eq. 9) is usually used for simple problems and the second Gaussian (Eq. 10) Walk is usually used for hard problems. In our proposed algorithm, we used a mixture of the two Gaussian Walks to benefit from the characteristics of both and enhance the evolutionary search further. The selection between those Gaussian walks is done randomly based on a Walk random number *W*. If *W* is less than 0.5, the first Gaussian Walk (Eq. 9) is used. Otherwise, the second Gaussian Walk (Eq. 10) is used.

3) Putting it all together

Our proposed algorithm namely SFS-$DP_{DE\text{-}GW}$ is presented in Fig. 1. Lines 1-2 present the initialization of the population. The algorithm starts by initializing *NP* *D*-dimensional individuals uniformly distributed within the search space of the problem being solved as follows:

$$x^j_{i,0} = x^j_{\min} + rand(0,1).(x^j_{\max} - x^j_{\min}) \quad j = 1,2,...,D \tag{24}$$

where *j* is the index of parameter value in the i^{th} individual vector at time, *t*=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 that, the Differential-Gaussian Diffusion process is called which consists of a combination of SHADE mutation and two Gaussian Walks. Lines 3-15 describe the SHADE mutation in the Diffusion process as presented in Sub-section 1. Lines 17-28 describe the Gaussian Walks in the diffusion process as presented in Sub-section 2. Next, the update process of original SFS is started which consists of two update process as lines 30-32 show.

IV. Experimental Results

The algorithm is tested on a set of 30 benchmark optimization problems from the special session and competition on real-parameter optimization held under the IEEE CEC 2014 [25] benchmarks on 30*D* and 50*D*. The benchmark problems span a diverse set of characteristics such

Algorithm: SFS-$DP_{DE\text{-}GW}$ Algorithm

Input: Entire population at time t : $P(t) = \langle X_1(t), X_2(t), ..., X_{NP}(t) \rangle$
Maximum number of diffusion: γ
Gaussian Walk: *W*

Initialize memory M of μF and μCR with 0.5 with size H

Output: Optimal solution of an optimization problem

1. Initialize population at first time t_0, $P_{t_0} = \langle x_1^{t_0}, \ldots, x_{NP}^{t_0} \rangle$
2. Evaluate population P_{t_0} by calling objective function
3. **While** termination criterion is not met **Do**
4. **Call Differential-Gaussian Diffusion process**
1. **For** j = 1 to γ
2. **If** *rand* < 0.5
3. **Call Differential Diffusion process (SHADE mutation)**
4. **For** i = 1 to NP
5. Generate a random index $r_i = rand(1, H)$
6. Generate $F_i = randc(\mu_{F_{r_i}}, 0.1)$, $CR_i = randn(\mu_{CR_{r_i}}, 0.1)$
7. Apply current-to-pbest (Eq. 14) to generate trial vector u_i
8. **If** $f(u_i) \le f(x_i)$
9. $x_i = u_i$
10. **If** $f(u_i) \ne f(x_i)$
11. $F_i \rightarrow S_F, CR_i \rightarrow S_{CR}$
12. **End If**
13. **End If**
14. **EndFor**
15. Update memory M if $S_F \ne \phi, S_{CR} \ne \phi$
16. **ElseIf**
17. **Call Gaussian Diffusion process**
18. **For** i = 1 to NP
19. **If** *rand* < W
20. Select the best individual $x_{best,g}$ of population P_t
21. Use first Gaussian Walk (Eq. 9) to generate new y_i
22. **Else**
23. Use second Gaussian Walk (Eq. 10) to generate new x_i
24. **EndIf**
25. **If** $f(y_i) \le f(x_i)$
26. $x_i = y_i$
27. **EndIf**
28. **EndFor**
29. **EndIf**
30. **Call Update Process of SFS**
31. Call first update process using Eq. 12
32. Call second update process using Eq. 13
33. **EndWhile**

Fig. 1. Pseudo-code of SFS-DP$_{DE\text{-}GW}$ Algorithm

as unimodal (problems 1-3) multimodality (problems 4-16), hybrid (17-22) composition (23-30) nonseparable, ill-conditioning and noise in fitness. More details can be found in [25]. In summary, functions 1–3 are unimodal, functions 4–16 are multimodal, functions 17–22 are multimodal, and functions 23–30 are composition functions. The performance of the SFS-DP$_{DE\text{-}GW}$ is compared with the results of several other well-known state-of-the-art algorithms.

A. Parametric Setup

The algorithm was coded using Matlab and was run on a PC with 2.26GHz Core processor with 4 GB RAM and windows 7. The algorithm was run 51 times for each test problem with a number of function evaluations equals to $10,000 \times D$. When the difference between the best solution found and optimal is equal to or less than 10^{-8}, the error is treated as 0. All parameter values are set as follows:

- Population size (*NP*) is set to 100
- Memory size (*H*) is set to 10
- Initial mean values for μF and μCR are set to 0.5
- Maximum number of diffusion (γ) is set to 1
- Gaussian Walk (*W*) is set to 0.5

B. Algorithms Compared

The performance of SFS-DV$_{DE\text{-}GW}$ algorithm was compared with the following state-of-the-art algorithms as follows:

1. Differential Evolution with Composite Trial Vector Generation Strategies and Control Parameters (CoDE) [17].
2. Differential evolution algorithm with ensemble of parameters and mutation strategies (EPSDE) [16].
3. Adaptive differential evolution with optional external archive (JADE) [18].
4. Self-adaptive differential evolution algorithm for numerical optimization (SaDE) [14].
5. Success-History Based Parameter Adaptation for Differential Evolution (SHADE) [19].
6. Stochastic Fractal Search (SFS) [21].

We used the same control parameter values that were suggested in the original papers to run the codes. Each algorithm is run for 10,000×*D* functions evaluations for 51 independent runs. For the SFS algorithm, we used the same control parameters as recommended by the authors of the SFS algorithm [21]: the second Gaussian walk (Eq. 10) is used which is recommended to solve the hard problems, the maximum diffusion number (γ) is set to 1 and the population size is set to 100.

C. Results on Numerical Benchmarks

The mean and standard deviation of the best-of-run errors for 51 independent runs of all used algorithms on 30 benchmark problems for 30*D*, and 50*D*, respectively, are given in Tables I–II. The best error values are marked in bold. Table I shows that the SFS-DP$_{DE\text{-}GW}$ on 30*D* is at least as good as the other algorithms in 20 functions.

TABLE I

MEAN AND STANDARD DEVIATION OF THE ERROR VALUES FOR FUNCTIONS F1-F30 @ 30*D*. BEST ENTRIES ARE MARKED IN BOLDFACE.

	CoDE	EPSDE	JADE	SaDE	SHADE	SFS	SFS-DV$_{DE-GW}$
F1	2.6332E+04 (1.8386E+04)	2.4162E+04 (8.2591E+04)	4.4782E+02 (1.0397E+03)	2.9897E+05 (2.3297E+05)	4.8134E+02 (7.8643E+02)	2.1491E+03 (1.0207E+03)	**0.0000E+00 (0.0000E+00)**
F2	**0.0000E+00 (0.0000E+00)**	**0.0000E+00 (0.0000E+00)**	**0.0000E+00 (0.0000E+00)**	**0.0000E+00 (0.0000E+00)**	**0.0000E+00 (0.0000E+00)**	5.1611E-02 (2.8378E-02)	**0.0000E+00 (0.0000E+00)**
F3	**0.0000E+00 (0.0000E+00)**	**0.0000E+00 (0.0000E+00)**	5.6292E-04 (2.5067E-03)	1.4258E+01 (5.8311E+01)	**0.0000E+00 (0.0000E+00)**	3.408E-03 (1.791E-03)	**0.0000E+00 (0.0000E+00)**
F4	2.5174E+00 (1.2583E+01)	3.2127E+00 (2.2348E+00)	**0.0000E+00 (0.0000E+00)**	3.7184E+01 (3.7647E+01)	**0.0000E+00 (0.0000E+00)**	3.485E-01 (4.461E-01)	**0.0000E+00 (0.0000E+00)**
F5	**2.0064E+01 (9.7834E-02)**	2.0347E+01 (3.2166E-02)	2.0287E+01 (3.7919E-02)	2.0536E+01 (5.5479E-02)	2.0101E+01 (1.9092E-02)	2.051E+01 (5.064E-02)	2.0121E+01 (1.8001E-02)
F6	1.9891E+00 (1.6744E+00)	1.8893E+01 (1.6361E+00)	9.4229E+00 (2.1611E+00)	5.4599E+00 (1.9996E+00)	5.2891E-01 (7.2172E-01)	8.656E+00 (3.762E+00)	**0.0000E+00 (0.0000E+00)**
F7	1.4502E-04 (1.0357E-03)	2.0760E-03 (5.5347E-03)	**0.0000E+00 (0.0000E+00)**	1.2333E-02 (1.9953E-02)	4.8332E-04 (1.9731E-03)	1.450E-04 (1.036E-03)	**0.0000E+00 (0.0000E+00)**
F8	**0.0000E+00 (0.0000E+00)**	**0.0000E+00 (0.0000E+00)**	**0.0000E+00 (0.0000E+00)**	7.8036E-02 (3.3553E-01)	**0.0000E+00 (0.0000E+00)**	8.281E-02 (6.178E-02)	**0.0000E+00 (0.0000E+00)**
F9	4.0371E+01 (1.2218E+01)	4.4362E+01 (6.7553E+00)	2.6166E+01 (4.1785E+00)	3.8121E+01 (9.1222E+00)	1.5832E+01 (3.1702E+00)	4.861E+01 (9.015E+00)	**1.3656E+01 (2.3594E+00)**
F10	5.0019E-01 (4.3629E-01)	2.0140E-01 (2.2970E-01)	5.3069E-03 (1.0065E-02)	2.6919E-01 (4.6192E-01)	1.2658E-02 (1.6694E-02)	1.392E+01 (3.829E+00)	**5.6532E-04 (3.1027E-03)**
F11	1.9513E+03 (4.8759E+02)	3.5646E+03 (3.4162E+02)	1.6399E+03 (2.6440E+02)	3.1475E+03 (6.9708E+02)	**1.3969E+03 (2.1231E+02)**	2.246E+03 (2.289E+02)	1.4780E+03 (2.4940E+02)
F12	**5.9989E-02 (3.5730E-02)**	5.2516E-01 (7.9944E-02)	2.7113E-01 (3.7518E-02)	7.9419E-01 (1.1392E-01)	1.6227E-01 (2.4622E-02)	5.575E-01 (7.923E-02)	1.6376E-01 (2.3318E-02)
F13	2.3130E-01 (5.0203E-02)	2.4299E-01 (4.3511E-02)	2.2030E-01 (3.9054E-02)	2.5159E-01 (4.0437E-02)	2.0401E-01 (3.3979E-02)	2.756E-01 (4.710E-02)	**1.9565E-01 (3.6867E-02)**
F14	2.3889E-01 (3.5058E-02)	2.7812E-01 (6.1561E-02)	2.3399E-01 (3.0026E-02)	**2.2853E-01 (3.6057E-02)**	2.2468E-01 (2.8035E-02)	2.103E-01 (2.742E-02)	2.5100E-01 (3.3380E-02)
F15	3.1752E+00 (8.2967E-01)	5.6693E+00 (8.5685E-01)	3.0980E+00 (4.2989E-01)	4.1410E+00 (1.3877E+00)	2.5641E+00 (3.5823E-01)	5.558E+00 (6.750E-01)	**2.4868E+00 (3.6992E-01)**
F16	9.2625E+00 (8.7692E-01)	1.1146E+01 (3.0350E-01)	9.3698E+00 (3.6416E-01)	1.0911E+01 (2.7495E-01)	**9.1478E+00 (4.6678E-01)**	1.048E+01 (2.710E-01)	9.1553E+00 (4.4693E-01)
F17	1.4530E+03 (1.6939E+03)	4.6053E+04 (5.9842E+04)	9.6734E+03 (5.9893E+04)	1.1531E+04 (1.0350E+04)	1.0589E+03 (3.1817E+02)	9.228E+02 (1.948E+02)	**3.6192E+02 (1.1614E+02)**
F18	**1.3443E+01 (5.4219E+00)**	3.3188E+02 (3.9702E+02)	3.5805E+02 (1.5087E+03)	4.4383E+02 (6.5301E+02)	4.9875E+01 (2.4497E+01)	2.821E+01 (6.398E+00)	1.9726E+01 (3.9324E+00)
F19	**2.7041E+00 (4.5214E-01)**	1.3300E+01 (9.9535E-01)	4.4373E+00 (6.7044E-01)	4.0013E+00 (8.6299E-01)	4.3057E+00 (7.2316E-01)	5.074E+00 (5.793E-01)	4.7291E+00 (9.1087E-01)
F20	**1.0913E+01 (5.6106E+00)**	5.0042E+01 (8.1396E+01)	2.8854E+03 (2.3250E+03)	1.2454E+02 (9.5739E+01)	1.2642E+01 (6.8123E+00)	2.641E+01 (3.946E+00)	1.5477E+01 (2.2300E+00)
F21	**2.0005E+02 (1.7005E+02)**	8.8443E+03 (1.1976E+04)	7.5750E+03 (3.7939E+04)	2.8350E+03 (3.0596E+03)	2.5176E+02 (1.1511E+02)	4.301E+02 (1.029E+02)	2.4576E+02 (1.2426E+02)
F22	1.9333E+02 (1.1256E+02)	2.5561E+02 (8.3208E+01)	1.4570E+02 (7.2532E+01)	1.6633E+02 (5.4534E+01)	1.1419E+02 (5.5673E+01)	1.241E+02 (6.656E+01)	**9.0100E+01 (6.4577E+01)**
F23	3.1524E+02 (3.3677E-13)	**3.1401E+02 (8.9971E-13)**	3.1524E+02 (2.2964E-13)	3.1524E+02 (5.0125E-13)	3.1524E+02 (2.2964E-13)	3.152E+02 (1.425E-10)	3.1524E+02 (4.0186E-13)
F24	2.2509E+02 (2.1852E+00)	2.2884E+02 (6.6430E+00)	2.2561E+02 (3.5926E+00)	2.2862E+02 (5.0472E+00)	2.2494E+02 (1.9816E+00)	2.237E+02 (6.597E-01)	2.2389E+02 (1.1340E+00)
F25	2.0362E+02 (6.7389E-01)	**2.0023E+02 (4.0583E-02)**	2.0388E+02 (1.2730E+00)	2.0828E+02 (2.3077E+00)	2.0342E+02 (7.3703E-01)	2.033E+02 (2.358E-01)	2.0316E+02 (4.7200E-02)
F26	1.0023E+02 (5.7694E-02)	1.0026E+02 (3.9448E-02)	1.0218E+02 (1.3974E+01)	1.0614E+02 (2.3705E+01)	1.0020E+02 (3.3515E-02)	1.003E+02 (4.940E-02)	**1.0019E+02 (2.0958E-02)**
F27	3.8340E+02 (3.6817E+01)	8.3334E+02 (1.1550E+02)	3.4415E+02 (5.1560E+01)	4.1786E+02 (3.7565E+01)	3.2454E+02 (3.8865E+01)	4.002E+02 (7.977E-02)	**3.0196E+02 (1.4003E+01)**
F28	8.3876E+02 (3.3236E+01)	**3.9831E+02 (1.4468E+01)**	8.0273E+02 (3.5775E+01)	8.9045E+02 (2.9373E+01)	8.4047E+02 (3.1556E+01)	8.918E+02 (3.056E+01)	8.3445E+02 (1.7572E+01)
F29	7.8866E+02 (1.2518E+02)	**2.1430E+02 (1.1392E+00)**	7.5909E+02 (1.5931E+02)	1.0792E+03 (1.8654E+02)	7.1516E+02 (6.4332E+01)	3.780E+02 (6.259E+01)	7.1637E+02 (2.0265E+00)
F30	9.9971E+02 (4.5398E+02)	**5.6708E+02 (1.1242E+02)**	1.9192E+03 (6.4143E+02)	1.6680E+03 (5.7200E+02)	1.5656E+03 (7.2836E+02)	**8.598E+02 (1.534E+02)**	1.1006E+03 (4.2661E+02)

For 50*D* as shown in Table II, the SFS-DP$_{DE-GW}$ is at least as good as the other algorithms in 23 functions when increasing the dimensionality to 50*D*. This affirms the effectiveness and robustness of the proposed algorithm with increasing the

TABLE II

MEAN AND STANDARD DEVIATION OF THE ERROR VALUES FOR FUNCTIONS F1-F30 @ 50*D*. BEST ENTRIES ARE MARKED IN BOLDFACE.

	CoDE	EPSDE	JADE	SaDE	SHADE	SFS	SFS-DV$_{DE-GW}$
F1	2.3214E+05 (1.0747E+05)	1.7374E+06 (5.5064E+06)	1.5275E+04 (1.3422E+04)	9.3147E+05 (3.1419E+05)	1.8614E+04 (1.3638E+04)	3.5520E+04 (2.9330E+04)	**2.7585E+00 (5.9779E+00)**
F2	6.5010E+01 (1.9284E+02)	1.4999E-08 (3.4609E-08)	**0.0000E+00 (0.0000E+00)**	3.7564E+03 (3.8115E+03)	**0.0000E+00 (0.0000E+00)**	1.8218E+03 (1.3552E+03)	**0.0000E+00 (0.0000E+00)**
F3	2.5387E+01 (4.1188E+01)	3.7908E-04 (1.8292E-03)	4.5134E+03 (2.3441E+03)	3.4631E+03 (2.2424E+03)	**0.0000E+00 (0.0000E+00)**	8.4882E-01 (4.2219E-01)	**0.0000E+00 (0.0000E+00)**
F4	2.6067E+01 (3.3789E+01)	4.1596E+01 (2.2985E+01)	1.9346E+01 (3.9290E+01)	8.2104E+01 (4.4460E+01)	**9.7224E+00 (2.9435E+01)**	8.1011E+01 (2.4655E+01)	1.5442E+01 (3.9417E+01)
F5	**2.0032E+01 (6.8626E-02)**	2.0596E+01 (3.2400E-02)	2.0356E+01 (3.6845E-02)	2.0733E+01 (4.1549E-02)	2.0140E+01 (1.9410E-02)	2.0639E+01 (5.5436E-02)	2.0138E+01 (5.3443E-02)
F6	8.4983E+00 (3.3570E+00)	4.5596E+01 (2.6277E+00)	1.6589E+01 (6.6285E+00)	1.8148E+01 (3.2634E+00)	5.1707E+00 (2.4284E+00)	1.2087E+01 (5.0632E+00)	**1.9080E-01 (4.4651E-01)**
F7	1.5467E-03 (3.1935E-03)	5.1192E-03 (7.5766E-03)	2.0769E-03 (4.9084E-03)	1.3887E-02 (1.5352E-02)	3.9093E-03 (7.4077E-03)	5.8008E-04 (2.0082E-03)	**0.0000E+00 (0.0000E+00)**
F8	4.8773E-01 (8.2941E-01)	1.9509E-02 (1.3932E-01)	**0.0000E+00 (0.0000E+00)**	1.0925E+00 (1.0945E+00)	**0.0000E+00 (0.0000E+00)**	8.9156E-02 (1.5473E-01)	**0.0000E+00 (0.0000E+00)**
F9	8.2582E+01 (1.7230E+01)	1.4593E+02 (1.6322E+01)	5.1867E+01 (8.6673E+00)	9.1673E+01 (1.4268E+01)	3.4182E+01 (6.1592E+00)	1.0525E+02 (1.6659E+01)	**3.1522E+01 (4.4965E+00)**
F10	5.1856E+00 (3.0648E+00)	5.5334E+02 (6.0708E+02)	1.2247E-02 (1.3099E-02)	1.2284E+00 (1.0759E+00)	1.0287E-02 (1.1368E-02)	2.1706E+01 (3.8438E+00)	**2.2883E-03 (4.8075E-03)**
F11	4.3391E+03 (9.4713E+02)	8.9455E+03 (5.3473E+02)	3.8695E+03 (2.7785E+02)	6.8299E+03 (1.5764E+03)	3.5552E+03 (3.2722E+02)	4.7676E+03 (3.7755E+02)	**3.4391E+03 (3.0948E+02)**
F12	**8.7329E-02 (4.5691E-02)**	8.5469E-01 (6.9658E-02)	2.5535E-01 (3.1321E-02)	1.0986E+00 (1.0537E-01)	1.6366E-01 (2.1190E-02)	6.1671E-01 (7.2547E-02)	1.6510E-01 (2.0162E-02)
F13	3.3636E-01 (4.9736E-02)	3.6268E-01 (5.9205E-02)	3.3223E-01 (4.9590E-02)	4.3681E-01 (5.4721E-02)	3.1034E-01 (4.5483E-02)	3.8390E-01 (4.7927E-02)	**2.4304E-01 (3.0417E-02)**
F14	**2.7652E-01 (2.9839E-02)**	3.5209E-01 (9.0136E-02)	3.1348E-01 (9.1988E-02)	3.1170E-01 (3.2642E-02)	2.9453E-01 (5.7219E-02)	2.5988E-01 (2.2820E-02)	3.2397E-01 (8.3224E-02)
F15	7.4043E+00 (1.3955E+00)	1.8616E+01 (2.6294E+00)	7.0826E+00 (9.4236E-01)	1.7092E+01 (5.8042E+00)	5.7376E+00 (6.1481E-01)	1.2893E+01 (1.6214E+00)	**6.8975E+00 (1.3995E+00)**
F16	1.8290E+01 (8.9113E-01)	2.0695E+01 (4.1566E-01)	1.7720E+01 (4.3341E-01)	2.0222E+01 (2.7357E-01)	**1.7444E+01 (4.2728E-01)**	1.9123E+01 (4.6960E-01)	1.7425E+01 (4.7906E-01)
F17	1.5714E+04 (1.3291E+04)	2.2167E+05 (1.4896E+05)	2.3000E+03 (5.9502E+02)	5.9118E+04 (3.9220E+04)	2.2347E+03 (7.8109E+02)	2.0599E+03 (3.9736E+02)	**1.3574E+03 (3.9909E+02)**
F18	3.2916E+02 (3.4839E+02)	3.6879E+03 (8.1501E+03)	1.8203E+02 (4.7150E+01)	7.5884E+02 (5.1263E+02)	1.5375E+02 (3.6617E+01)	7.2489E+01 (1.8364E+01)	**9.9619E+01 (1.5245E+01)**
F19	**6.1691E+00 (9.8380E-01)**	2.4584E+01 (1.6861E+00)	1.3421E+01 (5.2199E+00)	1.6741E+01 (8.7552E+00)	9.3810E+00 (3.2466E+00)	1.0247E+01 (2.1614E+00)	1.1124E+01 (3.0602E+00)
F20	2.1861E+02 (2.3515E+02)	4.5080E+02 (5.6348E+02)	7.9529E+03 (6.9640E+03)	9.0611E+02 (7.3143E+02)	1.9816E+02 (5.7311E+01)	7.4670E+01 (1.3192E+01)	**4.7599E+01 (8.9308E+00)**
F21	6.4912E+03 (5.5372E+03)	7.4370E+04 (8.5331E+04)	2.4322E+04 (1.6455E+05)	8.5745E+04 (4.5410E+04)	1.2297E+03 (4.0604E+02)	1.1860E+03 (2.0900E+02)	**4.6959E+02 (1.4816E+02)**
F22	6.0223E+02 (2.2061E+02)	8.0373E+02 (2.0458E+02)	5.1982E+02 (1.3024E+02)	5.1001E+02 (1.7759E+02)	**3.6769E+02 (1.5946E+02)**	4.9241E+02 (1.4991E+02)	4.0148E+02 (1.2036E+02)
F23	**3.4400E+02 (1.1482E-13)**	**3.3705E+02 (3.4523E-12)**	**3.4400E+02 (3.7646E-13)**	**3.4400E+02 (1.1482E-13)**	**3.4400E+02 (1.1482E-13)**	**3.4400E+02 (1.6202E-12)**	**3.4400E+02 (4.5927E-13)**
F24	2.7139E+02 (2.9312E+00)	2.7312E+02 (5.7937E+00)	2.7482E+02 (1.9487E+00)	2.7596E+02 (3.0905E+00)	2.7478E+02 (1.7334E+00)	**2.6156E+02 (4.6475E+00)**	2.7509E+02 (8.5349E-01)
F25	2.0792E+02 (4.7505E+00)	2.0116E+02 (2.0229E+00)	2.1699E+02 (6.7636E+00)	2.1665E+02 (9.7720E+00)	2.1217E+02 (6.9165E+00)	2.0746E+02 (6.7402E-01)	**2.0546E+02 (3.2251E-01)**
F26	1.0618E+02 (2.3701E+01)	1.0035E+02 (4.7521E-02)	1.0237E+02 (1.3950E+01)	1.9030E+02 (2.9941E+01)	1.0431E+02 (1.9529E+01)	1.0033E+02 (4.0009E-02)	**1.0024E+02 (2.6243E-02)**
F27	5.3680E+02 (6.5540E+01)	1.5566E+03 (5.8870E+01)	4.4058E+02 (5.6802E+01)	7.8213E+02 (6.2606E+01)	4.5329E+02 (5.6995E+01)	4.8675E+02 (1.2906E+02)	**3.3639E+02 (3.5793E+01)**
F28	1.1734E+03 (4.4835E+01)	**3.8746E+02 (1.1591E+01)**	1.1289E+03 (4.1317E+01)	1.4354E+03 (9.8737E+01)	1.1413E+03 (5.2616E+01)	1.4761E+03 (1.1827E+02)	1.1112E+03 (2.1985E+01)
F29	9.2726E+02 (1.3813E+02)	**2.2556E+02 (1.1299E+01)**	9.0482E+02 (1.2689E+02)	1.4224E+03 (3.1715E+02)	8.9071E+02 (5.5375E+01)	6.8024E+02 (1.2202E+02)	8.0519E+02 (3.7243E+01)
F30	9.0213E+03 (5.3860E+02)	**1.0618E+03 (2.0931E+02)**	9.7197E+03 (8.1792E+02)	1.1613E+04 (1.8321E+03)	9.3841E+03 (7.8714E+02)	9.8488E+03 (5.5581E+02)	8.7739E+03 (4.1797E+02)

dimension size of the problem. The results in these tables show that there is a clear difference between the proposed algorithm and its used components which are: SFS and SHADE algorithms. The SFS-DP$_{DE-GW}$ is at least as good as SFS in 21 and 22 functions on 30*D* and 50*D,* respectively. Compared to SHADE algorithm, the SFS-DP$_{DE-GW}$ is at least as good as in 28, 26 functions on 30*D* and 50*D* respectively. This proves the effectiveness of the proposed Differential-Gaussian

Diffusion process which hybridizes the SHADE mutation with a mixture of two Gaussian Walks.

V. Conclusion

In this paper, a new evolutionary algorithm is proposed that combines the explorative and exploitative capabilities of two evolutionary algorithms namely, Stochastic Fractal Search (SFS) and Differential Evolution (DE) algorithm. The proposed algorithm, SFS-$DP_{DE\text{-}GW}$ embeds the Differential Evolution in Stochastic Fractal Search. In this algorithm, we introduce a new adjustment to the Diffusion Process of Stochastic Fractal Search by hybridizing the SHADE mutation and a mixture of two Gaussian Walks. This combination enhances the evolutionary search clearly and increases the efficiency of update process which is called after the Diffusion process. The performance of the algorithm is evaluated on a set of 30*D* optimization problems. The results show that the SFS-$DP_{DE\text{-}GW}$ has a competitive performance and a favorable scalability behavior compared to other state-of-the-art algorithms.

Acknowledgment

This research was supported by funding from the Agency for Science, Technology and Research of Singapore (A*STAR) and Nanyang Technological University. We would like to thank the deanship of research in Jordan University of Science and Technology (JUST) for their help. Dr. Mostafa Z. Ali was on sabbatical leave from JUST with PSUT during the work done for this research.

References

[1] T.T. Nguyen, X. Yao, "An Experimental Study of Hybridizing Cultural Algorithms and Local Search, Int. J. of Neural Systems," 18 (2008) 1-18.

[2] F. Glover, "Tabu search-part I," ORSA J. Comput., 1 (1989) 190–206.

[3] F. Glover, "Future paths for integer programming and links to artificial intelligence," Comput. Oper. Res., 13 (1972) 533 – 549

[4] M.Z. Ali, N.H. Awad, "A novel class of niche hybrid cultural algorithms for continuous engineering optimization," Inf. Sci. 267 (2014) 158–190

[5] Y. Wang, H.-X. Li, T. Huang, L.Li, "Differential evolution based on covariance matrix learning and bimodal distribution parameter setting," Applied Soft Computing 18 (2014) 232–247

[6] A. Zhang, G. Sun, Z. Wang, Y. Yao, "a hybrid genetic algorithm and gravitational search algorithm for global optimization," vol 25, no 1 (2015), neural network world

[7] M.Z. Ali, N.H. Awad, R.G. Reynolds, "Hybrid Niche Cultural Algorithm for Numerical Global Optimization," In Proc. IEEE Congress of Evolutionary Computations, June 2013., pp. 309-316, Cancun, Mexico

[8] M.Z. Ali, N.H. Awad, P.N. Suganthan, "Multi-population Differential Evolution with Balanced Ensemble of Mutation Strategies for Large-Scale Global Optimization," Applied Soft Computing, V. 33, pp. 304-327, 2015

[9] R. Storn and K. Price, "Differential evolution a simple and efficient heuristic for global optimization over continuous spaces," J. Global Optimization, vol. 11, no. 4, pp. 341–359, 1997.

[10] K.V. Price, R.M. Storn, J.A. Lampinen, "Differential Evolution: A Practical Approach to Global Optimization," *1st ed. New York: Springer-Verlag*, Dec. 2005.

[11] J. Zhang, V. Avasarala, A.C. Sanderson, T. Mullen, "Differential evolution for discrete optimization: An experimental study on combinatorial auction problems," in Proc. IEEE World Congr. Comput. Intell., Hong Kong, China, Jun. 2008, pp. 2794–2800

[12] J. Zhang, A.C. Sanderson, "An approximate Gaussian model of differential evolution with spherical fitness functions," in Proc. IEEE Congr. Evol. Comput., Singapore, Sep. 2007, pp. 2220–2228.

[13] R. Gamperle, S. D. Muller, P. Koumoutsakos, "A parameter study for differential evolution," in Proc. Advances Intell. Syst., Fuzzy Syst.,Evol. Comput., Crete, Greece, 2002, pp. 293–298.

[14] A. K. Qin, P. N. Suganthan, "Self-adaptive differential evolution algorithm for numerical optimization," in Proc. IEEE Congr. Evol. Comput., vol. 2. Sep. 2005, pp. 1785–1791.

[15] 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., vol. 10, no. 6, pp. 646–657, Dec. 2006.

[16] R. Mallipeddi, P. N. Suganthan, Q. K. Pan, M. F. Tasgetiren, "Differential evolution algorithm with ensemble of parameters and mutation strategies," Appl. Soft Comput., vol. 11, no. 2, pp. 1679–1696, 2011.

[17] Y. Wang, Z. Cai, Q. Zhang, "Differential Evolution With Composite Trial Vector Generation Strategies and Control Parameters," IEEE Tran. Evol. Comput., vol. 15, no. 1, pp. 55–66, 2011.

[18] J. Zhang, A.C. Sanderson, "JADE: Adaptive differential evolution with optional external archive," IEEE Trans Evol Comput, vol. 13, no. 5, pp. 945–958, 2009

[19] R. Tanabe, A. Fukunaga, "Success-History Based Parameter Adaptation for Differential Evolution," in IEEE CEC, 2013, pp. 71–78.

[20] R. Tanabe, A. Fukunaga, "Improving the Search Performance of SHADE Using Linear Population Size Reduction," in IEEE CEC, 2014, pp. 1658 - 1665.

[21] H. Salimi, "Stochastic Fractal Search: A powerful metaheuristic algorithm," Elsevier, Knowledge-Based Systems, V. 75, pp. 1-18, 2015

[22] R. Storn and K. V. Price, "Differential evolution-A simple and efficient heuristic for global optimization over continuous Spaces," Journal of Global Optimization, vol.11, pp.341-359. 1997.

[23] S. Das and P. N. Suganthan, "Differential evolution: A survey of the state-of-the-art," IEEE Tran. Evol. Comput., vol. 15, no. 1, pp. 4–31,2011.

[24] K.V. Price, R.M. Storn, J.A. Lampinen, Differential Evolution: A Practical Approach to Global Optimization, 1st ed. New York: Springer-Verlag, Dec, 2005

[25] 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," Zhengzhou University and Nanyang Technological University, Tech. Rep., 2013.