A Differential Evolution Algorithm with Success-based Parameter Adaptation for CEC2015 Learning-based Optimization

Noor Awad, Mostafa Z. Ali
Jordan University of Science & Technology
Irbid, Jordan 22110
{noorawad1989, mzali.pn}@gmail.com

Robert G. Reynolds
Wayne State University
Detroit, MI USA 48202
Robert.reynolds@wayne.edu

***Abstract*— Developing efficient evolutionary algorithms for solving learning-based real-parameter single objective optimization is a very challenging and essential task in many real applications. This task involves finding the best optimal solution with least computational cost, avoiding premature convergence. This paper proposes a new efficient Differential Evolution algorithm with success-based parameter adaptation with resizing population space. We introduce a new technique to adapt the control parameters which uses a memory-based structure of previous successful settings. Moreover, the population size is adapted linearly to find the most suitable size which helps to guide the search in each optimization loop. The proposed algorithm is tested on the benchmarks of the CEC2015 real parameter single objective competition. The results affirm the efficiency and robustness of our approach to reach good results.**

Keywords*— *evolutionary algorithms, differential evolution, premature convergence, single objective optimization

I. INTRODUCTION

The optimization field is concerned with the selection of the best feasible solution from a set of available alternatives based on solution objectives and some constraints criteria [1]. Evolutionary algorithms and other nature-inspired techniques were introduced to deal with single objective optimization problems such as: biological Evolutionary Computation (EC) [2-4,], Tabu Search (TS) [5], Genetic Algorithm (GA) [6], [7], Simulated Annealing (SA) [8], Particle Swarm Optimization (PSO) [9]-[11] and Ant Colony Optimization (ACO) [12].

Differential Evolution was proposed by Storn and Price [13] as a heuristic evolutionary algorithm that was recently designed to solve optimization problems over continuous spaces. Several types of mutation and crossover strategies were proposed for DE [14]-[18]. Like any other evolutionary algorithm, DE has three main control parameters which are the population size *NP*, scaling factor *F* and crossover rate *CR*. Those three control parameters play a dominant role when assessing the performance of the DE algorithm. It is known that those settings are problem-dependent and hence it is necessary to fine-tune those control parameters to guide the search toward optimal results. As a result, many researchers introduced self-adaptive mechanisms to adjust control parameters during the search process [19]-[24].

One of the early works that seeks to automate the control parameter settings as well the selection of a mutation strategy is SaDE algorithm [19]. It uses a memory to save the past search behavior to select the best mutation strategy along with the associated parameters to perform relatively well in each optimization loop. It uses a pool to store 4 strategies and one of them is probabilistically chosen for each individual by recording the number of successes and failures. This randomized selection is biased with a higher number of successes of a strategy. The scaling factor F does not adapt and a normal distribution of a fixed mean and standard deviation is used. In contrast, the crossover rate CR is adapted by sampling the successful CR values over previous generations using normal distribution.

Another algorithm that is introduced to adapt the control parameters of DE algorithm is jDE [20]. This algorithm assigns different values for F and CR for each individual in the population. In each search loop, each parameter is changed according to some heuristic rules. In EPSDE algorithm [21], they used three separate pools for control parameters F, CR and for mutation strategy that includes: rand/1/bin, best/2/bin and current-to-rand/1. In each generation, the parameters are inherited by successful individuals.

One of the most effective DE algorithms is JADE [22], which introduces a novel mutation strategy called current-to-pbest/1 with an external optional archive to save previously generated individuals. This new mutation strategy uses a random individual that is selected from the top best individuals in each loop. F and CR values are adapted using the normal and Cauchy distributions respectively with modified means MCR and MF. The successful values of CR and F in each generation are used to update the mean values MCR and MF. SHADE is another algorithm developed based on JADE algorithm [23]. They adapted the greediness factor in JADE and used a history-based mechanism to enhance the parameter adaptation. Another version of SHADE algorithm is also introduced in which they used a linear population size reduction [24].

In this paper, we introduce an enhancement of JADE algorithm namely DEsPA which uses a success-based parameter adaptation with resizing population space. The algorithm starts with a small number of individuals that are

978-1-4799-7492-4/15/$31.00 ©2015 IEEE

needed for an optimization loop according to selected mutation strategy which is current-to-pbest/1. At the next generations, a linear population size proliferation is used to increase the size of population. When the maximum size is reached, the population is kept with a fixed size for some generations to give a higher chance for them to grow up and to move to promising regions and increase their survival before the second round of linear population size reduction is started. This is different from JADE which uses a fixed size of population. The F and CR values are adapted using a memory-based scheme to store the best behaviors of successful individuals. As a result, the proposed algorithm seeks to adapt all the control parameters of DE algorithm which are: F, CR and NP.

The remaining sections complete this paper as follows. Section 2 briefly introduces Differential Evolution and JADE algorithm. In Section 3, the proposed method is elaborated. Section 4 describes the optimization problems, parameter settings and simulation results. Finally, section 5 summarizes the conclusion of this paper and future work.

II. SCIENTIFIC BACKGROUND

A. Classical Differential Evolution

The Differential Evolution algorithm is one of the most promising population-based evolutionary algorithms because of its simplicity with powerful stochastic direct search technique [25]. The standard DE consists of a population of *NP* individuals. Each individual is represented as a vector of D-dimensional parameters as follows:

$$X_{i,G} = \left\{ x^{1}_{i,G}, ..., x^{D}_{i,G} \right\}, \; i = 1, ..., NP \tag{1}$$

Where G is the generation number and NP is the number of individuals in the population space. The DE algorithm aims at evolving the population space towards the global optima. The algorithm distributes individuals randomly within the search space of the problem being solved with D dimensions where each parameter of the problem is represented by a lower and upper bounds of as $X_{\min} = \left\{ x^{1}_{\min}, ..., x^{D}_{\min} \right\}$ and $X_{\max} = \left\{ x^{1}_{\max}, ..., x^{D}_{\max} \right\}$, receptively. Hence, the initialization of the population space at generation G=0 is formulated by the following equation:

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

Where j is the index of parameter value in the i^{th} individual at generation G=0, and $rand(0,1)$ is a uniformly distributed random generator in the range [0,1].

Five mutation strategies were suggested for the original DE [14][15][16][17][18] as described below:

"DE/rand/1": $v_i = x_{r_1} + F \cdot (x_{r_2} - x_{r_3})$ (3)

"DE/best/1": $v_i = x_{best} + F \cdot (x_{r_1} - x_{r_2})$ (4)

"DE/current-best/1":

$$v_i = x_i + F \cdot (x_{best} - x_i + x_{r_1} - x_{r_2}) \tag{5}$$

"DE/best/2":

$$v_i = x_{best} + F \cdot (x_{r_1} - x_{r_2} + x_{r_3} - x_{r_4}) \tag{6}$$

"DE/rand/2":

$$v_i = x_{r_1} + F \cdot (x_{r_2} - x_{r_3} + x_{r_4} - x_{r_5}) \tag{7}$$

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 random integer numbers.

After mutant vectors have been generated, the crossover phase is applied to generate new offspring or 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 [13]:

$$u^{j}_{i,G} = \left\{ \begin{array}{l} v^{j}_{i,G}, \; if \;\; (with \;\; probability \;\; of \;\; CR) \;\; or \;\; (j = j_{rand}) \\ x^{j}_{i,G}, \; otherwise \end{array} \right\} \; j = 1, 2, ..., D \tag{8}$$

CR is the crossover rate which is 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.

To ensure that the new trial vectors are within upper and lower bounds, DE must check parameter values of its trial vectors and when they exceed the search range, their values will be reinitialized. After that, the fitness values of trial vectors are calculated by computing the objective function of the problem being solved. Then, a selection criterion is performed as in Eq. 9 to fill in the population space with new individuals for the next generation by comparing fitness values of target and trial vectors.

$$X_{i,G+1} = \left\{ \begin{array}{l} U_{i,G}, if \; f(U_{i,G}) \le f(X_{i,G}) \\ X_{i,G}, otherwise \end{array} \right\} \tag{9}$$

The above DE steps including mutation, crossover and selection will be repeated until a termination criterion is met which is usually after finishing a specified number of generations.

B. Review of JADE Algorithm

The basis of our proposed algorithm is JADE and it is mainly chosen because of its main features which are: a new mutation strategy namely "current-to-pbest/1" with an optional external archive and adaptive settings of F and CR values. The following subsections briefly describe each of these.

1) current-to-pbest/1 with optional external archive

This mutation strategy is an extension of basic current-to-*p*best/1 where the best individual is updated as shown in Eq. 10.

$$v_{i,g} = x_{i,g} + F_{i,g} .(x_{pbest,g} - x_{i,g}) + F_{i,g} .(x_{r_1,g} - x_{r2,g}) \tag{10}$$

Where $x_{pbest,g}$ is a random selected individual from the top $NP \times p$ ($p \in [0,1]$) best members of g^{th} generation. F_i is an adapted scaling parameter associated with individual x_i. Eq. 10 shows that parameter p is the control of greediness and is used to balance between exploitation and exploration in which small p values behave more greedily. JADE uses an optional external archive A to maintain diversity of size equal to the population size. When the archive is used, it stores the parent vectors $x_{i,g}$ that were worse than the trial vector $u_{i,g}$. In this case, the individual $x_{r2,g}$ in Eq. 10 is chosen from the union of population P and archive A $(P \cup A)$. Randomly selected individuals from archive A are deleted when the size exceeds the limit to make space for newly individuals.

2) Parameter Adaptaion

Each individual of population has its own F_i and CR_i values that are used to generate trial vectors. Those parameters are updated in an adaptive manner as shown below:

$$F_i = randn_i(\mu F, 0.1) \tag{11}$$

$$CR_i = randc_i(\mu CR, 0.1) \tag{12}$$

Where $randn_i(\mu F, 0.1)$, $randc_i(\mu CR, 0.1)$ are values sampled from normal and Cauchy distributions of mean μF, μCR and a variance 0.1. Initially the μF, μCR are set both to 0.5 and are adapted during the search process. 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.

In each optimization loop, the successful values of F_i, CR_i, that are able to generate a trial vector better than the parent vector $x_{i,g}$, are recorded in S_{CR}, S_F. At the end of the optimization loop, the two means μF, μCR are modified as follows:

$$\mu F = (1-c).\mu F + c.mean_L(S_F) \tag{13}$$

$$\mu CR = (1-c).\mu CR + c.mean_A(S_{CR}) \tag{14}$$

Where c is a meta level control parameter and set to 0.1 as suggested in JADE. $mean_L$ is the Lehmer mean and it is computed as shown in Eq. 15 while $mean_A$ is the regular arithmetic mean.

$$mean_L(S_F) = \frac{\sum_{F \in S_F} F^2}{\sum_{F \in S_F} F} \tag{15}$$

III. DE WITH SUCCESS-BASED PARAMETER ADAPTATION

The proposed algorithm namely DEsPA is an enhanced version of JADE and L-SHADE algorithms without using external archives. The algorithm focuses on solving some weaknesses and increases its performance by adding the following features, explained in comparison to recent state-of-the-art adaptive algorithms:

- Adapting the greediness factor which is denoted by p parameter: The JADE algorithm maintains this value static and is set manually and this affects the exploration and exploitation capabilities of search process. By adapting this parameter, it gives more control on the greediness factor and it provides an effective way to balance between the exploration and exploitation capabilities. SHADE algorithm which is an enhanced version of JADE, adapted p and the key difference between our methodology and theirs is that our adaptation is connected to the population size parameter *NP* which is also adapted in our algorithm. Using this way, parameter p behaves much better than in SHADE algorithm.

- The proposed algorithm maintains a diverse set of parameters to guide the control settings of *F*, *CR* by using a memory based scheme: in JADE they used a single pair represented by mean values μF, μCR to guide adaptation. According to probabilistic nature of DE, the poor settings of F and CR are possibly included in S_{CR}, S_F. If this case happens, μF, μCR are affected to move the search process to undesirable regions and hence the performance of JADE is degraded. SHADE also used the same way by using a history-based mechanism but the key difference here is that the memory used in our algorithm has an adapted size based on the population size *NP*. This way provides the good size for the used memory.

- Adapting the population size *NP*: although JADE adapts the control parameters of mutation strategy, F_i and CR_i but it maintains the size of the population fixed for the whole search process. This work proposes a new and a novel way of adapting *NP* parameter in which a linear population size proliferation and also reduction is used. L-SHADE algorithm used a linear reduction to decrease the size of population in each generation. It starts its evolution by a large number of individuals equals to $D \times 20$ and this wastes a considerable number of function evaluations at first generations without further improvements. And also there is no higher chance for individuals to survive from elimination. In our algorithm we start the search process with the least possible number of individuals to start an optimization loop to save a good number of function evaluations for later generations. The proliferation then starts linearly and the new starter individuals will have higher chance for survival as there is

a saturation phase for a number of generations before the reduction starts.

If the above features are well designed, the robustness of the algorithm increases by adapting all the parameters that affect the performance of DE algorithm. In addition the convergence rate is improved by increasing the diversity of the population. The following subsections explain how each of these targets is accomplished in the proposed algorithm.

A. Adapting the greediness parameter p

The JADE algorithm used a fixed value of parameter *p* that aims to adjust the greediness of mutation strategy current-to-pbest/1. In DEsPA algorithm, the parameter *p* is adapted to provide more control of mutation greediness and hence control the exploration and exploitation in an effective manner. Each individual in the population has its own p_i and is set according to Eq. 16. The maximum value 0.2 of parameter *p* is suggested by JADE and according to this, the parameter *N* is chosen to behave more greedily and balance exploitation and exploration of current-to-*p*best mutation strategy. As a result a value of 0.1 multiplied by population size is chosen.

$$p_i = rand(0,1) \times N \ \ where$$
$$N = NP \times 0.1 \tag{16}$$

In this manner, the $x_{pbest,g}$ in Eq. 10 is selected using an adaptive manner to guide the search toward promising regions.

B. Success-based parameter adaptation

A memory scheme is used to enhance the adaptation mechanism of F and CR values of JADE algorithm. Two-dimensional memory structure is used to store the mean values μF and μCR to maintain a diverse set of parameters for better guiding control settings. The size of memory *M* depends on population size *NP* and is computed as shown in Eq. 17. All the contents of memory are initialized to 0.5.

$$M = \frac{NP}{2} \tag{17}$$

When applying equations 13 and 14, firstly a random index k_i is chosen such that $k_i = rand(1, M)$ where *M* is the size of memory. This index in used to select the μF and μCR values as shown below:

$$F_i = randc_i(\mu F_{ki}, 0.1) \tag{18}$$

$$CR_i = randn_i(\mu CR_{ki}, 0.1) \tag{19}$$

The same procedure as JADE algorithm is used if values of F_i and CR_i are outside valid range.

The successful values of F_i and CR_i are stored in S_F and S_{CR} as with JADE but the weighted Lehmer mean is used for computing both the μF and μCR to update the memory at index k_i as follows:

$$mean_{WL}(S_F) = \frac{\sum_{F \in S_F} w_k . S_F^{\,2}}{\sum_{F \in S_F} w_k . S_F} \tag{20}$$

$$w_k = \frac{\triangle f_k}{\sum_{F \in S_F} \triangle f_k} \tag{21}$$

$$\triangle f_k = |f_{u_k,g} - f_{x_k,g}| \tag{22}$$

C. Adaptating population size NP

The DEsPA algorithm is adapts the population size *NP* by resizing it in both directions, increasing and decreasing. This introduces a novel way of adapting the *NP* parameter to maintain a higher diversity during the search process of each generation. Fig. 1 shows the framework of this adaptation.

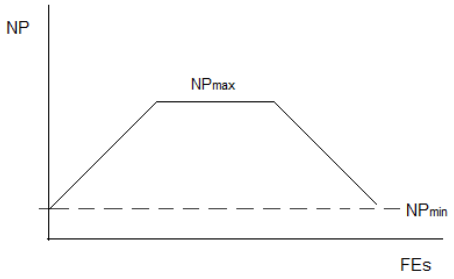

Fig. 1. Adaptation Framework of popualtion size

Initially the size of population is set to the minimum number that is needed to start an optimization loop according to chosen mutation strategy current-to-pbest/1 which is 4 in our case. In each next generation, a linear population size proliferation is started to increase population size as shown in the following equation:

$$NP_{plus} = Round[(\frac{NP + NP_{\max}}{FEs_{\max}}).FEs + NP_{\min}] \tag{23}$$

Where NP_{plus} is number of extra individuals to be added to the current population with size *NP*. $NP_{\max}$ is the maximum number of individuals to be reached. $FEs_{\max}$ is the maximum number of function evaluations. $NP_{\min}$ is the smallest possible value that is needed to perform mutation strategy. After that, the population size is increased by NP_{plus} individuals as shown in Eq. 24.

$$NP_{new} = NP + NP_{plus} \tag{24}$$

Those new individuals are computed using the same way that is used for initialization as shown in Eq. 2 except that when

the values of $x_{i,g}$ exceed lower and upper boundaries, the re-initialization as described in [22] is used as follows:

$$v_{i,j,g} = \begin{cases} \dfrac{x_j^{\min} + x_{i,j,g}}{2}, if\ v_{i,j,g} < x_j^{\min} \\ \dfrac{x_j^{\max} + x_{i,j,g}}{2}, if\ v_{i,j,g} > x_j^{\max} \end{cases} \quad (25)$$

After the proliferation reaches the threshold which is determined by parameter $NP_{\max}$, the size will be fixed for a number of generations. This saturation will give higher chance for new individuals that are added in each generation to grow up and reach more promising regions and increase their survival before the population starts to reduce again.

As with proliferation, a linear reduction is used to reduce the size of the population space as shown in in Eq. 26. The worst $NP_{\min us}$ individuals are eliminated according to fitness values.

$$NP_{\min us} = Round[(\frac{NP_{\min} - NP}{FEs_{\max}}).FEs + NP] \quad (26)$$

D. Overall algorithm

Fig. 2 shows the overall implementation of DEsPA algorithm. It starts with the initialization phase to set all required parameters. In each optimization loop, trial vectors are generated using success-based adaptation as was discussed in section III-B. After that, the selection is applied and the memory contents are updated based on the success of previous generations. The size of population is also adapted by linearly increasing it in the first generations and then reducing it in later generations as shown in section III-C. Those steps are repeated until termination criterion is met. It should be noticed that using the weighted Lehmer mean adds a necessary constraint in which F_i and CR_i values are just updated when trial vector is less than but not equal to target vector, $f(u_{i,g}) < f(v_{i,g})$. If they are equal, this results to inappropriate setting of parameter updates as we use the weight of score differences as shown in Eq. 22.

Algorithm: DEsPA

1. Initialize population at first generation g_0, $P_{g_0} = \langle x_1^{g_0}, \ldots, x_{NP}^{g_0} \rangle$
2. initialize memory M of μF and μCR with 0.5
3. **While** termination criterion is not met **Do**
4. Reset S_F and S_{CR}, $S_F = \phi, S_{CR} = \phi$
5. **For** i=1to NP
6. Generate a random index $k_i = rand(1, M)$
7. Generate $F_i = randc_i(\mu F_{ki}, 0.1)$, $CR_i = randn_i(\mu CR_{ki}, 0.1)$
8. Generate $p_i = rand(0,1) \times N$, $N = NP \times 0.1$
9. Apply current-to-pbest/1 to generate trial vector $u_{i,g}$
10. **If** $f(u_{i,g}) \le f(x_{i,g})$
11. $x_{i,g+1} = u_{i,g}$
12. If $f(u_{i,g}) \neq f(x_{i,g})$
13. $F_i \to S_F, CR_i \to S_{CR}$
14. **Else**
15. $x_{i,g+1} = x_{i,g}$
16. **End**
17. Update memory M if $S_F \neq \phi, S_{CR} \neq \phi$
18. **If** $NP < NP_{\max}$
19. Increase P_g with a number of individuals equal to NP_{extra}
20. **For** h=1 to NP_{extra}
21. Generate new Individual x_h
22. Re-initialize $x_{h,j}$ if exceeds boundaries using Eq. 25
23. Add x_h to P_g
24. **End**
25. **End**
26. **If** $(NP \ge NP_{\max})$ and after saturation phase
27. Reduce P_g by $NP_{\min us}$ and eliminate worst individuals
28. **End**
29. **EndWhile**

Fig. 2. Pseudo-code of DEsPA algorithm

IV. Experimental Results

The performance of DEsPA algorithm is evaluated using a set of problems presented in the CEC2015 competition on learning-based single objective optimization [26]. This benchmark contains 15 test problems with different dimensions. The algorithm was coded using Java 1.8, and was run on a PC with 3.7GHz Core processor with 12 GB RAM and windows 8.1. 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 best solution found and optimal was 10^{-8}, the error was treated as 0. All parameter values are set as follows: the memory size M is set to 10, minimum size of population $NP_{\min}$ is set to 4 and maximum size $NP_{\max}$ is set to $15 \times D$. The rest of the parameters of the algorithms (F and CR) are updated as discussed in the previous section.

Results are presented in tables I-IV and are discussed in the following subsections. Each column shows best, worst, median, mean and standard deviation of the error value between the best fitness values found in each run and the true optimal value.

A. Results for 10D

The results of DEsPA algorithm is shown in Table I on D = 10. The table shows the best, worst, median, mean and standard deviation of error values for 51 independent runs. Referring to those results, the DEsPA algorithm was able to obtain the optimal solutions for unimodal functions (F_1, F_2). For multimodal functions, the algorithm was able to obtain the exact optimal for functions F_1, and F_2. DEsPA obtained good solutions for F_4, F_7, F_6, F_8 and F_{10}. For hybrid functions (F_6, F_7 and F_8), the DEsPA was robust and obtained better solutions than multimodal functions. The algorithm got trapped in local solutions in some of the composition functions.

TABLE I. STATISTICAL RESULTS OF THE 10-*D* BENCHMARK FUNCTIONS, AVERAGED OVER 51 INDEPENDENT RUNS

	Best	Worst	Median	Mean	Std
F_1	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
F_2	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
F_3	0.00E+00	2.00E+01	2.00E+01	1.67E+01	7.24E+00
F_4	0.00E+00	7.96E+00	3.03E+00	3.56E+00	1.30E+00
F_5	6.75E-01	2.52E+02	2.52E+01	5.19E+01	6.01E+01
F_6	2.08E-01	5.39E+00	1.41E+00	1.59E+00	1.21E+00
F_7	4.58E-02	1.18E+00	2.37E-01	3.42E-01	2.85E-01
F_8	2.19E-06	8.12E-01	4.48E-02	1.95E-01	2.22E-01
F_9	1.02E+02	2.01E+02	1.05E+02	1.06E+02	1.35E+01
F_{10}	6.90E+00	1.01E+01	7.48E+00	7.56E+00	3.96E-01
F_{11}	2.17E-01	2.00E+02	1.37E+00	9.46E+01	1.00E+02
F_{12}	1.00E+02	1.01E+02	1.01E+02	1.01E+02	2.84E-01
F_{13}	1.12E+01	2.32E+01	1.79E+01	1.77E+01	2.88E+00
F_{14}	1.00E+02	2.66E+03	1.00E+02	3.09E+02	6.95E+02
F_{15}	2.05E+02	2.05E+02	2.05E+02	2.05E+02	1.35E-03

B. Results for 30D

The results of DEsPA algorithm on D = 30 is presented in Table II. Same as 10D, DEsPA was able to obtain the optimal solutions for unimodal functions.

TABLE II. STATISTICAL RESULTS OF THE 30-*D* BENCHMARK FUNCTIONS, AVERAGED OVER 51 INDEPENDENT RUNS

	Best	Worst	Median	Mean	Std
F_1	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
F_2	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
F_3	2.00E+01	2.02E+01	2.01E+01	2.01E+01	4.36E-02
F_4	3.98E+00	1.79E+01	9.95E+00	9.71E+00	3.02E+00
F_5	9.48E+02	2.73E+03	1.79E+03	1.85E+03	3.97E+02
F_6	2.72E+01	4.34E+02	1.79E+02	1.61E+02	8.00E+01
F_7	1.07E+00	4.95E+00	3.21E+00	3.09E+00	7.41E-01
F_8	3.40E+00	1.52E+02	2.24E+01	2.55E+01	2.29E+01
F_9	1.16E+02	2.01E+02	2.01E+02	1.80E+02	3.60E+01
F_{10}	3.50E+01	3.38E+02	1.68E+02	1.71E+02	7.08E+01
F_{11}	2.01E+02	3.51E+02	3.38E+02	3.11E+02	5.52E+01
F_{12}	1.08E+02	1.09E+02	1.08E+02	1.08E+02	3.18E-01
F_{13}	6.93E+01	9.31E+01	8.18E+01	8.13E+01	5.60E+00
F_{14}	2.73E+04	3.23E+04	2.73E+04	2.81E+04	1.71E+03
F_{15}	2.73E+02	2.74E+02	2.73E+02	2.73E+02	1.50E-01

For multimodal functions, the performance of the algorithm degrades smoothly after increasing the dimensionality in F_3 and F_4 and clearly for F_5, F_6, F_{11}, and F_{14}. The same situation occurs for hybrid and composition functions except for F_{13} and F_{15} that shows good solutions comparable to 10D results. The best obtained values were the worst in cases of F11 (2.01E+02) and F14 (2.73E+04), compared to 2.17E-01 and 1.00E+02, for the same functions, respectively, when D=10.

C. Results for 50D

The results of DEsPA algorithm on D = 50 is presented in Table III. DEsPA was able to obtain the optimal solution for the unimodal function F_2. On the other hand, the performance of DEsPA algorithm degrades for F_1 with a value equal to 9.50E+01. For multimodal functions, the performance of the algorithm is almost the same order as that for 30D results. For the hybrid and composition functions, the performance of the DEsPA algorithm degrades not sharply compared to the 30D results. The best obtained results were not as good as in 30D for most of the functions, especially for F5 (2.68E+03) and F6 (1.77E+02).

TABLE III. STATISTICAL RESULTS OF THE 50-*D* BENCHMARK FUNCTIONS, AVERAGED OVER 51 INDEPENDENT RUNS

	Best	Worst	Median	Mean	Std
F_1	3.36E-01	1.20E+03	3.25E+01	9.50E+01	2.09E+02
F_2	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
F_3	2.00E+01	2.03E+01	2.01E+01	2.01E+01	9.63E-02
F_4	6.96E+00	2.98E+01	1.59E+01	1.69E+01	4.31E+00
F_5	2.68E+03	7.16E+03	4.08E+03	4.25E+03	8.50E+02
F_6	1.77E+02	1.74E+03	9.75E+02	9.44E+02	3.25E+02
F_7	3.95E+01	4.16E+01	4.07E+01	4.06E+01	4.94E-01
F_8	5.08E+01	6.75E+02	2.40E+02	2.93E+02	1.67E+02
F_9	1.18E+02	2.02E+02	2.02E+02	1.90E+02	2.93E+01
F_{10}	3.33E+02	8.15E+02	5.74E+02	5.82E+02	1.09E+02
F_{11}	3.06E+02	4.84E+02	3.74E+02	3.78E+02	4.07E+01
F_{12}	1.07E+02	1.09E+02	1.08E+02	1.08E+02	3.48E-01
F_{13}	1.29E+02	1.83E+02	1.47E+02	1.48E+02	1.13E+01
F_{14}	2.98E+04	5.53E+04	2.98E+04	3.13E+04	5.90E+03
F_{15}	2.83E+02	2.84E+02	2.84E+02	2.84E+02	1.91E-01

D. Results for 100D

The results of DEsPA algorithm on D = 100 is presented in Table IV. Comparing with 50D results, the unimodal functions show degraded performance especially for F_1. The reason behind this degradation is the complex nature of F_1 which differ from the other functions by its quadratic ill-conditioned feature. For the multimodal function F_3, the DEsPA algorithm shows stable performance. The algorithm did not show a sharp decrease in the solutions for functions F_4 and F_5 compared to those obtained for 50D results. For the hybrid function F_7, the DEsPA algorithm shows the same performance comparing to 50D result and a lower performance for F_6 and F_8. The DEsPA

algorithm behaves the same for composition functions on 50D except for F_{10}.

TABLE IV. STATISTICAL RESULTS OF THE 100-D BENCHMARK FUNCTIONS, AVERAGED OVER 51 INDEPENDENT RUNS

	Best	Worst	Median	Mean	Std
F_1	1.85E+05	8.05E+05	4.51E+05	4.60E+05	1.34E+05
F_2	2.42E-01	9.38E+02	8.14E+00	3.07E+01	1.30E+02
F_3	2.00E+01	2.08E+01	2.04E+01	2.03E+01	2.12E-01
F_4	3.18E+01	6.17E+01	4.48E+01	4.51E+01	7.08E+00
F_5	8.97E+03	1.52E+04	1.27E+04	1.25E+04	1.60E+03
F_6	2.46E+03	6.00E+03	4.34E+03	4.35E+03	8.13E+02
F_7	4.99E+01	6.23E+01	5.40E+01	5.46E+01	3.05E+00
F_8	5.80E+02	2.68E+03	1.43E+03	1.45E+03	3.88E+02
F_9	2.05E+02	2.07E+02	2.06E+02	2.06E+02	2.71E-01
F_{10}	2.04E+03	3.83E+03	2.97E+03	2.90E+03	4.12E+02
F_{11}	6.33E+02	1.06E+03	8.58E+02	8.56E+02	9.23E+01
F_{12}	1.16E+02	1.18E+02	1.17E+02	1.17E+02	4.51E-01
F_{13}	3.41E+02	3.95E+02	3.63E+02	3.61E+02	1.13E+01
F_{14}	8.73E+04	8.85E+04	8.73E+04	8.74E+04	2.64E+02
F_{15}	3.00E+02	3.04E+02	3.01E+02	3.01E+02	8.88E-01

E. Algorithm Complexity

The computational complexity of DEsPA algorithm is calculated based on problem dimensions D = 10, 30, 50 and 100 as described in [26]. A summary of results is presented in Table V. In this table, T_0 represents the execution time of the following program:

for $i = 1,000,000$

$x = 0.55 + (double)i;$

$x = x + x; x = x / 2; x = sqrt(x);$

$x = \log(x); x = \exp(x); x = x / (x + 2);$

end

T_1 denotes the computing time of F_1 for 200,000 function evaluations. Assume T_2 is the complete running time of DEsPA algorithm for F_1 using 200,000 function evaluations. $\hat{T}_2$ is the mean value of the five evaluated T_2. Finally, the algorithm complexity is computed by $(\hat{T}_2 - T_1) / T_0$.

TABLE V. ALGORITHM COMPLEXITY

	T_0	T_1	$\hat{T}_2$	$(\hat{T}_2 - T_1) / T_0$
$D = 10$	0.1210	0.3320	0.4344	0.8463
$D = 30$		1.2650	12.9122	96.2579
$D = 50$		2.6860	22.9400	167.3884
$D = 100$		8.4470	52.3218	362.6017

V. CONCLUSION

This paper proposes DEsPA algorithm which is an enhanced version of JADE algorithm. This algorithm aims at improving the performance by adding three main features. Firstly, the greediness factor of mutation strategy current-to-*p*best is adapted in a way to balance exploration and exploitation capabilities. The second feature is to use a memory-based scheme with an adapted size for better guiding the adaptation process of the scaling factor and crossover rates. The third main feature is introducing a novel way for adapting the population size of a DE variant algorithm. The population is resized in both direction: proliferation and reduction in a linear manner. This way helps to increase the diversity of population at each optimization loop and also improves the convergence rate. The algorithm is evaluated on CEC2015 competition on learning-based single objective optimization and showed a good performance in most of the suggested functions.

These functions will be evaluated and the results will be compared with other state-of-the-art algorithms and other competing algorithms from this competition in a future version of this work.

REFERENCES

[1] K. Deb, Multi-Objective Optimization Using Evolutionary Algorithms, Wiley, New York, 2001

[2] L. Fogel, A.J.Owens, M.J. Walsh, Artificial intelligence through simulated evolution, Wiley. ISBN0-471-26516-0, 1966

[3] J.R. Kosa, Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems. Rep No. STAN-CS-90-1314, Stanford University, CA, 1990

[4] I. Rechenberg, Cybernetic solution path of an experimental problem. Royal Aircraft Establishment Library Translation, 1122, 1965

[5] F. Glover, "Heuristic for integer programming using surrogate constraints," Decision Sci, vol. 8, no. 1, pp. 156-166, 1977

[6] M. Krug, S. K. Nguang, J. Wu and J. Shen, "GA-based predictive control of boiler-turbine systems," International Journal of Inoovative Computing, Information and Control, vol. 6, no. 11, pp. 5237-5248, 2010.

[7] J.H. Holland, Adaption in Natural and Artificial Systems, University of Michigan Press, Ann Arbor, MI, 1975

[8] S. Kirkpatrick, C. Gelatt, M. Vecchi, "Optimization by simulated annealing," Science, vol. 220, no. 4598, pp. 671-680, 1983

[9] J. Kennedy, R. Eberhart, "Particle swarm optimization," In Proc. of IEEE International Conference on Neural Networks, vol. 4, pp. 1942–1948, 1995.

[10] C.-J. Lin, J.-G. Wang and S.-M. Chen, "2D/3D face recongnition using neural network based on hybrid Tahuchi-particle swarm optimization," International Journal of Inoovative Computing, Information and Control, vol. 7, no. 2, pp. 537-553, 2011.

[11] X. Cai, Z. Cui, J. Zeng and Y. Tan, "Particle swarm optimization with self-adjusting congitive selection strategy," International Journal of Inoovative Computing, Information and Control, vol. 4, no. 4, pp. 943-952, 2008.

[12] M. Dorigo, V. Maniezzo, A. Colorni, "Ant System: Optimization by a Colony of Cooperating Agents," IEEE Transactions on Systems, Man, and Cybernetics–Part B, vol. 26, no. 1, pp. 29–41, 1996

[13] 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.

[14] 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
[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. Evolut. Comput., vol. 10, no. 6, pp. 646–657, 2006
[16] S. Das, A. Abraham, U.K. Chakraborty, A. Konar, "Differential evolution using a neighborhood based mutation operator," IEEE Trans on Evolutionary Computation, vol. 13, no. 3, pp. 526-553, 2009
[17] K.V. Price, "An Introduction to Differential Evolution. New Ideas in Optimization," McGraw Hill, London (UK), pp. 79-108, 1999
[18] S. Rahnamayan, H.R. Tizhoosh, M.M. Salama, "Opposition-Based Differential Evolution," IEEE Trans on Evolutionary Computation, vol. 12, no. 1, pp. 64-79, 2008
[19] K. Qin, P.N. Suganthan, "Self-adaptive differential evolution algorithm for numerical optimization," In Proc. IEEE Congr Evol Comput, pp. 1785–1791, 2005
[20] 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, 2006
[21] R. Mallipeddi, P. N. Suganthan, Q. K. Pan, and M. F. Tasgetiren, "Differential evolution algorithm with ensemble of parameters and mutation strategies," Applied Soft Computing, vol. 11, no. 2, pp. 1679–1696, 2011.
[22] 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
[23] R. Tanabe and A. Fukunaga, "Success-History Based Parameter Adaptation for Differential Evolution," in IEEE CEC, 2013, pp. 71–78.
[24] R. Tanabe and A. Fukunaga, "Improving the Search Performance of SHADE Using Linear Population Size Reduction," in IEEE CEC, 2014, pp. 1658 - 1665.
[25] 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.
[26] J. J. Liang, B. Y. Qu, and P. N. Suganthan, "Problem Definitions and Evaluation Criteria for the CEC 2015 Competition on Learning-based Real-Parameter Single Objective Optimization," Zhengzhou University and Nanyang Technological University, Tech. Rep., 2015.