DE GRUYTER OPEN

JAISCR, 2016, Vol. 6, No. 2, pp. 103 – 118
10.1515/jaiscr-2016-0009

# ADAPTING DIFFERENTIAL EVOLUTION ALGORITHMS FOR CONTINUOUS OPTIMIZATION VIA GREEDY ADJUSTMENT OF CONTROL PARAMETERS

Miguel Leon, Ning Xiong

*School of Innovation, Design and Engineering*
*Malardalen University, Vasteras, Sweden*

**Abstract**

Differential evolution (DE) presents a class of evolutionary and meta-heuristic techniques that have been applied successfully to solve many real-world problems. However, the performance of DE is significantly influenced by its control parameters such as scaling factor and crossover probability. This paper proposes a new adaptive DE algorithm by greedy adjustment of the control parameters during the running of DE. The basic idea is to perform greedy search for better parameter assignments in successive learning periods in the whole evolutionary process. Within each learning period, the current parameter assignment and its neighboring assignments are tested (used) in a number of times to acquire a reliable assessment of their suitability in the stochastic environment with DE operations. Subsequently the current assignment is updated with the best candidate identified from the neighborhood and the search then moves on to the next learning period. This greedy parameter adjustment method has been incorporated into basic DE, leading to a new DE algorithm termed as Greedy Adaptive Differential Evolution (GADE). GADE has been tested on 25 benchmark functions in comparison with five other DE variants. The results of evaluation demonstrate that GADE is strongly competitive: it obtained the best rank among the counterparts in terms of the summation of relative errors across the benchmark functions with a high dimensionality.

**Keywords:** Differential Evolution, Optimization, Parameter Adaptation.

## 1 Introduction

Evolutionary algorithms (EAs) are biologically inspired metaheuristics that provide powerful and robust means to solve complex and high dimensional optimization problems in real world [1]. One important advantage of EAs over classic optimization techniques is that they dont use derivative information of objective functions such that they can be widely applied in situations where the problem space is not differentiable or continuous. Many variants of EAs have been developed to deal with real-parameter continuous optimization problems, including evolution strategies [2], real-coded genetic algorithms [3], memetic algorithms [4], differential evolution (DE) [5], particle swarm optimization [6], and artificial bee colony algorithms [7].

Differential evolution (DE) shares common concepts of EAs and it also relies on recombination and mutation operators to produce new individuals in the population. Nevertheless, DE differs from other EAs in that mutation in DE is based on differences of individuals selected from the population. Thus, the direction and magnitude of the search is decided by the distribution of solutions rather than a pre-specified probability function. DE attains increasing popularity due to its at-

tractive features such as fewer running parameters, easy in programming, high efficiency, as well as strong global search ability. In [8] it was indicated that DE algorithms were more efficient and more accurate than several other optimization methods, including controlled random search, simulated annealing and genetic algorithms. Moreover, DE has achieved quite high rank in the competition held in the IEEE Congress on Evolutionary Computation [9]. A comprehensive review of the state-of-the-art of DE algorithms is given in [10].

Unfortunately, the performance of DE is not always excellent. It can easily get stuck in a local optimum or stop generating progressively better solutions before the population has converged. One reason of such failures can be poorly assigned control parameters like scaling factor and crossover probability [11], [12], [13]. It has also been observed that, when DE explores different regions of the space, it would require using different parameter values to maintain the efficiency of search.

On-line adaptation of control parameters for DE appears a promising research direction that has been addressed by a number of researchers. The purpose is to dynamically adjust the parameters of DE using feedback from the search. Liu and Lampinen [14] proposed the use of fuzzy logic controllers to modify the scaling factor and crossover probability based on progresses in the search. Xue and Sanderson [15] independently developed a similar method based on heuristic knowledge in form of fuzzy rules for parameter adaptation in a multi-objective DE algorithm. However, the fuzzy rule-based heuristic methods may be over-tailored to a specific problem without generality to wide applications. The works published in [16], [17], [18], [19] can be classified as statistical approaches to parameter adaptation. They rely on certain probability distributions (such as Gaussian and Cauchy density functions) to generate scaling factor and crossover rate for individual vectors in the population, and those parameter values that succeeded in producing trial solutions surviving in the next generation are utilized to update the locations of the distribution functions for improved DE performance. But the stochastic nature of mutation and crossover operators are not taken into account in these approaches in judging successful control parameter values.

This paper proposes a new adaptive DE algorithm by greedy adjustment of the control parameters (scaling factor and crossover rate) during the running of DE. The basic idea is to perform greedy search for better parameter assignments in successive learning periods in the whole evolutionary process. Within each learning period, the current parameter assignment and its neighboring assignments are tested (used) in a number of times to acquire a reliable assessment of their suitability in the stochastic environment with DE operations. Subsequently, the current assignment is updated with the best candidate identified from the neighborhood and then the search moves on to the next learning period. As greedy adjustments of parameters are conducted repeatedly from one period to the next, we achieve continuous adaptation of DE behavior along the course of search. The proposed DE algorithm has been examined and compared with other five DE variants in the experiments. The results of evaluation on a set of 25 benchmark functions demonstrate that our method is rather competitive: it obtained the best rank among the counterparts according to the summation of relative errors across the benchmark functions with high dimensionality.

The rest of the paper is organized as follows. The relevant works are discussed in Section 2. Section 3 introduces the basic DE algorithm, which is followed by the presentation of the adaptive DE with greedy parameter adjustment in Section 4. The results of evaluation are given in Section 5. Finally, we convey the concluding remarks in Section 6.

## 2 Relevant Works

It is well known that the performance of DE is heavily dependent on the setting of control parameters such as the mutation factor and the crossover probability. This section reviews some of the methods for dynamic adjustment of control parameters within the most well known adaptive DE algorithms. Although these adaptive algorithms also include self-adaptive or certain advanced mutation strategies, this section only focuses on the part of parameter adaptation for DE.

One of the first adaptive DE algorithms is called SaDE [16]. It was proposed by Qin and Suganthan in 2005. This algorithm does not have fixed values for the scaling factor and crossover probability

during its execution. The scaling factor for every individual is created following a fixed normal distribution. The crossover rate (CR) follows a normal distribution too, but its center is updated using the median of the successful CR values during last 25 generations.

JADE [17] was proposed by Zhang and Sanderson in 2009. Similarly to SaDE, JADE creates random values for both scaling factor and crossover rate. The generation of F follows a Cauchy distribution while CR is created according to a Normal distribution. The center of the Cauchy distribution is updated by a weighted sum of the actual value and the Lehmer mean of the successful F values in the last generation. Likewise, the center of the Normal distribution is revised whereas with the arithmetic mean of the successful CR values.

The method MDE_pBX [18], proposed by Minzahul Islam et al., is similar to JADE in two aspects. First, it employs a Cauchy distribution to generate F values and a Normal distribution to create CR values. Second, it also actualizes the centers of the probability distributions after every generation. The difference between the two algorithms lies in the way in which the centers are revised: MDE_pBX uses a Power mean rather than a Lehmer mean or arithmetic mean as in JADE.

SHADE [20] is a success-history based DE algorithm proposed by Tanabe and Fukunaga in 2013. It maintains two memories, one for scaling factor and the other crossover rate. The entities in both memories are randomly selected as the center for the Cauchy distribution for generating F values and the center of the Normal distribution for generating CR values respectively. Further, one entity in each memory will be replaced by some mean of successful F or CR values after each generation in a cyclical manner.

## 3 Basic DE

DE is a stochastic and population based algorithm for optimization. A population in DE consists of a set of individuals, each of which stands for a possible solution to the problem. Optimization is conducted by evolving the population progressively from one generation to another. Here we use $X_{i,g}$ to denote individual $i$ in the population at generation $g$, with $i = 1, 2, \ldots, N_P$ and $N_P$ being the population size (the number of solutions in the population). DE has three consecutive steps in every iteration: mutation, crossover and selection. The explanation of these steps is given below:

MUTATION. At each generation $g$, $N_P$ mutated individuals are generated based on the current parent population. The vector for the mutated solution is called mutant or donor vector and it is represented by $V_{i,g}$. There are several alternative ways to mutate an individual in the current population. A mutation strategy is often notated as DE\$x\$y, where $x$ stands for the vector to be mutated and $y$ represents the number of difference vectors used in the mutation. The following are three basic mutation strategies frequently used in the literature:

– DE/rand/1:

$$V_{i,g} = X_{r_1,g} + F \times (X_{r_2,g} - X_{r_3,g}) \tag{1}$$

– DE/best/1:

$$V_{i,g} = X_{best,g} + F \times (X_{r_1,g} - X_{r_2,g}) \tag{2}$$

– DE/current-to-best/1

$$\begin{aligned} V_{i,g} &= X_{i,g} + F_1 \times (X_{best,g} - X_{i,g}) + \\ &+ F_2 \times (X_{r_1,g} - X_{r_2,g}) \end{aligned} \tag{3}$$

where $r_1, r_2, r_3$ are mutually exclusive integers randomly selected from 1 to $N_P$, The scaling factor $F$ is a real positive control parameter which lies in the interval (0,2]. $X_{best,g}$ represents the best solution from the population at generation $g$. Other mutation strategies and their performance are discussed in [21].

As can be seen from above, the values in the mutant vector may violate predefined boundary constraints. To solve this issue we repair $V_{i,g}$ according to Equation 4.

$$V_{i,g}[j] = \begin{cases} (Low[j] & \text{if } V_{i,g}[j] < Low[j], \\ (Upper[j] & \text{if } V_{i,g}[j] > Upper[j]. \end{cases} \tag{4}$$

where $X_{i,g}[j]$ denotes the $j$th component of vector $X_{i,g}$ and $V_{i,g}[j]$ denotes the $j$th component of the mutant vector $V_{i,g}$.

CROSSOVER. In the second step we recombine the set of mutated solutions created in the first step (mutation) with the set of original members in

the population to produce trial solutions. The new trial vector is denoted by $T_{i,g}$ where $i$ is the index of the corresponding parent member. Every parameter in the trial vector is derived according to Equation 5 if the binomial crossover method is used.

$$T_{i,g}[j] = \begin{cases} V_{i,g}[j] & \text{if } rand[0,1] <= CR \text{ or} \\ & j = j_{rand} \\ X_{i,g}[j] & \text{otherwise} \end{cases} \quad (5)$$

where $j$ represents the parameter index in a vector, $CR$ is the probability of recombination and $j_{rand}$ is an integer randomly selected from $\{1,2,\ldots,N_P\}$ to ensure that at least one parameter from the mutant vector is selected.

SELECTION. In this last step we compare a trial vector with its parent member in the population, to choose the stronger one to enter the next generation. Therefore, if the problem to address is a minimization problem, each individual in the next generation is created according to Equation 6.

$$X_{i,g+1} = \begin{cases} T_{i,g} & \text{if } f(T_{i,g}) < f(X_{i,g}) \\ X_{i,g} & \text{otherwise} \end{cases} \quad (6)$$

where $X_{i,g}$ represents a parent individual in the population, $X_{i,g+1}$ is the individual in the next generation, and $f(T_{i,g})$ and $f(X_{i,g})$ stand for the fitness values of vectors $T_{i,g}$ and $X_{i,g}$ respectively.

# 4 Adaptive DE with Greedy Parameter Adjustment

As is known the DE control parameters such as mutation Factor (F) and Crossover Rate (CR) are largely problem dependent, i.e. solving different problems may need different parameter values to ensure good performance. Further, the proper values of DE parameters often change with time in the evolutionary process. Hence it is important to automatically determine and adjust such parameters for DE when solving a practical problem. To this end we propose a new adaptive DE algorithm that dynamically adjusts its control parameters using greedy (local) search. In this section we shall first present the greedy scheme for parameter adjustment in Subsection 4.1 and then we will discuss the integration of this scheme within a DE cycle in Subsection 4.2.

## 4.1 Greedy Search for Parameter Adaptation

Our basic idea is to perform local greedy search to adjust the values of control parameters (scaling factor and crossover probability) of DE to improve its performance. This means that at every step the current parameter assignment is compared with its neighbours and then moves to the best candidate in the neighbourhood. Nevertheless, the comparison of different DE parameters is not a trivial task. It is complicated by the stochastic characteristics of the mutation and crossover operators such that a good parameter assignment may also lead to undesired trial solutions created in the course of search.

It is advocated in the paper that a candidate for parameter assignment undergoing sufficient tests for reliable evaluation of its quality. The tests are made in a learning period comprising a specified number of generations to see how the candidate was useful to contribute to the creation of good trial solutions. We desire those parameter assignments that not only offer a high chance of survival for trial solutions but also enable substantial improvement of fitness in the next generation. In view of this, the relative improvement (RI) brought by a candidate assignment $C$ (for either scaling factor or crossover probability) in test k is defined as:

$$RI(C,j) = \begin{cases} f(X^k) * 10^n - f(V^k) * 10^n, & \text{if} f(X^k) \geq f(V^k), \\ 0, & \text{otherwise} \end{cases} \quad (7)$$

where $X^k$ and $V^k$ represent respectively the parent and trial solutions in test $k$, and $n$ is an integer such that $f(X^k) * 10^n$ lies in the interval [1,10] or [-10,-1]. Further, the progress rate (PR) for $C$ is the average of the relative improvements from all the m tests of using $C$ for producing trial solutions. Thus we can write

$$PR(C) = \frac{1}{N} \sum_{j=1}^{N} RI(C,j) \quad (8)$$

progress rate is used in this paper as the criterion to evaluate and compare candidates for DE parameter assignments.

In the greedy search procedure, the current parameter assignment and its two generated neighbours are randomly selected for being used in producing new trial solutions during the learning pe-

riod. The best of them is then identified using the metric of progress rate as defined in 8. As proper values of control parameters can change over time, we perform life-long search from one learning period to the next to achieve continuous adjustment of parameters in the course of optimization. An algorithmic description of the greedy scheme for parameter adjustment is given in the following:

The greedy search for parameter adjustment:

1. $F_0 \leftarrow$ initial assignment for scaling factor.
2. $P_0 \leftarrow$ initial assignment for crossover probability.
3. Expand $F_0$: Creating its two neighbours $F_1$ and $F_2$.
4. Expand $P_0$: Creating its two neighbours $P_1$ and $P_2$.
5. count(x)=0, count(y)=0, sum(x)=0 and sum(y)=0 for all $x \in \{F_0, F_1, F_2\}, y \in \{P_0, P_1, P_2\}$
6. i = 0
7. **while** $(i \leq LP \times N_P)$ % LP is the number of generations in the learning period
8. Randomly select $x^* from \{F_0, F_1, F_2\}$
9. Randomly select $y^* from \{P_0, P_1, P_2\}$
10. Perform mutation and crossover using $x^*$ and $y^*$.
11. Derive RI($x^*$) and RI($y^*$) upon the trial and parent solutions
12. count($x^*$) = count($x^*$)+1 and count($y^*$) = count($y^*$)+1
13. sum($x^*$) = sum($x^*$) + RI($x^*$) and sum($y^*$) = sum($y^*$) + RI($y^*$)
14. i = i + 1;
15. **end while**
16. PR($x$)=sum($x$)/count($x$) for all $x \in \{F_0, F_1, F_2\}$
17. PR($y$)=sum($y$)/count($y$) for all $y \in \{P_0, P_1, P_2\}$
18. $F_0 = arg \max_{x \in \{F_0, F_1, F_2\}} PR(x)$;
19. $P_0 = arg \max_{y \in \{P_0, P_1, P_2\}} PR(y)$;
20. Go to step 3

## 4.2 Greedy Adaptive DE Algorithm

The Greedy Adaptive Differential Evolution (GADE) algorithm is developed by incorporation of the greedy search mechanism into the basic DE algorithm. The whole evolutionary process is divided into a sequence of learning periods and every learning period consists of a fixed number of generations. The greedy search is performed in successive learning periods to facilitate continuous and dynamic adjustment of F and CR values during the execution of the algorithm.

The initial candidate for mutation factor is set as $F = 0.5$, and its two neighbors are $F + d_1$ and $F - d_1$ respectively, where $d_1$ is a user specified small positive number. The initial candidate for crossover rate is a Cauchy distribution with its center $CR_m = 0.5$ and its scale parameter equal to 0.2. The two neighbors of this current distribution are the shifted Cauchy distributions with their centers being located at $CR_m + d_2$ and $CR_m - d_2$ respectively, where $d_2$ is a small positive number specified by user. Every current and neighboring candidate (for both mutation factor and crossover rate) receives a probability of 1/3 to be associated with an individual vector in the population in order to get a sufficient number of usages in the learning period. At the end of the learning period, a neighboring candidate may replace the current one according to the assessed progress rates.

A more detailed description of our GADE algorithm is given in the pseudocode of Algorithm 1. Although the simple random mutation strategy is used in the present version of the algorithm, the principle and mechanism proposed in this paper is generic and can be easily applied with other mutation strategies as well.

**Algorithm 1**: GADE

1. Set $CR_m = 0.5, F = 0.5, LP = 20, d_1 = d_2 = 0.01$;
2. $Z_F = \{F - d_1, F, F + d_1\}$;
3. $Z_{CR} = \{CR_m - d_2, CR_m, CR_m + d_2\}$;
4. $g = 1$;
5. Initialize the population $(X_{1,1}, X_{2,1}, \ldots, X_{N_p,1})$

6. **WHILE** The termination condition is not satisfied
7. **FOR** $i = 1\ N_P$
8. Set $F_i$ by randomly selecting one element from $Z_F$.
9. Set $\mu_{CR}$ by randomly selecting one element from $Z_{CR}$.
10. $CR_i = Cauchy(\mu_{CR}, 0.2)$.
11. Create the mutant vector using the random mutation strategy by:
$V_{i,g} = X_{r_1,g} + F \times (X_{r_2,g} - X_{r_3,g})$
12. Repair the mutant vector if it has values outside the boundaries:
$V_{i,G}[j] = \begin{cases} (Low[j] & \text{if } V_{i,g}[j] < Low[j], \\ (Upper[j] & \text{if } V_{i,g}[j] > Upper[j]. \end{cases}$
13. Create the trial vector:
$T_{i,G}[j] = \begin{cases} V_{i,g}[j] & \text{if } rand[0,1] \leq CR \text{ or } j = j_{rand} \\ X_{i,g}[j] & \text{otherwise} \end{cases}$
14. **IF** $f(T_{i,g}) < f(X_{i,g})$
15. $X_{i,g+1} = T_{i,g}$
16. **ELSE**
17. $X_{i,g+1} = X_{i,g}$
18. **ENDIF**
19. **ENDFOR**
20. /* Update F*/
21. **IF** $G\%LP == 0$
22. $F = arg\max_{x \in Z_F} PR(x)$;
23. $Z_F = \{F - d_1, F, F + d_1\}$;
24. $CR_m = arg\max_{y \in Z_{CR}} PR(y)$;
25. $Z_{CR} = \{CR_m - d_2, CR_m, CR_m + d_2\}$;
26. **ENDIF**
27. $g = g + 1$;
28. **ENDWHILE**

# 5 Experiments and Evaluation

This section examines the adaptation capability of our GADE algorithm in comparison with other relevant algorithms. The tests were made on 25 benchmark functions from [22] and [23] with dimension (D) 10 and 30. A complete description of these functions is given in Table I, where Functions f1-f7 and Functions f14-f18 are unimodal, and Functions f8-f13 and Functions f19-f25 are multimodal. In D10, only Functions f14-f25 were used while in D30 experiments were made on all the 25 functions.

## 5.1 Experimental Settings

Beside GADE, the basic DE (DE/rand/1) and the methods from four other DE variants: SaDE, JADE, SHADE and MDE_pBX were also tested in the experiments for comparison. As our purpose was to compare different parameter adaptation methods, mutation strategy adaptation was not implemented in our experiments. Here we use the suffix "-P" to denote the parameter adaptation method from the original adaptive DE algorithm that was incorporated into the basic DE. All the algorithms in comparison employed the binomial crossover operator and the rand/1 mutation strategy. The settings of these algorithms are listed below:

- DE/rand/1: population size $N_P = 60$, $F = 0.9$ and $CR = 0.9$
- SaDE-P: population size $N_P = 60$, initial $CR_m = 0.5$ and length of learning period $LP = 20$
- JADE-P: population size $N_P = 60$, initial $\mu F = 0.5$, initial $\mu CR = 0.5$ and $c = 0.01$
- SHADE-P: population size $N_P = 60$, memory size $H = 60$, initial $M_F = \{0.5,\ldots,0.5\}$ and initial $M_{CR} = \{0.5,\ldots,0.5\}$
- MDE_pBX-P: population size $N_P = 60$, initial $F_m = 0.5$ and initial $CR_m = 0.6$
- GADE: population size $N_P = 60$, initial $F = 0.5$, initial $CR_m = 0.5$, $d_1 = d_2 = 0.01$, and length of the learning period $LP_F = LP_{CR} = 20$

**Table 1**. The 25 benchmark functions used in the experiments

| FUNCTION | NAME |
|---|---|
| f1(x) = $\sum_{i=1}^{n} x_i^2$ | Sphere |
| f2(x) = $\sum_{i=1}^{n} \lvert x_i\rvert + \prod_{i=1}^{n} \lvert x_i\rvert$ | Schwefel 2.22 |
| f3(x) = $\sum_{i=1}^{n} (\sum_{j=1}^{i} x_j)^2$ | Schwefel 1.2 |
| f4(x) = $max_i\{\lvert x_i\rvert, 1 \le i \le n\}$ | Schewefel 2,21 |
| f5(x) = $\sum_{i=1}^{n-1} [100 \times (x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$ | rosenbrock |
| f6(x) = $\sum_{i=1}^{n} *(x_i + 0.5)^2$ | step |
| f7(x) = $\sum_{i=1}^{n} i \times x_i^4 + random[0,1)$ | Noisy Quartic |
| f8(x) = $\sum_{i=1}^{n} -x_i \times sin(\sqrt{\lvert x_i\rvert})$ | Schwefel 2.26 |
| f9(x) = $\sum_{i=1}^{n} [x_i^2 - 10 \times cos(2 \times \pi \times x_i) + 10]$ | Rastrigin |
| f10(x) = $-20 \times exp(-0.2 \times \sqrt{\frac{1}{n} \times \sum_{i=1}^{n} x_i^2}) - exp(\frac{1}{n} \times \sum_{i=1}^{n} cos(2\pi x_i)) + 20 + e$ | Ackley |
| f11(x) = $\frac{1}{4000} \times \sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} cos(\frac{x_i}{\sqrt{i}}) + 1$ | Griewank |
| f12(x) = $\frac{\pi}{n} \times \{10sin^2(\pi y_i) + \sum_{i=1}^{n-1}((y_i - 1)^2[1 + 10sin^2(\pi y_{i+1})]) + (y_n - 1)^2\} +$ $+ \sum_{i=1}^{n} u(x_i, 10, 100, 4)$, where $y_i = 1 + \frac{1}{4}(x_i + 1)$ $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & x_i > a \\ 0, & -a \le x_i \le a \\ k(x_i - a)^m, & x_i < -a \end{cases}$ | |
| f13(x) = $0.1 \times \{sin^2(3\pi x_1) + \sum_{i=1}^{n-1}((x_i - 1)^2[1 + sin^2(3\pi x_{i+1})]) +$ $+ (x_n - 1)[1 + sin(2\pi x_n)^2]\} + \sum_{i=1}^{n} u(x_i, 5, 100, 4)$ | |
| f14(x) = $\sum_{i=1}^{n} z_i^2; z = x - o$ | Shifted Sphere |
| f15(x) = $\sum_{i=1}^{n} (\sum_{j=1}^{i} z_j)^2; z = x - o$ | Shifted Schwefel 1.2 |
| f16(x) = $\sum_{i=1}^{n} (10^6)^{\frac{i-1}{n-1}} z_i^2; z = (x - o) * M$ | Shifted Rotated High Conditioned Elliptic |
| f17(x) = $(\sum_{i=1}^{n} (\sum_{j=1}^{i} z_j)^2) \times (1 + 0.4\lvert N(0,1)\rvert); z = x - o$ | Shifted Schwefel 1.2 with Noise in Fitness |
| f18(x) = $max\{A_i x - B_i\}$; *check [23] | Shifted Schwefel 2.6 with global optimum on Bounds |
| f19(x) = $\sum_{i=1}^{n-1} [100 \times (-z_{i+1} + z_i^2)^2 + (z_i - 1)^2]; z = x - o + 1$ | Shifted Rosenbrock |
| f20(x) = $\frac{1}{4000} \times \sum_{i=1}^{n} z_i^2 - \prod_{i=1}^{n} cos(\frac{z_i}{\sqrt{i}}) +; z = (x - o) * M$ | Shifted Rotated Griewank without bounds |
| f21(x) = $-20 \times exp(-0.2 \times \sqrt{\frac{1}{n} \times \sum_{i=1}^{n} z_i^2}) - exp(\frac{1}{n} \times \sum_{i=1}^{n} cos(2\pi z_i)) + 20 + e$ $z = (x - o) * M$ | Shifted Rotated Ackley with global optimum on bounds |
| f22(x) = $\sum_{i=1}^{n} [z_i^2 - 10 \times cos(2 \times \pi \times z_i) + 10]; z = x - o$ | Shifted Rastrigin |
| f23(x) = $\sum_{i=1}^{n} [z_i^2 - 10 \times cos(2 \times \pi \times z_i) + 10]; z = (x - o) * M$ | Shifted Rotated Rastrigin |
| f24(x) = $\sum_{i=1}^{n} (\sum_{k=0}^{K_{max}} [a^k cos(2\pi b^k (x_i + 0.5))]) - N \sum_{k=0}^{k_{max}} [a^k cos(2\pi b^k \times 0.5)]$ where $a = 0.5, b = 3, K_{max} = 20; z = (x - o) * M$ | Shifted Rotated Weierstrss |
| f25(x) = $\sum_{j=1}^{n} (A_i - B_i(x))^2$ *check [23] | Schwefel 2.13 |

All the algorithms were tested on the benchmark functions in Table 1 for comparative evaluation of their performance. Every algorithm was executed 30 times on each test function to acquire a fair and reliable result for comparison. The termination condition is that the fitness evaluation number has exceeded 10,000 x D or the error of the best solution found is below 10E-08 with respect to the true global optimum.

### 5.2 Comparison of GADE with other algorithms on dimension 10

This subsection aims to evaluate GADE with respect to other algorithms on problems with dimension 10. Since all the algorithms performed perfect on functions f1-f13 under this low dimension, we focus on comparing the performance of them in the remaining functions (f14-f25) with the results being indicated in Table 2, where the values in boldface represent the mean errors of solutions found by the algorithms and the values in brackets are the standard deviations.

We can observe from Table 2 that GADE obtained the best results on 4 unimodal functions (f14, f15, f17, f18) and it was the third best on unimodal function f16. In the multimodal case SHADE-P outperformed the others in 3 of the 7 functions, MDE_pBX-P was superior in 2 functions and GADE got the best average performance in function f25.

Further we do comparison of GADE with every other algorithm in terms of their errors obtained on the 12 test functions. The numbers of functions on which GADE was superior ($\succ$), identical ($=$) and inferior ($\prec$) to its counterpart are given in Table 3 respectively. It can be seen from the table that GADE was superior or identical to SaDE-P, JADE-P, MDE_pBX-P and basic DE in at least 10 of the 12 test functions. Moreover, GADE behaved better than or identically to SHADE-P in 4 of the 5 unimodal functions, whereas its performance was less good in comparison to SHADE-P in 5 of the 7 multimodal functions.

Table 4 shows a ranking of the algorithms in terms of the summation of their relative errors across the 12 test functions. The relative error of an algorithm on a certain function is defined as the ratio of the (mean) error of the algorithm on that function to the worst error on the function among all the algorithms. Here we see that GADE is ranked as the second best according to its relative performance on the problems with the low dimensionality.

### 5.3 Comparison of GADE with other algorithms on dimension 30

Next we study the comparative performance of GADE in optimizing the benchmark functions f1-f25 with dimension 30. The average errors of solutions found by GADE and the other algorithms are given in Table 5. We can observe from this table that GADE overall performed the best on the unimodal functions since it obtained the best results in 9 of the 12 functions. Moreover, GADE was the second best on function f18 and the third best on functions f5 and f7.

Regarding multimodal functions it is revealed from Table 5 that GADE generally was more attractive than basic DE and MDE_pBX-P, as it was better than basic DE in 11 of the 13 functions and better than MDE_pBX-P in 9 of the 13 functions. GADE was better than SaDE-P in 5 of the 13 functions, while in Functions f8-f13 both algorithms produced equal results. Moreover, GADE appeared competitive to JADE-P in the sense that GADE outperformed JADE-P in 3 functions (f19, f20 and f25) while JADE-P was superior to GADE in Functions f23 and f24. Although SHADE-P was slightly better than GADE in 5 of the 13 multimodal functions, GADE was much better than SHADE-P on Function f25.

Table 6 summarizes the results of pairwise comparisons of GADE against the other algorithms on all the benchmark functions (both unimodal and multimodal). The table shows that GADE outperformed basic DE, SaDE-P, JADE-P and MDE_pBX-P in at least 12 functions (which is almost 50% of all the functions used for tests), and this number is much larger than the number of functions on which GADE was dominated by any one of them. When compared with SHADE-P, GADE exhibited its superiority on 7 functions, while SHADE-P outperformed GADE on the other 7 functions.

The total ranking of algorithms is now made according to their relative errors across the 25 benchmark functions with dimension 30. As is indicated

**Table 2**. Average errors of solutions found by the algorithms in dimension 10

| FUNCTION | GADE mean(Std. Dev.) | SaDE-P mean(Std. Dev.) | JADE-P mean(Std. Dev.) | SHADE-P mean(Std. Dev.) | MDE_pBX-P mean(Std. Dev.) | Basic DE mean(Std. Dev.) |
|---|---|---|---|---|---|---|
| f14 | **0,00E+00**(0,00E+0) | **0,00E+00**(0,00E+00) | **0,00E+00**(0,00E+00) | **0,00E+00**(0,00E+00) | **0,00E+00**(0,00E+00) | **0,00E+00**(0,00E+00) |
| f15 | **0,00E+00**(0,00E+00) | 4,20E-06(9,76E-06) | 5,46E-02(5,97E-02) | **0,00E+00**(0,00E+00) | 6,92E-04(3,56E-03) | **0,00E+00**(0,00E+00) |
| f16 | 9,72E+02(2,08E+03) | 2,42E+05(3,15E+05) | 6,66E+05(4,03E+05) | **0,00E+00**(0,00E+00) | 3,08E+04(3,23E+04) | 3,36E-03(4,30E-03) |
| f17 | **0,00E+00**(0,00E+00) | 9,65E-02(5,48E+03) | 4,36E+01(2.93E+01) | 1,15E-01(6,21E-01) | 2,61E-07(1,29E-06) | **0,00E+00**(0,00E+00) |
| f18 | **0,00E+00**(0,00E+00) | **0,00E+00**(0,00E+00) | 5,86E-07(8,26E-07) | **0,00E+00**(0,00E+00) | **0,00E+00**(0,00E+00) | **0,00E+00**(0,00E+00) |
| f19 | 2,59E+00(1,86E+00) | 2,69E+00(1,79E+00) | 1,26E+00(9,95E-01) | **0,00E+00**(0,00E+00) | 3,71E+00(9,68E-01) | **0,00E+00**(0,00E+00) |
| f20 | 9,90E-02(5,79E-02) | 7,50E-02(2,20E-02) | 1,44E-01(4,40E-02) | 6,30E-02(5,11E-02) | **2,20E-02**(1,66E-02) | 4,42E-01(1,14E-01) |
| f21 | 2,03E+01(9,01E-02) | 2,04E+01(6,43E-02) | 2,03E+01(1,42E-01) | **2,01E+01**(1,48E-01) | 2,04E+01(6,62E-02) | 2,04E+01(6,89E-02) |
| f22 | **0,00E+00**(0,00E+00) | **0,00E+00**(0,00E+00) | **0,00E+00**(0,00E+00) | **0,00E+00**(0,00E+00) | **0,00E+00**(0,00E+00) | 1,41E+01(5,29E+00) |
| f23 | 1,16E+01(2,73E+00) | 8,10E+00(1,64E+00) | 1,18E+01(2,88E+00) | 8,69E+00(2,35E+00) | **7,01E+00**(3,12E+00) | 2,86E+01(6,02E+00) |
| f24 | 5,69E+00(1,78E+00) | 6,11E+00(8,91E-01) | 6,07E+00(6,80E101) | **4,69E+00**(2,26E+00) | 5,72E+00(1,17E+00) | 7,01E+00(2,49E+00) |
| f25 | **3,44E+01**(1,31E+02) | 2,14E+02(1,27E+02) | 3,60E+02(1,90E+02) | 1,41E+02(1,91E+02) | 4,63E+01(6,96E+03) | 4,95E+01(2,41E+02) |

**Table 3**. Comparison of GADE with every other algorithm (D10)

| ALGORITHM | UNIMODAL FUNCTIONS | | | MULTIMODAL FUNCTIONS | | | OVERALL | | |
|---|---|---|---|---|---|---|---|---|---|
| | $\succ$ | = | $\prec$ | $\succ$ | = | $\prec$ | $\succ$ | = | $\prec$ |
| SaDE-P | **3** | 2 | 0 | **4** | 1 | 2 | **7** | 3 | 2 |
| JADE-P | **4** | 1 | 0 | **5** | 1 | 1 | **9** | 2 | 1 |
| SHADE-P | 1 | **3** | 1 | 1 | 1 | **5** | 2 | 4 | **6** |
| MDE_pBX-P | **3** | 2 | 0 | **4** | 1 | 2 | **7** | 3 | 2 |
| DE/rand/1 | 0 | **4** | 1 | **6** | 0 | 1 | **6** | 4 | 2 |

**Table 4**. Ranking of the algorithms (D10)

| ALGORITHM | RANK | SUM OF RELATIVE ERRORS |
|---|---|---|
| GADE | 2 | 3,23 |
| SaDE-P | 4 | 4,08 |
| JADE-P | 6 | 7,93 |
| SHADE-P | 1 | **2,53** |
| MDE_pBX-P | 3 | 3,29 |
| DE/rand/1 | 5 | 5,13 |

in Table 7, GADE is ranked as the best in overall performance and the sum of its relative errors is much smaller than that of the others.

## 5.4 Overall Remarks and Discussion

Based on comparison of GADE against the other six DE variants in benchmark functions with dimensions 10 and 30, we would like to make the overall remarks and discussion as follows: No algorithm in comparison was better than any others on all the benchmark functions. This is not surprising and it is consistent with the No Free Lunch Theorems for optimization [24], [25], which imply that every optimization algorithm can be more competent than others in a specific class of problems.

Generally GADE outperformed the four algorithms: basic DE, SaDE-P, JADE-P, and MDE-pBX-P in problems of dimensions 10 and 30. This is reflected by the fact that GADE obtained better results than its counterparts on the majority of the functions used for comparison. Further, considering the sum of relative errors, the superiority of GADE seemed substantially enhanced when the problem dimension was increased from 10 to 30.

The comparative performance of GADE against SHADE-P depends on the problem dimension. In problems with dimension 10, GADE was considered inferior in terms of both the sum of relative errors and the number of functions on which one dominates the other. However, when dimension is scaled to 30, GADE turned to be strongly competitive and it was ranked prior to SHADE-P in view of accumulated relative errors.

The current results of experiments leave us with a sense that GADE would be particularly powerful in improving the performance of optimization in high dimensional problems. We conjecture this based on the evidences of the enhanced superiority of GADE against basic DE, SaDE-P, JADE-P and MDE-pBX-P, as well as the switch of the ranked positions between GADE and SHADE-P, when the dimension of problems scaled up from 10 to 30.

GADE could improve problem solving in unimodal problems by accelerating the speed of convergence, which is exemplified by Figure 1 (a) showing the evolution processes with various algorithms on function f16. On multimodal functions, GADE has shown its strength to help avoiding local optima, as demonstrated in Figure 1 (b) for the example on function f25.

Figures 2 and 3 illustrate how the scaling factor and the location parameter for the crossover probability were adapted in GADE on several benchmark functions. It is seen that both parameters were modified on-line with small and smooth variations. This is consistent with the nature of the local (parameter) adjustment scheme used by GADE during the evolutionary process. Perhaps there is some room for enhancement of our adaptation method to accelerate reaching suitable parameter settings, which would be an open interesting issue for further investigation.

**Table 5**. Average errors of solutions found by the algorithms in dimension 30

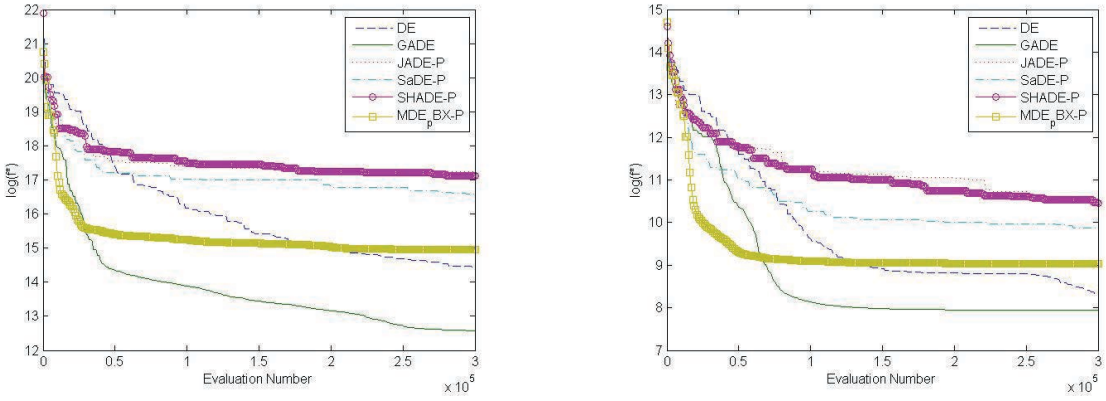
| FUNCTION | GADE mean(Std. Dev.) | SaDE-P mean(Std. Dev.) | JADE-P mean(Std. Dev.) | SHADE-P mean(Std. Dev.) | MDE_pBX-P mean(Std. Dev.) | Basic DE mean(Std. Dev.) |
|---|---|---|---|---|---|---|
| f1 | **0,00E+00**(0,00E+00) | **0,00E+00**(0,00E+00) | **0,00E+00**(0,00E+00) | **0,00E+00**(0,00E+00) | **0,00E+00**(0,00E+00) | **0,00E+00**(0,00E+00) |
| f2 | **0,00E+00**(0,00E+00) | **0,00E+00**(0,00E+00) | **0,00E+00**(0,00E+00) | **0,00E+00**(0,00E+00) | **0,00E+00**(0,00E+00) | 1,82E-08(1,13E-08) |
| f3 | **3,09E-01**(7,00E+00) | 1,66E+03(2,01E+03) | 4,40E+03(2,86E+03) | 4,35E+02(1,34E+03) | 6,58E+02(3,21E+02) | 6,55E+01(3,92E+01) |
| f4 | **7,30E-02**(5,21E-01) | 1,68E+00(3,32E+00) | 2,93E-01(5,85E-01) | 2,56E+00(2,15E+00) | 7,85E+00(3,38E+00) | 6,22E+00(5,07E+00) |
| f5 | 2,54E+01(5,26E+01) | 2,75E+01(5,44E+01) | 5,57E+01(1,55E+00) | **6,95E+00**(5,01E+00) | 3,54E+01(2,30E+01) | 2,31E+01(2,00E+01) |
| f6 | **0,00E+00**(0,00E+00) | **0,00E+00**(0,00E+00) | **0,00E+00**(0,00E+00) | **0,00E+00**(0,00E+00) | **0,00E+00**(0,00E+00) | **0,00E+00**(0,00E+00) |
| f7 | 2,27E-03(1,73E-03) | 7,48E-03(1,54E-02) | 4,76E-03(1,97E-03) | 2,09E-03(8,34E-04) | **1,51E-03(4,74E-04)** | 1,22E-02(3,79E-03) |
| f8 | **0,00E+00**(0,00E+00) | **0,00E+00**(0,00E+00) | **0,00E+00**(0,00E+00) | **0,00E+00**(0,00E+00) | 5,13E+01(9,17E+01) | 2,72E+03(8,15E+02) |
| f9 | **0,00E+00**(0,00E+00) | **0,00E+00**(0,00E+00) | **0,00E+00**(0,00E+00) | **0,00E+00**(0,00E+00) | 4,76E+00(1,23E+01) | 1,30E+01(3,70E+00) |
| f10 | **0,00E+00**(0,00E+00) | **0,00E+00**(0,00E+00) | **0,00E+00**(0,00E+00) | **0,00E+00**(0,00E+00) | 3,99E-07(2,18E-06) | 1,88E+01(4,28E+00) |
| f11 | **0,00E+00**(0,00E+00) | **0,00E+00**(0,00E+00) | **0,00E+00**(0,00E+00) | **0,00E+00**(0,00E+00) | 7,40E-04(2,26E-03) | 8,22E-04(2,49E-03) |
| f12 | **0,00E+00**(0,00E+00) | **0,00E+00**(0,00E+00) | **0,00E+00**(0,00E+00) | **0,00E+00**(0,00E+00) | 3,85E-03(1,89E-02) | 3,46E-03(1,86E-02) |
| f13 | **0,00E+00**(0,00E+00) | **0,00E+00**(0,00E+00) | **0,00E+00**(0,00E+00) | **0,00E+00**(0,00E+00) | **0,00E+00**(0,00E+00) | 3,66E-04(1,97E-03) |
| f14 | **0,00E+00**(0,00E+0) | **0,00E+00**(0,00E+00) | **0,00E+00**(0,00E+00) | **0,00E+00**(0,00E+00) | **0,00E+00**(0,00E+00) | **0,00E+00**(0,00E+00) |
| f15 | **6,63E+00**(2,60E+01) | 5,20E+02(7,67E+02) | 1,22E+03(1,18E+03) | 2,71E+02(8,63E+02) | 2,90E+03(2,10E+03) | 3,39E+01(2,04E+01) |
| f16 | **7,01E+05**(5,21E+05) | 1,30E+07(8,26E+06) | 1,78E+07(9,33E+06) | 6,60E+06(1,07E+07) | 8,08E+06(8,48E+06) | 6,41E+06(2,21E+06) |
| f17 | **9,79E-01**(3,23E+00) | 7,57E+03(5,48E+03) | 1,15E+04(5,27E+03) | 6,68E+03(7,37E+03) | 6,49E+03(4,76E+03) | 2,72E+02(1,50E+02) |
| f18 | 1,02E+03(7,15E+02) | 2,50E+03(7,20E+02) | 3,50E+03(5,12E+02) | 1,60E+03(1,25E+03) | 2,64E+03(4,52E+02) | **1,36E+02**(1,61E+02) |
| f19 | 3,21E+01(1,99E+01) | 4,83E+01(3,12E+01) | 4,04E+01(2,37E+01) | **9,39E+00**(1,26E+01) | 1,12E+04(6,05E+04) | 3,18E+01(2,89E+01) |
| f20 | 6,58E-03(8,67E-03) | 2,70E-02(3,48E-02) | 1,12E-01(6,89E-02) | **3,89E-03**(6,48E-03) | 1,75E-01(3,28E-01) | 5,23E-02(2,80E-01) |
| f21 | 2,09E+01(5,89E-02) | 2,11E+01(6,77E-02) | 2,09E+01(5,70E-02) | **2,02E+01**(2,26E-01) | 2,09E+01(5,64E-02) | 2,10E+01(4,68E-02) |
| f22 | **0,00E+00**(0,00E+00) | 3,32E-02(1,82E-01) | **0,00E+00**(0,00E+00) | **0,00E+00**(0,00E+00) | 2,42E+00(1,42E+00) | 1,22E+01(5,00E+00) |
| f23 | 1,08E+02(2,86E+01) | 7,56E+01(3,05E+01) | 7,48E+01(1,48E+01) | **6,02E+01**(2,33E+01) | 3,62E+01(1,31E+01) | 1,47E+02(5,38E+01) |
| f24 | 3,09E+01(5,10E+00) | 2,93E+01(8,62E+00) | 2,98E+01(2,58E+00) | **2,68E+01**(5,06E+00) | 2,98E+01(4,30E+00) | 3,96E+01(1,19E+00) |
| f25 | **2,53E+03**(2,95E+03) | 2,17E+04(6,25E+03) | 3,29E+04(5,52E+03) | 2,93E+04(8,64E+03) | 9,91E+03(6,96E+03) | 1,22E+04(8,97E+03) |

**Table 6**. Comparison of GADE with every other algorithm (D30)

| ALGORITHM | UNIMODAL FUNCTIONS | | | MULTIMODAL FUNCTIONS | | | OVERALL | | |
|---|---|---|---|---|---|---|---|---|---|
| | $\succ$ | = | $\prec$ | $\succ$ | = | $\prec$ | $\succ$ | = | $\prec$ |
| SaDE-P | **8** | 4 | 0 | **5** | 6 | 2 | **13** | 10 | 2 |
| JADE-P | **8** | 4 | 0 | **4** | 8 | 3 | **12** | 10 | 3 |
| SHADE-P | **6** | 4 | 2 | 1 | 7 | **5** | 7 | **11** | 7 |
| MDE_pBX-P | **7** | 4 | 1 | **9** | 2 | 2 | **16** | 6 | 3 |
| DE/rand/1 | **8** | 3 | 1 | **11** | 1 | 1 | **19** | 4 | 2 |

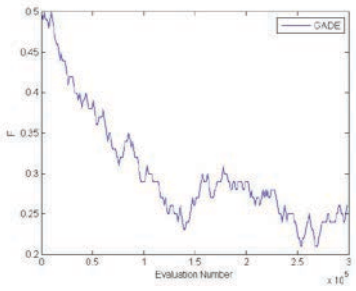
**Table 7**. Ranking of all algorithms (D30)

| ALGORITHM | RANK | SUM OF RELATIVE ERRORS |
|---|---|---|
| GADE | 1 | **3,65** |
| SaDE-P | 3 | 6,33 |
| JADE-P | 4 | 10,2 |
| SHADE-P | 2 | 4,73 |
| MDE_pBX-P | 5 | 11,2 |
| DE/rand/1 | 6 | 14,3 |

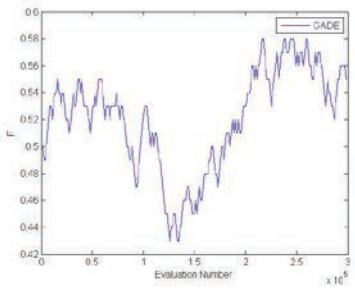



(a) f16 (b) f25

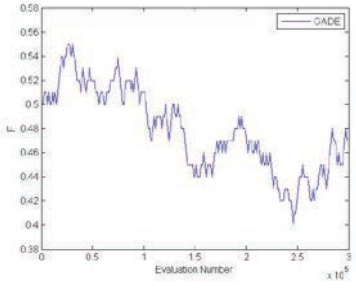
**Figure 1**. Median convergence characteristics of DE, GADE, SaDE-P, JADE-P, SHADE-P and MDE_pBX-P with dimension 30



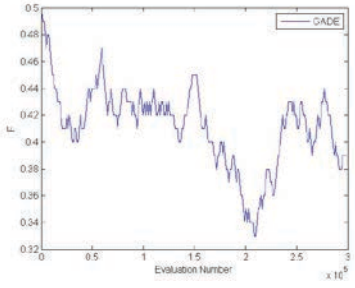

(a) f15



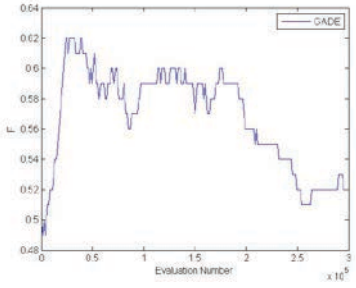

(b) f16



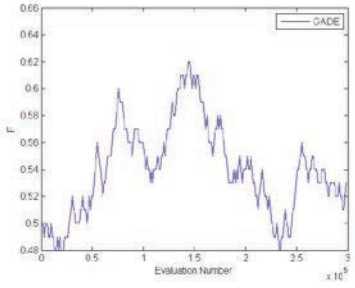

(c) f17



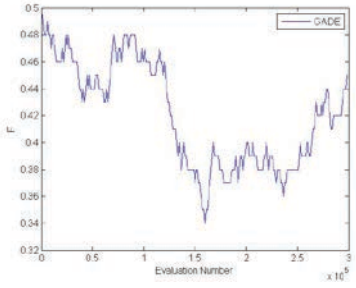

(d) f18



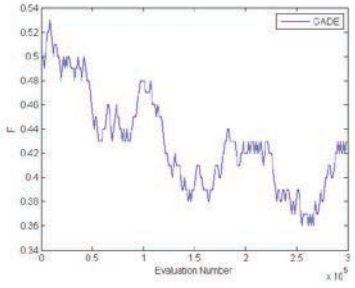

(e) f21




(f) f23

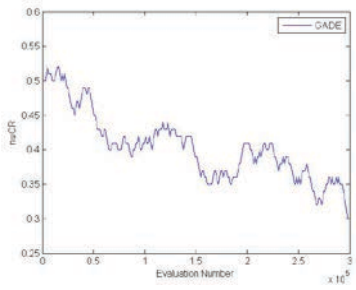



(g) f24

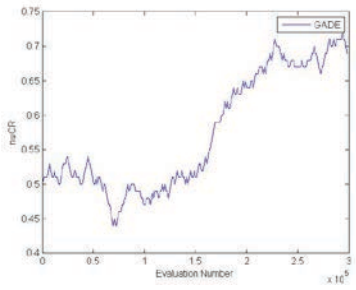



(h) f25

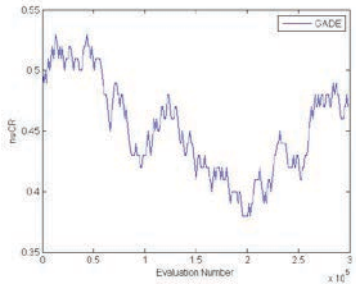
**Figure 2**. Self Adaptation of the scaling factor in GADE on functions with dimension 30



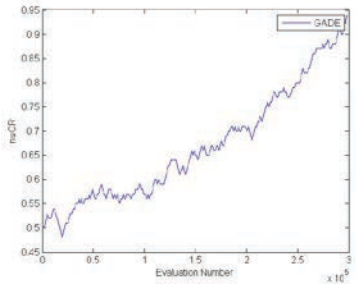

(a) f15



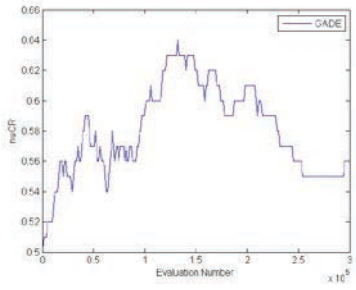

(b) f16



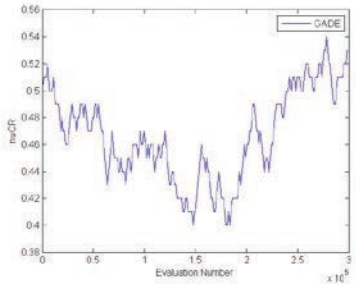

(c) f17



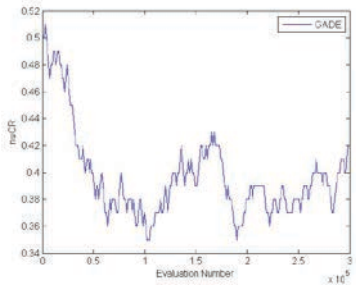

(d) f18



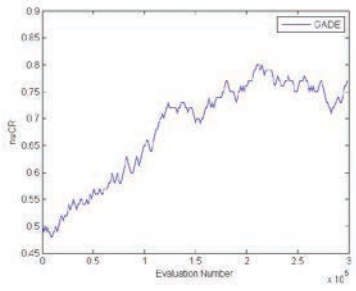

(e) f21




(f) f23




(g) f24




(h) f25

**Figure 3**. Self Adaptation of the location parameter for crossover probability in GADE on functions with dimension 30

## 6 Conclusion

In this paper, we propose GADE (greedy adaptive differential evolution) as a new adaptive DE algorithm, which adjusts two of its control parameters (scaling factor and crossover rate) during the process of running the algorithm. Greedy search is performed in GADE in order to progressively find better parameter assignments in the neighbourhood of the current assignment. We seek the parameter assignments that not only offer high chances for trial solutions to survive but also facilitate fast fitness improvement in the next generation. GADE was tested on 25 benchmark functions (with dimensions 10 and 30), in comparison with other five DE variants. The results of evaluation demonstrate that GADE is strongly competitive: it obtains the best rank in the high dimensional problems and the second best rank in the low dimensional problems among all the algorithms in comparison.

In future we plan to enhance GADE with a new mutation strategy or mutation strategy adaptation method. Also we intend to include some local search strategies such as [26], to develop a memetic adaptive DE algorithm that takes advantage of the best features from each of the techniques. Moreover, GADE will be tested and possibly further improved in real industrial scenarios.

## Acknowledgment

The work is funded by the Swedish Knowledge Foundation (KKS) grant (project no 16317). The authors are also grateful to ABB FACTS, PREVAS and VOITH for their co-financing of the project.

## References

[1] N. Xiong, D. Molina, M. Leon, and F. Herrera, A walk into metaheuristics for engineering optimization: Principles, methods, and recent trends, International Journal of Computational Intelligence Systems, vol. 8, no. 4, pp. 606–636, 2015.

[2] N. Hansen and A. Ostermeier, Completely derandomized self-adaptation in evolution strategies, Evolutionary Computation, vol. 9, no. 2, pp. 159–195, 2001.

[3] F. Herrera and M. Lozano, Two-loop real-coded genetic algorithms with adaptive control of mutation step size, Applied Intelligence, vol. 13, pp. 187–204, 2000.

[4] D. Molina, M. Lozano, A. M. Sanchez, and F. Herrera, Memetic algorithms based on local search chains for large scale continuous optimization problems: Ma-ssw-chains, Soft Computing, vol. 15, pp. 2201–2220, 2011.

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

[6] J. Kenedy and R. C. Eberhart, Particle swarm optimization, in In Proc. IEEE Conference on Neural Networks, 1995, pp. 1942–1948.

[7] D. Karaboga, B. Gorkemli, C.Ozturk, and N. Karaboga, A comprehensive survey: artificial bee colony (abc) algorithm and applications, Artificial Intelligence Review, vol. 42, no. 1, pp. 21–57, 2012.

[8] M. Ali and A. Torn, Population set based global optimization algorithms: Some modifications and numerical studies, Computers and Operations Research, vol. 31, pp. 1703–1725, 2004.

[9] S. Garcia, D. Molina, M. Lozano, and F. Herrera, A study on the use of non-parametric tests for analyzing the evolutionary algorithmss behaviour: A case study on the cec2005special session on real parameter optimization, Journal of Heuristics, vol. 15, no. 6, pp. 617–644, 2009.

[10] S. Das and N. Suganthan, Differential evolution: A survey of the state-of-the-art, in IEEE Transactions on Evolutionary Computation, vol. 15, no. 1, 2011, pp. 4–31.

[11] R. Gamperle, S. D. Muller, and P. Koumoutsakos, A parameter study for differential evolution, in Advances in intelligent systems, fuzzy systems, evolutionary computation, vol. 10, 2002, pp. 293–298.

[12] K. Zielinski, P. Weitkemper, R. Laur, and K. D. Kammeyer, Parameter study for differential evolution using a power allocation problem including interference cancellation, in IEEE Congress on Evolutionary Computation, 2006, pp. 1857–1864.

[13] J. Zhang and A. C. Sanderson, An approximate gaussian model of differential evolution with spherical fitness functions, in Proc. IEEE Congress on Evolutionary Computation, 2007, pp. 2220–2228.

[14] J. Liu and J. Lampinen, A fuzzy adaptive differential evolution algorithm, Soft Computing, vol. 9, no. 6, pp. 448–462, 2005.

[15] F. Xue, A. C. Sanderson, P. P. Bonissone, and R. J. Graves, Fuzzy logic controlled multiobjective differential evolution, in Proc. IEEE Conference on Fuzzy Systems, 2005, pp. 720–725.

[16] A. Qin and P. Suganthan, Self-adaptive differential evolution algorithm for numerical optimization, The 2005 IEEE Congress on Evolutionary Computation, vol. 2, pp. 1785–1791, 2005.

[17] J. Zhang and A. Sanderson, Jade: Adaptive differential evolution with optional external archive, IEEE Transactions on Evolutionary Computation, vol. 13, pp. 945–958, 2009.

[18] S. M. Islam, S. Das, S. Ghoshand, S. Roy, and P. N. Suganthan, An adaptive differential evolution algorithm with novel mutation and crossover strategies for global numerical optimization, Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on, vol. 42, no. 2, pp. 482–500, 2012.

[19] Z. Yang, K. Tang, and X. Yao, Scability of generalized adaptive differential evolution for large-scale continuous optimization, Soft Computing, vol. 15, no. 11, pp. 2141–2155, 2001.

[20] R. Tanabe and A. Fukinga, Success-history based parameter adaptation for differential evolution, in 2013 IEEE Congress on Evolutionary Computation (CEC), Cancun, Mexico, 2013, pp. 71–78.

[21] M. Leon and N. Xiong, Investigation of mutation strategies in differential evolution for solving global optimization problems, in Artificial Intelligence and Soft Computing. springer, June 2014, pp. 372–383.

[22] X. Yao, Y. Liu, and G. Lin, Evolutionary programming made faster, in Proc. IEEE Transactions on Evolutionary Computation, vol. 3, no. 2, 1999, pp. 82–102.

[23] P. N. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y. P. Chen, A. Auger, and S. Tiwari, Problem definitions and evaluation criteria for the cec 2005 special session on real-parameter optimization, Technical Report, Nanyang Technological University, Singapore And KanGAL Report Number 2005005 (Kanpur Genetic Algorithms Laboratory, IIT Kanpur), Tech. Rep., May 2005.

[24] D. Wolpert and W. Macready, No free lunch theorems for optimization, IEEE Transactions on Evolutionary Computation, vol. 1, no. 1, pp. 67–82, 1997.

[25] D. Whitley and J. Rowe, Focused no free lunch theorems, in Proc. Conf. Genetic Evolutionary Computing, 2008, pp. 811–818.

[26] M. Leon and N. Xiong, Using random local search helps in avoiding local optimum in diefferential evolution, in Proc. Artificial Intelligence and Applications, AIA2014, Innsbruck, Austria, 2014, pp. 413–420.

**Miguel Leon** received the B.S. and M.S. degree in Computer Science from Granada University, Spain in 2011 and 2013, respectively. He has been worked toward the Ph.D. degree in the School of Innovation, Design and Engineering, Mälardalen University, Sweden since 2013. His research interests include evolutionary algorithms, differential evolution, and applications of evolutionary algorithms.

**Ning Xiong** is associate professor at Mälardalen University, Sweden. He obtained the Ph.D from the University of Kaiserslautern (Germany) with outstanding distinction. His research interests include: machine learning, evolutionary computation, fuzzy systems, management of uncertainty, as well as multi-sensor data fusion. He is serving as editorial board member for four international journals. He was program chair for the International Conference on Natural Computation 2014. He also has been program committee member for a number of conferences and invited referee for many leading international journals.