Improving Adaptive Differential Evolution with Controlled Mutation Strategy

Sayan Basu Roy, Mainak Dan, and Pallavi Mitra

Department of Electronics and Telecommunication Engineering,
Jadavpur University, Kolkata, India
{sayanetce,mainak.dan,pallavi.mitra1992}@gmail.com

Abstract. In this paper we have proposed a DE variant, abbreviated by ADE_CM, to improve optimization performance of DE by imposing controlled mutation strategy. We also incorporated the concept of selective pressure to choose the random vectors in the selection of donor vector for each population. Basically we used DE/rand/1 and DE/target-to-best/1 schemes (with modifications using selective pressure) in the selection of donor using controlled mutation. The control parameter for mutation, linearly decreasing with generation, is the complement of the probability of selecting DE/target-to-best/1 in each generation. The algorithm is basically a trade-off between diversity and greediness. To improve diversity scaling factor is made adaptive and also a worst p% scheme is used in the difference vector of donor. ADE_CM is tested on 25 benchmark functions of CEC 2005 in 50 and 100 dimensions. Experimental results show that this algorithm outperforms many popular DE variants on most of the functions.

Keywords: Differential Evolution, Parameter Adaptation, Controlled Mutation, Selective Pressure.

1 Introduction

Global numerical optimization is a very important and developing field of research because many real world problems can be specified as optimization. In order to solve these problems, many Evolutionary Algorithms (EAs) have been developed throughout the last two decades. EAs are stochastic search methods that have been developed on basis of the evolutionary process realized in nature. The notion behind EAs is to modify gradually population candidate solutions by means of natural selection and biological evolution. Differential Evolution (DE) [1] developed by Storn and Price, emerged as a very simple but competitive form of EAs. DE is a population-based stochastic-direct-search method. The DE family of algorithms has found many applications in real-life optimization problems. Though during last decade, the research on DE has reached a formidable state, DE faces difficulties in large number of modern applications due to its search performance. The performance of DE is quite parameter-dependent as realized from both experimental [2] and theoretical studies [3]. The search performance of DE needs improvement to avoid difficulties in certain kind of function as described in [4]. To minimize parameter tuning problem, researcher tried to develop some adaptive algorithms which would be capable of self-tuning the parameters in every generation on basis of successful parameters who were able to generate more optimized population vectors in last generation. Adaptive DE algorithms are most recent trend and these algorithms have secured top ranks in various competitions held under the IEEE congress on Evolutionary computation (CEC) conference series. For detail Study, readers may go through Self-Adaptive DE (SaDE) [6], Adaptive DE with optional external Archive (JADE) [7], jDE [8] etc. In this paper we have introduced another DE variant with an intention to increase its search ability. In this proposed DE, the mutation has been controlled with a control parameter and scaling factor is adapted with help of previous successful scaling factors. This algorithm is named as improving Adaptive DE with controlled mutation strategy (ADE_CM) due to its own algorithmic components and characteristics.

2 ADE_CM Algorithm: Strategy and Parameter Adaptation

In this section we propose a new DE algorithm. These newly proposed components have been integrated with the classical DE family of algorithms. For detail study of Classical DE family, readers may consult [1].

2.1 Donor Selection through Controlled Mutation

The oldest and most successful DE mutation strategy is DE/rand/1/bin. Mezura-Montes *et al.* [5] have opined that incorporation of best solution information has some benefits over rand/1/bin strategy and used DE/target-to-best/1 in their algorithm. DE/target-to-best/*k* have faster convergence rate than DE/rand/*k* because incorporation of best solution, so far discovered guide the evolutionary search while the later has higher exploration capability than the former . However, best solution information may harm the exploration of sub search space, with a premature convergence. So initially exploring of the search subspace is more required. We have introduced a control parameter C which controls the donor selection for every target vector in each generation. C is linearly decreasing with increase in generation between 0.8 and 0.2. If rand(0,1) is greater than C, DE/target-to-best/1 policy is taken rather than DE/rand/1. This will enhance the chance of fast convergence near the optimum point.

2.2 Selective Pressure

In this scheme, r_1^i, r_2^i, r_3^i are chosen according to the linear distribution function proposed by Whitely [10]:

$$r_x := \left\lfloor \frac{\lambda}{2(\beta-1)} . \left(\sqrt{\beta^2 - 4.(\beta-1).U(0,1)}\right) \right\rfloor . \qquad (8)$$

In this formula, λ is the total number of population, among those r_x is to be selected. β is the bias term which controls the selective pressure. In our algorithm, we have selected β = 1.5. Selective Pressure is defined as the ratio of probabilities between selecting the fittest individual and selecting the individual with median fitness. Classical DE struggles with non-separable functions due to insufficient exploitation during selection of donor vector. To get rid of this difficulty, selective pressure is introduced here.

2.3 Worst $p\%$ Vector

Another modification is introduced for generating difference vector used in perturbation. We have created a group with worst $p\%$ vectors according to their functional value. The second vector in the difference term, i.e. the vector with index r_3^i is chosen from this group. Selective pressure is much more biased to the vectors with top-rank. This worst $p\%$ vector will diversify the population and explore the search subspace. When the algorithm reaches exploitation stage, the population vectors are in close proximity. Thus this policy does not affect the convergence procedure when the solutions are about to reach global optimum.

2.4 Parameter Adaptation

DE is more sensitive to the choice of scaling factor rather than crossover probability. For detail study, readers may go through [12]. So we have only used adaptation scheme only for *F*.

Scaling Factor. At each generation, the scaling factor (F_i) for every individual target vector is randomly produced from the Cauchy distribution with location parameter F_l and scaling parameter F_s according to the formula

$$F_i = randc(F_l, F_s). \tag{9}$$

The value of F_i's is truncated to 1 if distribution samples a value greater than 1. The value of Scaling Factor is re-sampled if a non-positive value is generated. S_F indicates the set of successful scale factors, so far generated in current generation, which produces a better trial vectors than the target vectors. Initially F_l and F_s are set to 0.5 and 0.15 respectively and then their values are updated in each generation as

$$\begin{aligned} F_l &= c_f.F_l + \left(1 - c_f\right).mean_p(S_F), \\ F_s &= c_f.F_s + \left(1 - c_f\right).mean_p(S_F), \end{aligned} \tag{10}$$

Where, the weight factor c_f varies randomly between 0.9 and 1. $mean_p(.)$ stands for the power mean defined as

$$mean_p(S_F) = \left(\sum_{x \in S_F} x^n / |S_F|\right)^{1/n}, \tag{11}$$

where, $|S_F|$ denotes the cardinality of set S_F. In ADE_CM, we use n = 1.5 as it gives the best results on various test problems.

Crossover Probability. *CR* is a tuning element. High value of *CR* adopts more components from donor than the target vector. A controlled parameter is already used to generate best donor vectors. We keep *CR* at a high value (at upper bound of *CR* i.e. 0.9), to incorporate more components of donor vector into the trial vector.

2.5 Pseudo Code of ADE_CM

```
01. Begin
02. Set $F_l = 0.5$; $F_s = 0.15$; $CR = 0.9$;
03. Create a random initial set of population $\{\vec{X}_{i,0} \mid i = 1,2,3,\ldots,NP\}$ of dimension D.
04. Calculate the fitness value of each individual population vector.
05. For G = 1 to gen_max
06.     $S_F = \varphi$; $C = 0.8 - 0.6 \cdot (G/gen\_max)$;
07.     Sort the population vectors according to their fitness values.
08.     For i = 1 to NP
09.         Generate $F_i = randc(F_l, F_s)$;
10.         Randomly choose $r_1^i, r_2^i$ from current population using selective pressure such that $r_1^i \neq r_2^i \neq i$.
11.         Randomly choose $r_3^i$ from the worst 20% population vectors such that $r_1^i \neq r_2^i \neq i \neq r_3^i$.
12.         If $C \leq rand(0,1)$
13.             $\vec{V}_{i,G} = \vec{X}_{r_1,G} + F.(\vec{X}_{r_2,G} - \vec{X}_{r_3,G})$;
14.         Else
15.             $\vec{V}_{i,G} = \vec{X}_{r_1,G} + F.(\vec{X}_{best,G} - \vec{X}_{i,G}) + F.(\vec{X}_{r_2,G} - \vec{X}_{r_3,G})$;
16.         End if
17.         Generate $j_{rand}$ randomly from {1,2,....,D}.
18.         For j = 1 to D
19.             If $j = j_{rand}$ or rand(0,1) ≤ CR
20.                 $u_{j,i,G} = v_{j,i,G}$;
21.             Else
22.                 $u_{j,i,G} = x_{j,i,G}$;
23.             End if
24.         End for
25.         If $f(\vec{U}_{i,G}) \leq f(\vec{X}_{i,G})$
26.             $\vec{X}_{i,G+1} = \vec{U}_{i,G}$
27.             $S_F \leftarrow F_i$;
28.         Else
29.             $\vec{X}_{i,G+1} = \vec{X}_{i,G}$;
30.         End if
31.     End for
```

```
32.        $c_f = 0.9 + rand(0,1) \cdot 0.1$;
33.        $F_L = c_f \cdot F_L + (1 - c_f) \cdot mean_p(S_F)$;
34.        $F_S = c_f \cdot F_S + (1 - c_f) \cdot mean_p(S_F)$;
35. End for
36. End
```

3 Experiment and Results

The ADE_CM algorithm is tested on 25 standard numerical benchmark functions from the special session and competition on real parameter optimization under the IEEE CEC 2005. Functions 1-5 are unimodal, functions 6-12 are basic multimodal, functions 13-14 are expanded multimodal and functions 15-25 are hybrid composite functions. For detail study on these functions, reader may see in [11].

Table 1. Mean and Standard Deviation of the Error Values for f_1–f_{10}(50D). The best entries are marked in boldface.

Func / Algorithm	f_1 Mean (Std)	f_2 Mean (Std)	f_3 Mean (Std)	f_4 Mean (Std)	f_5 Mean (Std)
DE/rand/1/bin	8.9605e-005 (1.3478e-005)	4.0516e+003 (8.7997e+002)	5.6567e+07 (1.4526e+007)	1.0923e+004 (3.332e+003)	9.6015e+003 (7.0345e+002)
DE/target-to-best/1/bin	4.4563e-006 (8.7963e-007)	2.0365e+003 (1.2366e+003)	1.0030e+007 (6.4509e+006)	8.5655e+003 (2.7865e+003)	7.3479e+003 (1.2981e+003)
JADE	7.5613e-014 (2.8336e-014)	5.8645e-004 (5.3273e-006)	**8.6632e+004 (3.5567e+004)**	3.1760e+003 (3.9895e-001)	3.0499e+003 (6.4429e+002)
jDE	1.2546e-009 (6.1254e-010)	5.1088e+003 (2.4682e+003)	2.8845e+007 (4.4672e+006)	1.1104e+004 (3.3309e-001)	3.9856e+003 (4.7823e+02)
SaDE	6.3256e-011 (2.4565e-012)	2.0362e-003 (7.9685e-003)	8.7640e+005 (6.7825e+004)	8.9067e+004 (6.8719e+003)	6.0451e+003 (3.8971e+002)
DEGL	5.5612e-010 (8.5675e-011)	1.3206e-005 (9.4613e-06)	2.2958e+005 (1.1104e+005)	2.5547e+003 (1.8890e+002)	5.9984e+003 (8.0976e+002)
ADE_CM	**5.6843e-014 (0.0000e+000)**	**3.6815e-008 (3.8240e-008)**	9.4416e+005 (2.7796e+005)	**2.4608e+000 (1.3786e+000)**	**1.9463e+003 (9.9583e+001)**
Func / Algorithm	**f_6 Mean (Std)**	**f_7 Mean (Std)**	**f_8 Mean (Std)**	**f_9 Mean (Std)**	**f_{10} Mean (Std)**
DE/rand/1/bin	4.8869e+001 (1.0095e+001)	8.6659e+003 (8.8857e-002)	2.1167e+001 (6.8895e-002)	3.4519e+002 (5.8671e+001)	3.8820e+002 (1.8834e+001)
DE/target-to-best/1/bin	4.6672e+001 (1.2984e+001)	8.4915e+003 (4.8838e+002)	2.1144e+001 (8.5504e-002)	2.5629e+002 (2.6516e+001)	3.0957e+002 (3.0056e+001)
JADE	1.5570e+001 (5.8873e+000)	6.3398e+003 (2.8974e+001)	2.1130e+001 (5.5539e-002)	1.4377e+002) (9.9918e+000)	2.0083e+002 (2.9938e+001)
jDE	4.0089e+001 (7.0089e+000)	6.1186e+003 (1.0085e+002)	2.1133e+001 (2.1937e-002)	1.8183e+002 (1.5208e+001)	1.1077e+002 (3.9978e+001)
SaDE	**1.1198e+001 (9.7539e+000)**	6.0958e+003 (5.9976e-009)	2.1132e+001 (3.7782e-002)	1.0944e+002 (1.0975e+001)	9.8864e+001 (1.4575e+001)
DEGL	1.4072e+001 (1.0928e+001)	6.1094e+003 (6.8840e-011)	2.1131e+001 (1.6609e-002)	1.5439e+002 (1.9862e+001)	1.5619e+002 (2.4409e+001)
ADE_CM	2.2437e+001 (1.2226e+001)	**5.2295e-009 (6.3359e-009)**	**2.0648e+001 (6.2600e-002)**	**4.9748e+001 (7.9907e+000)**	**7.3627e+001 (1.1667e+001)**

Better on 8 functions, Bitter on 2 functions, Equality on 0 functions

Table 2. Mean and Standard Deviation of the Error Values for f_{11}–f_{25}(50D). The best entries are marked in boldface.

Func / Algorithm	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}
	Mean (Std)	Mean (Std)	Mean (Std)	Mean (Std)	Mean (Std)
DE/rand/1/ bin	7.3651e+001 (2.1073e+000)	2.0497e+006 (6.0276e+005)	4.0647e+001 (2.4622e+000)	2.3849e+001 (1.4982e-001)	4.9167e+002 (7.9545e+001)
DE/target-to-best/1/bin	5.0677e+001 (3.4437e+000)	2.6642e+006 (2.0975e+005)	3.8947e+001 (9.1973e+000)	2.285e+001 (3.9728e-001)	5.0944e+002 (7.0649e+001)
JADE	6.4072e+001 (2.0957e+000)	1.3847e+005 (9.2289e+003)	2.6579e+001 (2.9271e+000)	2.2265e+001 (5.0349e-001)	3.7549e+002 (3.0097e+001)
jDE	7.3315e+001 (1.2415e+000)	1.0277e+005 (1.2364e+004)	2.8627e+001 (5.0629e+000)	2.2296e+001 (5.0674e-001)	4.0002e+002 (2.0977e-008)
SaDE	6.6672e+001 (1.1694e+000)	**1.0085e+04 (2.0865e+03)**	3.0975e+001 (6.4087e+000)	2.2498e+001 (4.0329e-001)	3.9014e+002 (5.0497e+001)
DEGL	6.2097e+001 (1.2657e+000)	6.0389e+004 (3.4977e+004)	3.29746e+001 (3.0975e+000)	2.2267e+001 (1.0659e-001)	3.8875e+002 (3.0164e+001)
ADE_CM	**4.9014e+001 (6.8108e+000**)	1.0643e+006 (3.8026e+005)	**4.7727e+000 (8.2710e-001)**	**2.1897e+001 (3.7870e-001)**	**2.7719e+002 (3.6051e+001)**
Func / Algorithm	f_{16}	f_{17}	f_{18}	f_{19}	f_{20}
	Mean (Std)	Mean (Std)	Mean (Std)	Mean (Std)	Mean (Std)
DE/rand/1/bin	2.8743e+002 (1.0097e+001)	3.7948e+002 (1.0974e+002)	9.9157e+002 (5.8197e+001)	9.4700e+002 (5.6195e+001)	9.8713e+002 (5.4943e+001)
DE/target-to-best/1/bin	2.6719e+002 (2.0972e+001)	2.6719e+002 (4.09754e+001)	9.4974e+002 (4.0977e+001)	9.3072e+002 (2.7315e+001)	9.4061e+002 (6.0443e+001)
JADE	1.6458e+002 (1.9461e+001)	1.7911e+002 (2.0613e+001)	9.2027e+002 (3.7948e+000)	9.5591e+002 (4.9253e+001)	9.9034e+002 (6.0977e+001)
jDE	2.6487e+002 (9.0067e+000)	3.1974e+002 (3.7919e+001)	9.1861e+002 (2.3716e+000)	9.1627e+002 (1.4329e+001)	9.9816e+002 (1.9134e+001)
SaDE	1.4215e+002 (2.0974e+001)	1.8974e+002 (4.5619e+000)	9.0452e+002 (4.2919e+001)	9.3195e+002 (1.7492e+001)	9.3143e+002 (5.7254e+001)
DEGL	1.4038e+002 (1.9716e+001)	1.7309e+002 (5.9785e+000)	9.5673e+002 (1.6715e+001)	9.1575e+002 (3.0247e+001)	9.2196e+002 (4.6285e+001)
ADE_CM	**4.7949e+001 (2.0325e+000**)	**8.1306e+001 (6.4336e+001)**	**8.3667e+002 (5.7629e-003)**	**8.3713e+002 (1.453e-001)**	**8.3589e+002 (3.659e-002)**
Func / Algorithm	f_{21}	f_{22}	f_{23}	f_{24}	f_{25}
	Mean (Std)	Mean (Std)	Mean (Std)	Mean (Std)	Mean (Std)
DE/rand/1/bin	9.1108e+002 (6.4409e+002)	1.0039e+003 (7.7729e+001)	9.2184e+002 (3.1037e+002)	7.9327e+002 (1.9818e+001)	1.7618e+003 (7.1628e+000)
DE/target-to-best/1/bin	8.9937e+002 (4.9917e+002)	9.8815e+002 (6.3309e+001)	9.3487e+002 (2.9915e+002)	7.4841e+002 (1.7518e+001)	1.7834e+003 (8.9917e+000)
JADE	8.6619e+002 (4.7710e+002)	9.1347e+002 (1.0023e+001)	8.1067e+002 (1.9038e+002)	**2.0000e+002 (0.0000e+000)**	1.6728e+003 (5.9917e+000)
jDE	8.5584e+002 (1.5539e+002)	9.3416e+002 (1.9329e+001)	8.2969e+002 (1.3094e+002)	**2.0000e+002 (0.0000e+000)**	1.5927e+003 (3.9948e+000)
SaDE	8.4419e+002 (2.7747e+002)	9.5626e+002 (1.3548e+001)	8.7715e+002 (1.3382e+002)	**2.0000e+002 (0.0000e+000)**	1.7387e+003 (3.0185e+000)
DEGL	8.3612e+002 (1.0082e+002)	9.2857e+002 (1.2917e+001)	8.3491e+002 (1.5073e+002)	7.2276e+002 (8.619e+001)	1.6194e+003 (4.8749e+000)
ADE_CM	**7.2413e+002 (9.6580e-001)**	**5.0007e+002 (3.2000e-003)**	**7.2826e+002 (1.3640e-001)**	2.1583e+002 (6.3602e-002)	**2.1425e+002 (1.6429e+000)**

Better on 13 functions, Bitter on 2 functions, Equality on 0 functions

The error values achieved by ADE_CM is compared with several well-known DE variants including two Classical DE variants and four state-of-the-art ADE variants such as SaDE[6], JADE[7], jDE[8], DEGL[9] on 50*D* problems as shown in Table 1 and 2. Their parametric setups have been chosen according to the guidelines from their respective literature. We also simulate our algorithm in 100 dimensions and provide comparative results of all the algorithms in Table 3. We exclude the results of last eleven hybrid composite functions as they have excessive computational time in 100 dimensions. The population size for any DE variants has been kept at 100 irrespective of the dimension *D*. For each function, each of these algorithms is run 50 runs. The function error values are listed at maximum FES which was set to 5×10^5 for 50 dimensions and 1×10^6 for 100 dimensions.

All algorithms are implemented in MATLAB. Simulations have been done on an Intel core-2-duo PC with 3-GB RAM and 2.1 GHz speed.

Table 3. Mean and Standard Deviation of the Error Values for f_1–f_{14}(100D). The best entries are marked in boldface.

Algo / Func	DE/rand/1/bin	DE/target-to-best/1/bin	JADE	jDE	SaDE	DEGL	ADE_CM
f_1	3.0726e05 (4.097e-06)	8.4465e-06 (9.422e-08)	6.3185e-10 (4.425e-09)	8.483e-07 (4.985e-09)	8.3945e-08 (5.387e-09)	9.6844e-07 (4.094e-08)	**1.1937e-13 (1.798e-14)**
f_2	8.946e+04 (6.197e+4)	9.0218e+03 (7.955e-05)	3.3923e+03 (8.488e-06)	6.3871e+03 (7.493e-07)	8.0371e+04 (8.535e-03)	8.926e+04 (8.456e-06)	**8.49e-01 (6.109e-01)**
f_3	9.742e+07 (7.941e+5)	9.9016e+06 (4.485e+3)	**2.7371e+06 (6.473e+4)**	9.0632e+06 (4.893e+4)	7.9171e+06 (7.998e+2)	4.8326e+07 (5.095e+4)	6.271e+06 (2.772e+5)
f_4	2.9744e+05 (4.926e+3)	7.7854e+04 (8.198e+3)	4.935e+04 (7.977e-03)	7.9261e+04 (2.746e+3)	6.043e+04 (7.557e+3)	1.937e+05 (4.946e+4)	**3.4455e+03 (2.057e+2)**
f_5	1.045e+06 (4.782e+3)	5.099e+05 (6.038e+3)	7.5251e+05 (3.946e+3)	3.0641e+05 (7.435e+3)	8.9364e+05 (2.191e+3)	9.094e+05 (6.938e+3)	**7.451e+03 (7.133e+2)**
f_6	1.7584e+05 (4.039e+1)	9.4029e+04 (7.562e+1)	7.533e+04 (7.373e+1)	8.5734e+04 (7.365e+0)	2.1972e+04 (7.491e-1)	4.2684e+04 (5.835e+1)	**1.538e+02 (5.543e+1)**
f_7	1.8467e+05 (3.815e+2)	1.6781e+05 (8.844e+2)	9.0417e+04 (8.474e+2)	1.0325e+05 (6.594e+2)	9.8736e+04 (7.459e+2)	1.2383e+05 (9.457e+2)	**9.9000e-03 (1.18e-3)**
f_8	2.2645e+01 (3.2295e-1)	2.2297e+01 (8.1945e-1)	2.1964e+01 (9.467e-1)	2.1697e+01 (8.033e-1)	2.2695e+01 (9.079e-1)	2.221e+01 (4.825e-1)	**2.0943e+01 (4.290e-2)**
f_9	9.3384e+02 (4.648e+1)	9.4137e+02 (7.783e+1)	8.9272e+02 (4.504e+1)	9.4064e+02 (7.405e+1)	8.4337e+02 (4.135e+1)	9.5591e+03 (3.927e+1)	**1.8128e+02 (7.878e+0)**
f_{10}	9.7649e+02 (5.010e+1)	7.8905e+02 (6.452e+1)	5.857e+02 (8.876e+1)	6.9046e+02 (7.859e+1)	8.3596e+02 (1.656e+1)	8.2017e+02 (3.559e+1)	**2.4038e+02 (1.300e+1)**
f_{11}	9.2248e+01 (2.096e+0)	8.9057e+01 (6.194e+0)	9.2214e001 (4.872e+0)	**7.3325e+01 (9.445e+0)**	8.3915e+01 (4.091e+0)	8.1089e+01 (1.443e+1)	1.4533e+02 (4.265e-1)
f_{12}	9.0046e+06 (8.094e+5)	6.2285e+05 (4.113e+5)	6.768e+05 (2.106e+4)	7.4547e+05 (6.562e+5)	**9.0778e+04 (5.893e+3)**	8.7781e+05 (7.762e+4)	4.0002e+06 (5.226e+5)
f_{13}	5.1309e+01 (4.605e+0)	4.4482e+01 (3.015e+0)	3.3414e+00 (4.108e+0)	3.8890e+01 (3.583e+1)	3.5583e+00 (5.109e+0)	3.9914e+01 (2.082e+1)	**1.4151e+01 (1.686e+0)**
f_{14}	4.4315e+01 (1.6457e-1)	4.5126e+01 (4.1943e-1)	4.6611e+01 (3.9078e-1)	4.3309e+01 (2.7712e-1)	4.2891e+01 (2.9320e-1)	**4.2276e+01 (7.0286e-1)**	4.3444e+01 (5.713e-1)

Better on 10 functions, Bitter on 4 functions, Equality on 0 functions

4 Conclusion

From the Experimental results we conclude that ADE_CM algorithm has defeated all the six algorithms in 21 numbers of functions of 50*D* where as in 100*D* this number is 10 among the first 14 functions. Therefore, it can be concluded that this algorithm has the ability to optimize functions in high-dimensional search space. The idea to compare the result in 100*D* fitness problem is to evaluate the search performance of ADE_CM with scaling of search space. Thus, this algorithm has the high probability of being used in large scale optimization problems in future research.

References

1. Storn, R., Price, K.V.: Differential Evolution—A Simple and Efficient Adaptive Scheme for Global Optimization Over Continuous Spaces, ICSI, Berkeley, CA, Tech. Rep. TR-95-012, `http://http.icsi.berkeley.edu/~storn/litera.html`
2. Gamperle, R., Muller, S.D., Koumoutsakos, P.: A parameter study for differential evolution. In: Proc. Advances Intell. Syst., Fuzzy Syst., Evol. Comput., Crete, Greece, pp. 293–298 (2002)
3. Zhang, J., Sanderson, A.C.: An approximate Gaussian Model of Differential Evolution with Spherical Fitness Functions. In: Proc. IEEE Congr. Evol. Comput., Singapore, pp. 2220–2228 (2007)
4. Ronkkonen, J., Kukkonen, S., Price, K.V.: Real Parameter Optimization with Differential Evolution. In: Proc. IEEE CEC, vol. 1, pp. 506–513 (2005)
5. Mezura-Montes, E., Vlázquez-Reyes, J., Coello Coello, C.A.: A Comparative Study of Differential Evolution Variants for Global Optimization. In: Proc. GECCO, pp. 485–492 (2006)
6. Qin, A.K., Huang, V.L., Suganthan, P.N.: Differential Evolution Algorithm with Strategy Adaptation for Global Numerical Optimization. IEEE Trans. Evol. Comput. 13(2), 398–417 (2009)
7. Zhang, J., Sanderson, A.C.: JADE: Adaptive Differential Evolution with Optional External Archive. IEEE Trans. Evol. Comput. 13(5), 945–958 (2009)
8. Brest, J., Greiner, S., Bošković, B., Mernik, M., Žumer, V.: Self-adapting Control Parameters in Differential Evolution: A Comparative Study on Numerical Benchmark Problems. IEEE Trans. Evol. Comput. 10(6), 646–657 (2006)
9. Das, S., Abraham, A., Chakraborty, U.K., Konar, A.: Differential Evolution Using A Neighborhood Based Mutation Operator. IEEE Trans. Evol. Comput. 13(3), 526–553 (2009)
10. Whitely, L.D.: The GENITOR Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best. In: Third International Conference on Genetic Algorithms, San Mateo, CA (1989)
11. Suganthan, P.N., Hansen, N., Liang, J.J., Deb, K., Chen, Y.P., Auger, A., Tiwari, S.: Problem Definitions and Evaluation Criteria for the CEC 2005 Special Session on Real-Parameter Optimization. Nanyang Technol. Univ., Singapore (2005)
12. Brest, J., Greiner, V., Bošković, B., Mernik, M., Žumer, V.: Self-adapting Control Parameters in Differential Evolution: A Comparative Study on Numerical Benchmark Problems. IEEE Trans. on Evol. Comput. 10(6), 646–657 (2006)