Simulating the Effect of Adaptivity on Randomization

Adam Viktorin Roman Senkerik Michal Pluhacek

Faculty of Applied Informatics, Tomas Bata University in Zlin, T. G. Masaryka 5555, 760 01 Zlin, Czech Republic,
{aviktorin,senkerik,pluhacek}@fai.utb.cz

Abstract

This paper compares the development of multi-chaotic system during the optimization process on three classical benchmark functions – Rosenbrock, Rastrigin and Ackley. The multi-chaotic system involves five different randomizations based on discrete chaotic maps (Burgers, Delayed Logistic, Dissipative, Lozi and Tinkerbell) and the probability of their selection is adjusted according to the development of the optimization task. Two variants of Differential Evolution (DE) are used in order to simulate the effect of adaptivity on the randomization probability adjustment process. First non-adaptive variant is DE with rand/1 mutation strategy and the second adaptive variant is novel Success-History based Adaptive DE (SHADE).

Keywords: randomization, differential evolution, SHADE, chaos, parent selection

1 Introduction

The Differential Evolution (DE) has played a significant role in optimization and outperformed other Evolutionary Computation Techniques (ECT) in many cases (Price *et al*, 2006; Kim *et al*, 2007; Chauhan *et al*, 2009; Babu and Jehan, 2003). The original version was introduced in 1995 (Storn and Price, 1995) and since then has been thoroughly studied and improved. One branch of improvement is in adapting its control parameters to the solved optimization task. The examples of adaptive variants are jDE (Brest *et al*, 2006), JADE (Zhang and Sanderson, 2009) and Success-History based Adaptive DE (SHADE) (Tanabe and Fukunaga, 2013). The last listed is used as a representative of adaptive DE variants in this research paper.

One of the recent research directions in ECT is the studying of effect of different randomizations on various parts of the evolutionary algorithms and swarm intelligence algorithms. Especially, the chaotic maps are often used as Pseudo-Random Number Generators (PRNGs) instead of the classical ones with uniform distribution (dos Santos Coelho *et al*, 2014; Senkerik *et al*, 2015b; Caponeto *et al*, 2003) or combinations of multiple chaotic systems with some sort of switching mechanism (Pluhacek *et al*, 2014; Senkerik *et al*, 2015a).

The main research question of this paper is whether there is a randomization or their combination, that would be preferred in parent selection process of non-adaptive and adaptive variants of DE and if the preferences vary for these two variants. In order to simulate that, the multi-chaotic framework containing five different chaotic map based PRNGs was created and probability adjustment process, which mirrors the preference is presented. DE and SHADE algorithms with multi-chaotic framework are tested on three classic benchmark functions – Rosenbrock, Rastrigin and Ackley and the resulting probability development is reported.

The remainder of this paper is structured as follows. Section 2 illustrates chaotic maps and their use as a PRNGs. Section 3 describes DE, SHADE and multi-chaotic framework with pseudo-codes. Following Section 4 is devoted to experiments and results and the whole paper is concluded in Section 5.

2 Chaotic Maps

The chaotic maps are systems generated continuously from a single initial position by simple equations. The current coordinates are generated from the previous ones, consequently creating a system which is extremely dependent on the initial position. The generated chaotic sequence varies for different initial positions. Therefore, the generation of the initial position is randomized to obtain unique chaotic sequences. The generation of starting positions is carried out by PRNG with uniform distribution. Chaotic map equations may also contain control parameters, which determine the chaotic behavior and dynamics.

Chaotic systems used in this research, with their generating equations, control parameter values and initial position generator settings are depicted in Table 1. All the control parameter values were set according to previous experiments and suggestions in literature (Sprott and Sprott, 2003).

The multi-chaotic framework used for parent selection presented in this paper uses five chaotic maps – Burgers, Delayed Logistic, Dissipative, Lozi and Tinkerbell. Each of the chaotic map based PRNGs has different probability distribution and unique sequencing, which may be beneficial for the parent selection process where the obtained parent vector combinations exhibit a different dynamic than that of

parent vector combinations selected by a PRNG with uniform distribution. The distribution of 5,000 real numbers from range [0, 1] generated by each chaotic map can be seen in Figure 1 and Figure 2. It is important to mention the differences between these chaotic systems. While Lozi and Dissipative chaotic maps tend to generate values from the whole range without any clear preference, other three chaotic maps visibly favor values close to the left end of the specified range. In fact, the distribution identifier in Wolfram Mathematica 10.2 software shown that the distribution of Lozi map generated values is beta distribution with shape parameters $\alpha = 1.03457$ and $\beta = 1.56153$ and similarly the distribution of Dissipative map generated values was identified as uniform with range [0.00034, 0.99857]. Other three were identified as mixtures of different distributions.

Table 1. Chaotic maps, generating equations, control parameters and initial position ranges

Chaotic map	Equations	Parameters	Initial position
Burgers	$X_{n+1} = aX_n - Y_n^2$ $Y_{n+1} = bY_n + X_nY_n$	$a = 0.75$ $b = 1.75$	$X_0 = (-0.1, -0.01)$ $Y_0 = (0.01, 0.1)$
Delayed Logistic	$X_{n+1} = AX_n(1 - Y_n)$ $Y_{n+1} = X_n$	$A = 2.27$	$X_0 = Y_0 = (0.8, 0.9)$
Dissipative	$X_{n+1} = X_n + Y_{n+1} \pmod{2\pi}$ $Y_{n+1} = bY_n + k\ sinX_n \pmod{2\pi}$	$b = 0.1$ $k = 8.8$	$X_0 = Y_0 = (0, 0.1)$
Lozi	$X_{n+1} = 1 - a\lvert X_n\rvert - bY_n$ $Y_{n+1} = X_n$	$a = 1.7$ $b = 0.5$	$X_0 = Y_0 = (0, 0.1)$
Tinkerbell	$X_{n+1} = X_n + Y_n + aX_n + bY_n$ $Y_{n+1} = 2X_nY_n + cX_n + dY_n$	$a = 0.9$ $b = -0.6$ $c = 2$ $d = 0.5$	$X_0 = (-0.1, -0.01)$ $Y_0 = (0, 0.1)$

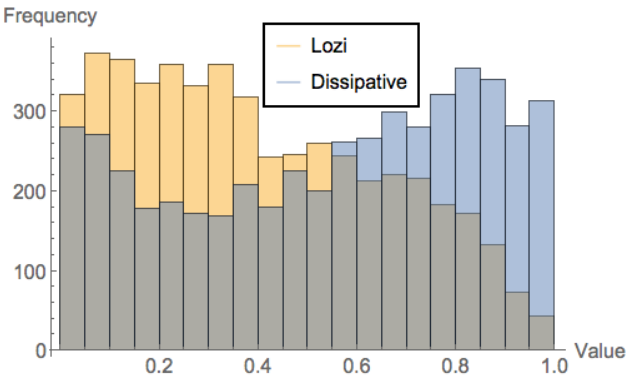

Figure 1. Lozi and Dissipative generated values in histogram.

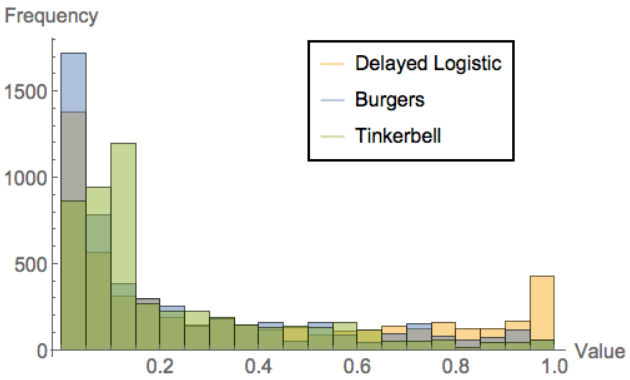

Figure 2. Delayed Logistic, Burgers and Tinkerbell generated values in histogram.

In order to use chaotic maps as PRNGs, the transformation rule had to be developed. The process of obtaining the i-th random integer value $rndInt_i$ from the chaotic map is presented in (1).

$$rndInt_i = \text{round}\left(\frac{\text{abs}(X_i)}{\max(\text{abs}(X_{i\in N}))} * (maxRndInt - 1)\right) + 1 \quad (1)$$

Where $\text{abs}(X_i)$ is the absolute value of the i-th generated X coordinate from the chaotic sequence of

length N, max(abs($X_{i \in N}$)) is a maximum value of all absolute values of generated X coordinates in chaotic sequence. The function round() is common rounding function and *maxRndInt* is a constant to ensure that integers will be generated in the range [1, *maxRndInt*].

3 Differential Evolution, Success-History based Adaptive Differential Evolution and Multi-Chaotic Framework

This section describes DE and SHADE algorithms and their individual parts. It also covers the proposed multi-chaotic framework for parent selection.

3.1 Differential Evolution and Success-History based Adaptive Differential Evolution

The DE (Storn and Price, 1995) has four static control parameters – number of generations G_{max}, population size NP, scaling factor F and crossover rate CR. In the evolutionary process of DE, these four parameters remain unchanged and depend on the user initial setting. SHADE algorithm, on the other hand, adapts the F and CR parameters during the evolution. The values that brought improvement to the optimization task are stored into according historical memories M_F and M_{CR}.

The whole process can be divided into five parts – initialization, mutation strategy with parent selection, crossover, elitism and historical memory update (the last one is only in SHADE algorithm).

3.1.1 Initialization

DE: The initial population of size NP is generated randomly from the objective space. Control parameters F, CR and G_{max} are set.

SHADE: The initial population is generated as in DE, external archive of inferior solutions A is initialized empty and has a maximum size of NP. Both historical memories have the same size H and are initialized to $M_{CR,i} = M_{F,i} = 0.5$ for ($i = 1, \ldots, H$).

3.1.2 Mutation Strategy with Parent Selection

DE: The selected mutation strategy for DE algorithm in this paper is rand/1 (Storn and Price, 1995), which combines 3 different randomly selected parent vectors $\boldsymbol{x}_{r1,G}$, $\boldsymbol{x}_{r2,G}$ and $\boldsymbol{x}_{r3,G}$ from current generation G. Additionally, parent vectors have to differ from the original vector $\boldsymbol{x}_{i,G}$, therefore $\boldsymbol{x}_{i,G} \neq \boldsymbol{x}_{r1,G} \neq \boldsymbol{x}_{r2,G} \neq \boldsymbol{x}_{r3,G}$. All three parents are selected by the PRNG with uniform distribution. The rand/1 mutation is depicted in (2) where $\boldsymbol{v}_{i,G}$ is the resulting mutated vector and F is the static scaling factor.

$$\boldsymbol{v}_{i,G} = \boldsymbol{x}_{r1,G} + F\left(\boldsymbol{x}_{r2,G} - \boldsymbol{x}_{r3,G}\right) \quad (2)$$

SHADE: In the original version of SHADE algorithm (Tanabe andY Fukunaga, 2013), parent selection for mutation strategy is carried out by the PRNG with uniform distribution. The mutation strategy used in SHADE is current-to-*p*best/1 and uses four parent vectors – current i-th vector $\boldsymbol{x}_{i,G}$, vector $\boldsymbol{x}_{pbest,G}$ randomly selected from the $NP \times p$ best vectors (in terms of objective function value) from current generation G. The p value is randomly generated by uniform PRNG $U[p_{min}, 0.2]$, where $p_{min} = 2/NP$. Third parent vector $\boldsymbol{x}_{r1,G}$ is randomly selected from the current generation and last parent vector $\boldsymbol{x}_{r2,G}$ is also randomly selected, but from the union of current generation G and external archive A. Also, vectors $\boldsymbol{x}_{pbest,G}$, $\boldsymbol{x}_{i,G}$, $\boldsymbol{x}_{r1,G}$ and $\boldsymbol{x}_{r2,G}$ has to differ, $\boldsymbol{x}_{pbest,G} \neq \boldsymbol{x}_{i,G} \neq \boldsymbol{x}_{r1,G} \neq \boldsymbol{x}_{r2,G}$. The mutated vector $\boldsymbol{v}_{i,G}$ is generated by (3).

$$\boldsymbol{v}_{i,G} = \boldsymbol{x}_{i,G} + F_i\left(\boldsymbol{x}_{pbest,G} - \boldsymbol{x}_{i,G}\right) + F_i\left(\boldsymbol{x}_{r1,G} - \boldsymbol{x}_{r2,G}\right) \quad (3)$$

The i-th scaling factor F_i is generated from a Cauchy distribution with the location parameter $M_{F,r}$ (selected randomly from the scaling factor historical memory M_F) and scale parameter value of 0.1 (4). If $F_i > 1$, it is truncated to 1 also if $F_i \leq 0$, (4) is repeated.

$$F_i = C\left[M_{F,r}, 0.1\right] \quad (4)$$

DE and SHADE: If any of the features of the mutated vector $\boldsymbol{v}_{i,G}$ is outside the boundaries of objective space in that dimension [$x_{j,min}$, $x_{j,max}$], it is constrained as shown in (5).

$$v_{j,i,G} = \begin{cases} \left(x_{j,min} + x_{j,i,G}\right)/2 & \text{if } v_{j,i,G} < x_{j,min} \\ \left(x_{j,max} + x_{j,i,G}\right)/2 & \text{if } v_{j,i,G} > x_{j,max} \end{cases} \quad (5)$$

3.1.3 Crossover

DE and SHADE: Binomial crossover operation generates the trial vector $\boldsymbol{u}_{i,G}$ from mutated vector $\boldsymbol{v}_{i,G}$ and current vector $\boldsymbol{x}_{i,G}$. The crossover operation uses compare rule with the threshold CR (6). In DE, this threshold is static, on the other hand, in SHADE its value CR_i is calculated for each individual in generation. CR_i is generated from a normal distribution with a mean parameter value $M_{F,r}$ (selected randomly from the crossover rate historical memory M_{CR}) and standard deviation value of 0.1 (7). If the CR_i value is outside of the interval [0, 1], the closer limit value (0 or 1) is used.

$$u_{j,i,G} = \begin{cases} v_{j,i,G} & \text{if } rand[0,1] \leq CR_i \text{ or } j = j_{rand} \\ x_{j,i,G} & \text{otherwise} \end{cases} \quad (6)$$

$$CR_i = N\left[M_{CR,r}, 0.1\right] \quad (7)$$

The j index is the index of vector feature and j_{rand} is the index of a feature (randomly selected), which has to be taken from the mutated vector. Without the j_{rand} index, the trial vector $\boldsymbol{u}_{i,G}$ could be the same as the current vector $\boldsymbol{x}_{i,G}$ and that would result in unnecessary elitism in the next step of the algorithm.

3.1.4 Elitism

DE and SHADE: Elitism is the algorithm feature which ensures that the next generation G+1 will contain only

equal or better individuals in terms of objective function value (8). If the objective function value of the trial vector $\boldsymbol{u}_{i,G}$ is better than that of the current vector $\boldsymbol{x}_{i,G}$, the trial vector will become the new individual in new generation $\boldsymbol{x}_{i,G+1}$ and the original vector $\boldsymbol{x}_{i,G}$ will be moved to the external archive of inferior solutions A (SHADE only). Otherwise, the original vector remains in the population in next generation and external archive remains unchanged.

$$\boldsymbol{x}_{i,G+1} = \begin{cases} \boldsymbol{u}_{i,G} & \text{if } f(\boldsymbol{u}_{i,G}) < f(\boldsymbol{x}_{i,G}) \\ \boldsymbol{x}_{i,G} & \text{otherwise} \end{cases} \quad (8)$$

3.1.5 Historical Memory Update

SHADE: Values of F_i and CR_i of individuals successful in elitism are stored into two corresponding arrays S_F and S_{CR}. After each generation, those arrays are used to update k-th cell in both historical memories M_F and M_{CR}. The index k is initialized to 1 before the first generation and after each update it is incremented by 1. If it overflows the size of historical memories H, it is set back to 1. When the whole generation fails to improve, S_F and S_{CR} arrays are empty and no update takes place. Also the k index value stays the same. Equations used for historical memory updates are given in (9) and (10).

$$M_{F,k,G+1} = \begin{cases} \text{mean}_{WL}(S_F) & \text{if } S_F \neq \emptyset \\ M_{F,k,G+1} & \text{otherwise} \end{cases} \quad (9)$$

$$M_{CR,k,G+1} = \begin{cases} \text{mean}_{WA}(S_{CR}) & \text{if } S_{CR} \neq \emptyset \\ M_{CR,k,G+1} & \text{otherwise} \end{cases} \quad (10)$$

The weights for both weighted Lehmer mean $\text{mean}_{WL}(S_F)$ and weighted arithmetic mean $\text{mean}_{WA}(S_{CR})$ are evaluated by (11) and used in mean equations given in (12) and (13).

$$w_k = \frac{\text{abs}\left(f(\boldsymbol{u}_{k,G}) - f(\boldsymbol{x}_{k,G})\right)}{\sum_{m=1}^{|S_{CR}|} \text{abs}\left(f(\boldsymbol{u}_{m,G}) - f(\boldsymbol{x}_{m,G})\right)} \quad (11)$$

Since both arrays S_{CR} and S_F are of the same size, either of them can be used for the m index upper boundary of the sum in (11).

$$\text{mean}_{WL}(S_F) = \frac{\sum_{k=1}^{|S_F|} w_k \cdot S_{F,k}^2}{\sum_{k=1}^{|S_F|} w_k \cdot S_{F,k}} \quad (12)$$

$$\text{mean}_{WA}(S_{CR}) = \sum_{k=1}^{|S_{CR}|} w_k \cdot S_{CR,k} \quad (13)$$

3.2 Multi-Chaotic Framework for Parent Selection

Both mutation strategies rand/1 and current-to-$p$best/1 require randomly chosen parents, therefore the mutation can be significantly influenced by the used PRNG for the parent selection. It was experimentally tested that the chaotic map based PRNGs used for parent selection may improve the convergence speed and the ability to reach the global optimum. But the chaotic PRNG which improved the performance of the algorithm on one objective function might not be as suitable as other chaotic PRNG on different objective function. Therefore, the multi-chaotic framework was developed.

Multi-chaotic framework for parent selection presented in this paper was partially inspired by the ranking selection process in Genetic Algorithm (GA) (Holland, 1975). In order to implement framework into the evolutionary algorithm, a chaotic map based PRNG pool ***Cpool*** has to be added to the process. The ***Cpool*** used in this research contains five chaotic PRNGs and each of them has assigned probability value pc_j where j is the index of chaotic PRNG. At the beginning, all pc_j values are initialized to the same pc_{init} value, $pc_{init} = 1/Csize$ where $Csize$ is the size of ***Cpool***. As for this paper, $Csize = 5$ and $pc_{init} = 1/5 = 0.2 = 20\%$. The pc_j value determines the probability of a j-th chaotic PRNG to be used for parent selection.

For each individual vector $\boldsymbol{x}_{i,G}$ in generation G, the chaotic generator $PRNG_k$ is selected from the ***Cpool*** based on its probability pc_k, where k is the index of the selected generator. The selected generator is then used for the random selection of parent vectors. If the trial vector $\boldsymbol{u}_{i,G}$ generated from these parent vectors succeeds in elitism, then the probability pc_k of the selected generator $PRNG_k$ is increased and all other generators probabilities are decreased. The upper boundary for the probability is 60%, $pc_{max} = 0.6$. If the selected chaotic PRNG reaches the maximum probability, then no adjustment takes place. The probability adjustment process is depicted in pseudo-code below – Algorithm 1.

Algorithm 1: Probability adjustment of multi-chaotic system

```
1   Cpool = {Burgers, Delayed Logistic,
    Dissipative, Lozi, Tinkerbell};
2   Csize = 5, pc_max = 0.6;
3   k is the index of the selected
    chaotic system and pc_k is its
    selection probability;
4   if f(u_i,G) < f(x_i,G) and pc_k < pc_max
    then
5   for j = 1 to Csize do
6   if j = k then
7   pc_j = (pc_j + 0.01)/1.01;
8   else
9   pc_j = pc_j /1.01;
10  end
11  end
12  else
13  pc_j = pc_j;
14  end
```

Since the parent selection processes of DE and SHADE differ, the pseudo-code is divided into two algorithms – 2 for DE and 3 for SHADE.

Algorithm 2: Multi-chaotic parent selection in DE

```
1   Cpool = {Burgers, Delayed Logistic,
    Dissipative, Lozi, Tinkerbell};
2   Csize = 5;
3   i is the index of active individual
    x_{i,G};
4   P is the current population of
    individuals;
5   Nsize = |P|;
6   k = 1 the index of selected chaotic
    system;
7   selectedChaos = Burgers;
8   prob = U[0,1];
9   do
10  prob = prob − pc_{chaosIndex};
11  k++;
12  if k > Csize then break;
13  while (prob > 0)
14  k = k - 1;
15  selectedChaos = Cpool[k];
16  do
17  x_{r1,G} = P[selectedChaos.rndInt(1,
    Nsize)];
18  x_{r2,G} = P[selectedChaos.rndInt(1,
    Nsize)];
19  x_{r3,G} = P[selectedChaos.rndInt(1,
    Nsize)];
20  while (x_{i,G} = x_{r1,G} or x_{i,G} = x_{r2,G} or
    x_{i,G} = x_{r3,G} or x_{r1,G} = x_{r2,G} or x_{r1,G} =
    x_{r3,G} or x_{r2,G} = x_{r3,G})
```

Algorithm 3: Multi-chaotic parent selection in SHADE

```
1   Cpool = {Burgers, Delayed
    Logistic, Dissipative, Lozi,
    Tinkerbell};
2   Csize = 5;
3   i is the index of active
    individual x_{i,G};
4   P is the current population of
    individuals, A is the external
    archive, PB is the array of best
    individuals in population, its
    size is given by p in SHADE
    algorithm;
5   Nsize = |P|, NAsize = |P| + |A|,
    PBsize = |PB|;
6   k = 1 the index of selected
    chaotic system;
7   selectedChaos = Burgers;
8   prob = U[0,1];
9   do
10  prob = prob − pc_{chaosIndex};
11  k++;
12  if k > Csize then break;
13  while (prob > 0)
14  k = k - 1;
15  selectedChaos = Cpool[k];
16  do
17  x_{pbest,G} = PB[selectedChaos.rndInt(1,
    PBsize)];
18  x_{r1,G} = P[selectedChaos.rndInt(1,
    Nsize)];
19  x_{r2,G} = (P ∪
    A)[selectedChaos.rndInt(1,
    NAsize)];
20  while (x_{i,G} = x_{pbest,G} or x_{i,G} = x_{r1,G}
    or x_{i,G} = x_{r2,G} or x_{r1,G} = x_{r2,G} or x_{r1,G}
    = x_{pbest,G} or x_{r2,G} = x_{pbest,G})
```

The function *rndInt*(*min*, *max*) generates a random integer from the range [*min*, *max*] according to (1).

DE and SHADE algorithms with multi-chaotic parent selection were labeled MC-DE and MC-SHADE and their pseudo-codes are below in algorithms 4 and 5.

Algorithm 4: MC-DE

```
1   G = 0;
2   Randomly initialized population P =
    (x_{1,G, … , x_{NP,G});
3   Cpool = {Burgers, Delayed Logistic,
    Dissipative, Lozi, Tinkerbell};
4   All values in pc set to pc_{init} =
    0.2;
5   while termination = false do
6   for i = 1 to NP do
7   Multi-chaotic parent selection –
    Algorithm 2;
8   u_{i,G} mutation (2) and crossover (6);
9   if f(u_{i,G}) < f(x_{i,G}) then
10  x_{i,G+1} = u_{i,G};
11  Probability adjustment of multi-
    chaotic system – Algorithm 1;
12  else
13  x_{i,G+1} = x_{i,G};
14  end
15  end
16  G++;
17  end
```

Algorithm 5: MC-SHADE

```
G = 0, k = 1, A = ∅, p_min = 2 / NP;
Randomly initialized population P = (x_1,G, … , x_NP,G);
All values in M_F and M_CR set to 0.5;
Cpool = {Burgers, Delayed Logistic, Dissipative, Lozi, Tinkerbell};
All values in pc set to pc_init = 0.2;
while termination = false do
S_F = ∅, S_CR = ∅;
for i = 1 to NP do
r = U[1, H];
F_i,G = C[M_F,r, 0.1];
CR_i,G = N[M_CR,r, 0.1];
p_i,G = U[p_min, 0.2];
Multi-chaotic parent selection – Algorithm 3;
u_i,G mutation (3) and crossover (6);
if f(u_i,G) < f(x_i,G) then
x_i,G+1 = u_i,G;
x_i,G → A;
F_i,G → S_F, CR_i,G → S_CR;
Probability adjustment of multi-chaotic system – Algorithm 1;
else
x_i,G+1 = x_i,G;
end
end
if |A| > NP then delete random individuals from A;
if S_F ≠ ∅ and S_CR ≠ ∅ then
Update M_F and M_CR;
k++;
if k > H then k = 1;
end
G++;
end
```

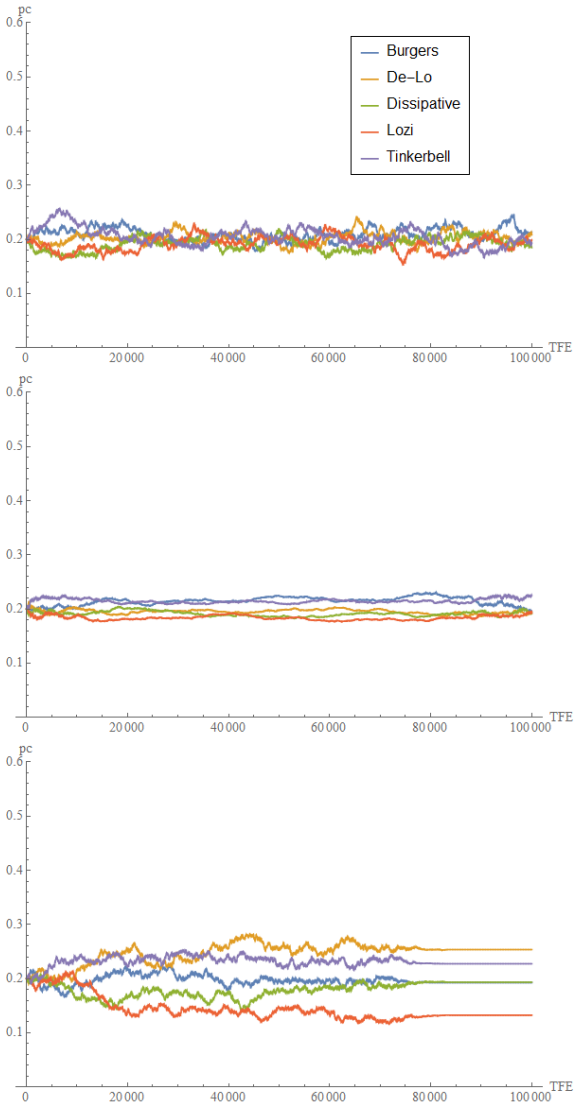

Figure 3. Development of multi-chaotic probabilities over test function evaluations of MC-DE algorithm. From top – Rosenbrock, Rastrigin, Ackley.

4 Experiments and Results

In order to test sensitivity to randomization of non-adaptive and adaptive algorithms, three classic benchmark functions were selected – Rosenbrock (unimodal function with global optima in a narrow, parabolic valley) (14), Rastrigin (complex, multimodal function) (15) and Ackley (multimodal with nearly flat outer region and large hole at the center) (16).

$$f(x) = \sum_{i=1}^{d-1} \left[100\left(x_{i+1} - x_i^2\right)^2 + (x_i - 1)^2\right] \tag{14}$$

$$f(x^*) = 0, x^* = (1, \dots, 1), x_i \in [-2.048, 2.048]$$

$$f(x) = 10d + \sum_{i=1}^{d} \left[x_i^2 - 10\cos(2\pi x_i)\right] \tag{15}$$

$$f(x^*) = 0, x^* = (0, \dots, 0), x_i \in [-5.12, 5.12]$$

$$f(x) = -20e^{-.2\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}} - e^{\frac{1}{d}\sum_{i=1}^{d}\cos(2\pi x_i)} + 20 + e \tag{16}$$

$$f(x^*) = 0, x^* = (0, \dots, 0), x_i \in [-32, 32]$$

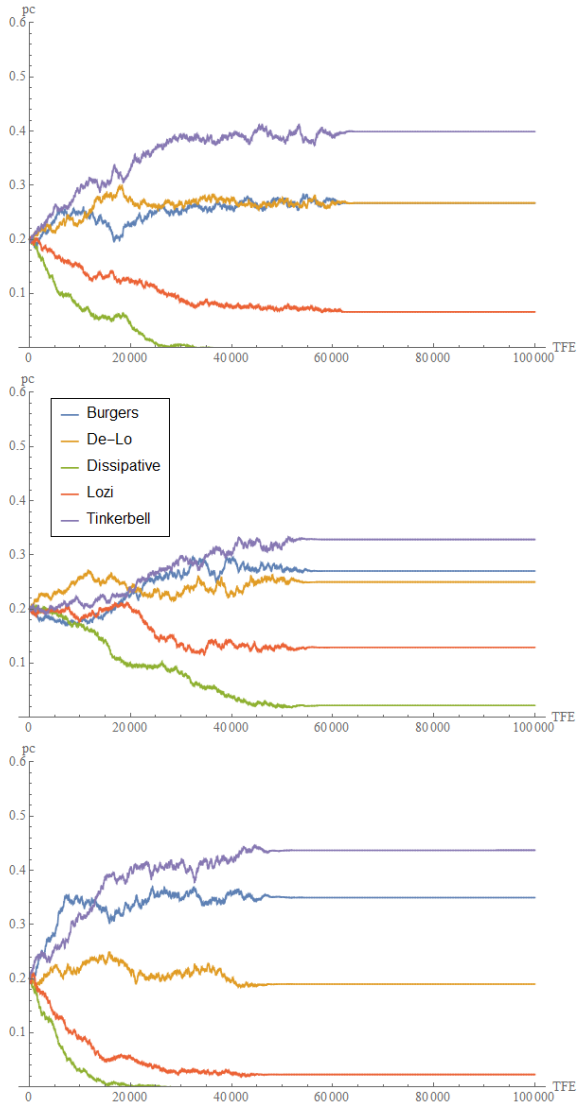

Figure 4. Development of multi-chaotic probabilities over test function evaluations of MC-SHADE algorithm. From top – Rosenbrock, Rastrigin, Ackley.

Both algorithms MC-DE and MC-SHADE were run 51 times on each test function with the maximum number of test function evaluations *maxTFE* set to 100,000. The population size *NP* was set to 100 and 10 dimensional space was selected. The MC-DE control parameters *F* and *CR* were set to 0.5 and 0.8 respectively and MC-SHADEs size of historical memories *H* was set to 10. The multi-chaotic system was initialized with five chaotic maps (Burgers, Delayed Logistic, Dissipative, Lozi and Tinkerbell) with the selection probability pc_{init} = 0.2. The development of the probabilities over the test function evaluations TFE was recorded and all 51 runs were averaged. The average development of probabilities for MC-DE on test functions is shown in Figure 3 and the same is presented for MC-SHADE in Figure 4.

As can be seen in Figure 3, the probabilities of single chaotic maps in non-adaptive algorithm for all three test functions move around the initial values, while as shown in Figure 4, the adaptive algorithm is more sensitive to the randomization system and prefers in each three cases three systems that favor the values close to the left end of specified range for generation. Additionally, in all three cases, the Dissipative chaotic map which was identified to generate values with uniform distribution is strongly suppressed by adaptive algorithm.

5 Conclusions

This paper simulated the effect of adaptivity on randomization on three classic benchmark functions and presented a multi-chaotic framework for parent selection in two DE variants.

In the past, adaptive DE variants outperformed the original DE on numerous benchmarks and real world problems in terms of convergence speed and ability to find the global optimum. Thus, it is important to analyze behavior of such algorithms.

As can be seen in Figure 4, the adaptive algorithm may prefer different randomizations for the selection of parent vectors during the evolutionary process, whereas non-adaptive algorithm seems to be less sensitive and there are mostly none preferred randomizations, which answered the main research question of this paper. The selection of the right randomization or their combination might be beneficial when using adaptive algorithms and the impact has to be studied and analyzed.

The future research will be devoted to thorough analysis of the performance of adaptive algorithms with various randomizations and to development of a robust adaptive randomization system derived from multi-chaotic system presented here.

Acknowledgements

This work was supported by the Ministry of Education, Youth and Sports of the Czech Republic within the National Sustainability Programme Project no. LO1303 (MSMT-7778/2014), further by the European Regional Development Fund under the Project CEBIA-Tech no. CZ.1.05/2.1.00/03.0089 and by Internal Grant Agency of Tomas Bata University under the Projects no. IGA/CebiaTech/2018/003. This work is also based upon support by COST (European Cooperation in Science & Technology) under Action CA15140, Improving Applicability of Nature-Inspired Optimisation by Joining Theory and Practice (ImAppNIO), and Action IC1406, High-Performance Modelling and Simulation for Big Data Applications (cHiPSet). The work was further supported by resources of A.I.Lab at the Faculty of Applied Informatics, Tomas Bata University in Zlin (ailab.fai.utb.cz).

References

BV Babu and M Mathew Leenus Jehan. Differential evolution for multi-objective optimization. In *Evolutionary*

Computation, 2003. CEC'03. The 2003 Congress on, volume 4, pages 2696–2703. IEEE, 2003.

Janez Brest, Sao Greiner, Borko Boskovic, Marjan Mernik, and Viljem Zumer. Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. *IEEE transactions on evolutionary computation*, 10(6):646–657, 2006.

Riccardo Caponetto, Luigi Fortuna, Stefano Fazzino, and Maria Gabriella Xibilia. Chaotic sequences to improve the performance of evolutionary algorithms. *IEEE transactions on evolutionary computation*, 7(3):289–304, 2003.

Nikunj Chauhan, Vadlamani Ravi, and D Karthik Chandra. Differential evolution trained wavelet neural networks: Application to bankruptcy prediction in banks. *Expert Systems with Applications*, 36(4):7659–7665, 2009.

Leandro dos Santos Coelho, Helon Vicente Hultmann Ayala, and Viviana Cocco Mariani. A self-adaptive chaotic differential evolution algorithm using gamma distribution for unconstrained global optimization. *Applied Mathematics and Computation*, 234:452–459, 2014.

John Henry Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

Hong-Kyu Kim, Jin-Kyo Chong, Kyong-Yop Park, and David A Lowther. Differential evolution strategy for constrained global optimization and application to practical engineering problems. *IEEE Transactions on Magnetics*, 43(4):1565–1568, 2007.

Michal Pluhacek, Roman Senkerik, and Ivan Zelinka. Particle swarm optimization algorithm driven by multichaotic number generator. *Soft Computing*, 18(4):631–639, 2014.

Kenneth Price, Rainer M Storn, and Jouni A Lampinen. *Differential evolution: a practical approach to global optimization*. Springer Science & Business Media, 2006.

Roman Senkerik, Michal Pluhacek, and Zuzana Kominkova Oplatkova. An initial study on the new adaptive approach for multi-chaotic differential evolution. In *Artificial Intelligence Perspectives and Applications,* pages 355–362. Springer, 2015a.

Roman Senkerik, Michal Pluhacek, Zuzana Kominkova Oplatkova, and Donald Davendra. On the parameter settings for the chaotic dynamics embedded differential evolution. In *Evolutionary Computation (CEC), 2015 IEEE Congress on*, pages 1410–1417. IEEE, 2015b.

Julien Clinton Sprott and Julien C Sprott. Chaos and time-series analysis, volume 69. Citeseer, 2003.

Rainer Storn and Kenneth Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341–359, 1997.

Ryoji Tanabe and Alex Fukunaga. Success-history based parameter adaptation for differential evolution. In *Evolutionary Computation (CEC), 2013 IEEE Congress on*, pages 71–78. IEEE, 2013.

Jingqiao Zhang and Arthur C Sanderson. Jade: adaptive differential evolution with optional external archive. *IEEE Transactions on evolutionary computation,* 13(5):945–958, 2009.