Scientific original paper

Informacije MIDEM
Journal of Microelectronics,
Electronic Components and Materials
Vol. 42, No. 3 (2012), 152 – 163

Towards Smaller Populations in Differential Evolution

Iztok Fajfar, Tadej Tuma, Janez Puhan, Jernej Olenšek, and Árpád Bűrmen

Faculty of Electrical engineering, University of Ljubljana, Ljubljana, Slovenia

Abstract: Differential evolution is a simple algorithm for global optimization. Basically it consists of three operations: mutation, crossover and selection. Despite many research papers dealing with the first two operations, hardly any attention has been paid to selection nor is there a place for this operation in the algorithm basic naming scheme. In the paper we show that employing certain selection strategies combined with some random perturbation of population vectors notably improves performance in high-dimensional problems. Further analysis of results shows that the improvement is statistically significant. The application of the approach on a real-world case of a simple operating amplifier circuit exhibits a similar behaviour and improvement as observed with the Quartic noisy test function. Due to the nature of the circuit objective function this was expected.

Key words: global optimization, direct search methods, differential evolution, heuristic, parallelization

K majšim populacijam v diferencialni evoluciji

Povzetek: Diferencialna evolucija je preprost algoritem za globalno optimizacijo. Algoritem v osnovi sestavljajo tri operacije: mutacija, križanje in izbor. Čeprav obstaja množica znanstvenih prispevkov, ki obravnava prvi dve operaciji, je tretji operaciji namenjeno komaj kaj pozornosti, niti ni zanjo namenjenega mesta v izvirnem načinu poimenovanja različic postopka. V prispevku pokažemo, da lahko z uporabo različnih postopkov izbora, ki jih kombiniramo z naključno perturbacijo populacijskih vektorjev, opazno izboljšamo delovanje postopka na večrazsežnostnih problemih. S podrobnejšo analizo rezultatov pokažemo, da so izboljšave statistično pomembne. S preizkusom postopka na resničnem primeru preprostega operacijskega ojačevalnika ugotovimo, da se algoritem vede podobno kot na preizkusni funkciji četrtega reda s superponiranim šumom. Glede na naravo kriterijske funkcije vezja smo to pričakovali.

Ključne besede: globalna optimizacija, direktni iskalni postopki, diferencialna evolucija, hevristični postopki, paralelizacija

* *Corresponding Author's e-mail: iztok.fajfar@fe.uni-lj.si*

1. Introduction

Differential Evolution (DE) is a simple yet powerful algorithm for global real parameter optimization proposed by Storn and Price [1]. Through the last decade, the algorithm has gained on popularity among research as well as engineering circles due to its extreme implementation simplicity and good convergence properties. The DE algorithm belongs to a broader class of Evolutionary Algorithms (EA), whose behavior mimics that of the biological processes of genetic inheritance and survival of the fittest. One outstanding advantage of EAs over other sorts of numerical optimization methods is that the objective function needs to be neither differentiable nor continuous, which makes them more flexible for a wide variety of problems.

A DE starts out with a generation of *NP* randomly generated *D*-dimensional parameter vectors. New parameter vectors are then generated by adding a weighted difference of two population vectors to a third vector. This operation is called mutation. One then mixes the mutated vector parameters with the parameters of another vector, called the target vector, to obtain the so-called trial vector. The operation of parameter mixing is usually called crossover in the EA community. Finally, the trial vector is compared to the target vector, and if it yields a better solution, it replaces the target vector. This last operation is referred to as selection. In each generation, each population vector is selected once as the target vector.

There exist several variants of the DE algorithm [2, 3, 4, 5, 9], of which the most commonly used is *DE/rand/1/bin* which we explore in this paper. Before using the algorithm, one has to decide upon the values of three parameters affecting the behavior of a DE. The first is the population size *NP*, the other two are control parameters – a scaling factor *F*, and a crossover rate *CR*. Choosing the values of these parameters is usually

a problem-dependent task requiring a certain user expertise. Researchers have attempted to tackle the problem using several adapting and self-adapting strategies to govern the values of the control parameters *F* and *CR* [6, 7, 8, 9 and the references within] and even the population size *NP* [10, 11, 12, 13, 14]. Others have proposed and studied different mutation and crossover strategies [15, 16, 17, 18]. No explicit research work has been done so far on the third of the DE operators, the selection, neither is there any intended place in the algorithm variant naming scheme (i.e. DE/x/y/z) for this operator. In this paper we investigate how different selection schemes affect the behavior of the DE algorithm, in particular its ability to escape the local minima or stagnation. In addition to that we applied what would in genetic algorithm be called mutation, i.e. we randomly changed the population vector parameters with a fixed probability. Since the term mutation is already reserved in DE, we named this operation a random perturbation.

In the next section, we shortly describe the functioning of the basic DE algorithm, and in Section 3 we propose a random vector perturbation and different selection algorithms that we investigate. Finally, we present some results of optimizing test functions and a real electronic circuit in Sections 4 and 5, respectively.

2. Differential Evolution Overview

Consider the objective (criterion) or *fitness* function $f: \mathrm{R}^n \rightarrow \mathrm{R}$, where one has to find a minimum $\boldsymbol{a} \in \mathrm{R}^n$ so that $\forall \boldsymbol{b} \in \mathrm{R}^n : f(\boldsymbol{a}) \leq f(\boldsymbol{b})$. In this case $\boldsymbol{a}$ is called a global minimum. It is rarely possible to find an exact global minimum in real problems, so for practical reasons one must accept a candidate with a reasonable good solution.

In order to search for a global minimum, differential evolution utilizes *NP D*-dimensional parameter vectors $\boldsymbol{x}_{i,G}$, i=1,2,...,*NP* as a population in generation *G*. *NP* does not change from generation to generation. The initial population is chosen randomly and – if no prior information about the fitness function is known – it should cover the entire search space uniformly.

During the optimization process, the new parameter vectors are generated by adding a weighted difference of two randomly chosen population vectors to a third vector: $\boldsymbol{v}_{i,G+1}=\boldsymbol{x}_{r1,G}+F\cdot(\boldsymbol{x}_{r2,G}-\boldsymbol{x}_{r3,G})$ with integer, mutually different, random indices $r_1,r_2,r_3 \in \{1,2,...,NP\}$, which must all be different from i as well, and a real constant factor $F \in [0,2]$. This operation is called *mutation* and the thus obtained vector the *mutated* vector.

The mutated vector parameters are then mixed with another vector, the so-called *target* vector, in order to produce a *trial* vector $\boldsymbol{u}_{i,G+1}=(u_{1i,G+1},u_{2i,G+1},...,u_{Di,G+1})$ where

$$u_{ji,G+1} = \begin{cases} v_{ji,G+1} & \text{if}(randb(j) \leq CR) \text{ or } j = rnbr(i) \\ x_{ji,G} & \text{if}(randb(j) > CR) \text{ and } j \neq rnbr(i) \end{cases}, \quad j = 1,2,\ldots,D. \tag{1}$$

Here, $randb(j) \in [0,1]$ is the jth execution of the uniform random generator, $CR \in [0,1]$ is user-determined constant, and $rnbr(i) \in \{1,2,...,D\}$ is a random index. The latter insures that the trial vector gets at least one parameter from the mutated vector. This operation of parameter mixing is usually called *crossover* in evolutionary search community.

Finally, a *selection* is performed in order to decide whether or not the trial vector should become a member of generation *G*+1. The value of the fitness function at the trial vector $\boldsymbol{u}_{i,G+1}$ is compared to its value at the target vector $\boldsymbol{x}_{i,G}$ using the greedy criterion. Only if the trial vector yields a better fitness value than the target vector, the target vector is replaced. Otherwise the trial vector is discarded and the target vector retained.

3. Random Perturbation and Different Selection Strategies

We focus our work on the stage of the DE algorithm after crossover, i.e. on the stage when the trial vector is already fully formed.

The idea for our modification came first from a simple observation that with a crossover rate *CR* approaching 1 not much of the target vector survives in its offspring (trial vector). In that sense one can argue that the search direction from the target to the trial vector can be as good (or as bad) as any other direction. The hypothesis we want to test is that there might exist some other (possibly better) candidate for replacement than the target vector itself.

In what follows, we propose and separately test three different rules for selecting the candidate to replace the trial vector. We select that candidate according to one of the three selection algorithms.

Algorithm 1: Replace the vector closest to the target vector.

Input: trial vector $\boldsymbol{u}_{i,G+1}$, target vector $\boldsymbol{x}_{i,G}$, Gth generation of NP parameter vectors $\boldsymbol{x}_{n,G}$, n=1,2,...,NP
$c = -1$
$d_{\min} = \infty$
for $n = 1$ **to** NP **do**
  if $f(\boldsymbol{u}_{i,G+1}) < f(\boldsymbol{x}_{n,G})$ **and** $d(\boldsymbol{x}_{i,G}, \boldsymbol{x}_{n,G}) < d_{\min}$ **then**
    $c = n$
    $d_{\min} = d(\boldsymbol{x}_{i,G}, \boldsymbol{x}_{n,G})$
  endif
endfor
if $c \neq -1$ **then**
  $\boldsymbol{x}_{c,G}$ is replaced by $\boldsymbol{u}_{i,G+1}$
endif

The notation $d(\cdot,\cdot)$ in Algorithm 1 denotes an Euclidean distance. The algorithm replaces, of all the vectors that yield a worse fitness value than the trial vector, the one that is geometrically closest to the target vector. Notice that this strategy, the same as the original algorithm, always replaces the target vector as long as it is worse than the trial vector. Otherwise, it seeks after the candidate which is closest possible to the target vector to replace it. If no such vector is found, then the trial vector is discarded. As in the original algorithm, the target vectors with a relatively bad fitness value will be replaced more likely, while those with a better fitness value will survive. In addition to that, however, some near vector is moved to the place where the target vector would move if the target vector were not worse than the trial vector. This speeds up the clustering of the population members around the members with generally better fitness values. On one hand this can accelerate the convergence significantly, on the other hand, however, there is a danger of losing a necessary diversity too soon and thus not finding a global solution.

Algorithm 2: Replace the vector closest to the trial vector.

Input: trial vector $\boldsymbol{u}_{i,G+1}$, Gth generation of NP parameter vectors $\boldsymbol{x}_{n,G}$, n=1,2,...,NP
$c = -1$
$d_{\min} = \infty$
for $n = 1$ **to** NP **do**
  if $f(\boldsymbol{u}_{i,G+1}) < f(\boldsymbol{x}_{n,G})$ **and** $d(\boldsymbol{u}_{i,G+1}, \boldsymbol{x}_{n,G}) < d_{\min}$ **then**
    $c = n$
    $d_{\min} = d(\boldsymbol{u}_{i,G+1}, \boldsymbol{x}_{n,G})$
  endif
endfor
if $c \neq -1$ **then**
  $\boldsymbol{x}_{c,G}$ is replaced by $\boldsymbol{u}_{i,G+1}$
endif

The approach with Algorithm 2 is quite different in that it searches for the candidate that is geometrically closest to the trial vector instead of the target vector. In that sense replacements are made that favor smaller jumps and encourage searching over less promising areas as well.

Algorithm 3: Replace either the target vector or the first one of the first half of the population that is worse than the trial vector.

Input: trial vector $\boldsymbol{u}_{i,G+1}$, target vector $\boldsymbol{x}_{i,G}$, Gth generation of NP parameter vectors $\boldsymbol{x}_{n,G}$, n=1,2,...,NP
if $f(\boldsymbol{u}_{i,G+1}) < f\,\boldsymbol{x}_{i,G}$) **then**
  $c = i$
else
  $c = -1$
  for $n = 1$ **to** $NP/2$ **do**
    if $f(\boldsymbol{u}_{i,G+1}) < f(\boldsymbol{x}_{n,G})$ **then**
      $c = n$
      exit_for_loop
    endif
  endfor
endif
if $c \neq -1$ **then**
  $\boldsymbol{x}_{c,G}$ is replaced by $\boldsymbol{u}_{i,G+1}$
endif

The construction of Algorithm 3 is not so obvious at the first glance. Similarly to the original algorithm and Algorithm 1, one first checks whether the target vector is to be replaced, i.e. if the trial vector yields a better fitness value than the target vector. Otherwise we replace the first member of the first half of the population whose fitness value is worse than that of the trial vector. The idea behind that is to have one half of the population evolve under the original DE rules while accelerating the other half with further replacements. Even these additional replacements are applied asymmetrically with the members with a smaller index affected more often. That way we wanted to induce as little a change to the original method as possible, while inducing a relatively strong drag on a limited number of population members. The 1:1 ratio between both parts of the population was chosen arbitrarily. It should be noted that more frequent replacements lead towards a faster loss of diversity in population, which in turn lessens a chance to find the global minimum, and we wanted to find the equilibrium between two usually conflicting goals, namely fast convergence and high probability of finding the global optimum. The randomization introduced in the remainder of this section is supposed to make up for the before mentioned loss of diversity and in the same time indirectly to fine tune the ratio between two parts of the population.

Before going into experiments, let us introduce one more tiny though important modification to the algorithm. It is interesting to notice that although the algorithm itself belongs to a class of metaheuristics and stochastic optimization, the randomness in the original concept is only used for the selection of the vectors from which the mutated vector will be formed and for mixing the mutated and target vector parameters. The vector parameters themselves are changing randomly only indirectly through the mutation and crossover, and the obtained values are limited to a set of linear combinations of parameters already contained in a population. Some authors have already introduced some more randomness into DE, either directly by randomization of the vector parameters [19, 20] or indirectly by randomizing the algorithm control parameters *F* and *CR* [14, 21, 22], thus increasing the explorational potential of the algorithm, and even making it possible to prove the convergence [20].

In our study we decided simply to mutate every single parameter of the trial vector with a fixed probability just before the selection procedure takes place:

$$u_{ji,G+1} = \begin{cases} rand(j), \text{if}(randb(j) \leq 0.005) \\ u_{ji,G+1}, \text{otherwise} \end{cases}, \; j = 1,2, \ldots, D, \tag{2}$$

where *rand*(*j*) is the call of the random generator that returns the uniformly distributed values along the entire *j*th axis of the parameter space. The constant probability of 0.005 was obtained empirically by a few preliminary test runs of the algorithm, which also indicated that the uniform distribution over the whole parameter space yielded somewhat superior performance compared to a normal distribution around the current parameter value often used in literature. We call this operation *perturbation*.

4. Experiments on test functions

Overall Performance

In order to get an overall picture and the first impression of the impact of the three proposed selection strategies and random vector perturbations, we carried out a simple test. For testing purposes, fourteen standard benchmark functions from [23] were selected, thirteen high-dimensional (D=30) and one low-dimensional (D=4) function. Then we randomly selected the three parameters from the intervals $NP \in \{10,...,100\}$, $F \in [0,1]$, and $CR \in [0,1]$, and initialized a random population of the *NP* parameter vectors lying within the boundaries given for the test function in question. Next we executed eight optimization runs of the 150,000 criterion function evaluations (CFEs) with the same parameter values and initial vector population, but each time applying either the original or one of the three proposed selection schemes, once without and once with a random perturbation. We repeated this 5,000 times for each test function, each time with different control parameter values and initial vector population. The results are summarized in Table 1.

Table 1: Comparison of different modifications of the algorithm with the original

Selection Method	Without Perturbation		With Perturbation	
	50,000 CFEs	150,000 CFEs	50,000 CFEs	150,000 CFEs
Original	–	–	44.1/51.7	43.7/44.1
Algorithm 1	53.5/43.4	45.1/48.0	69.9/26.9	63.1/29.0
Algorithm 2	52.8/44.1	45.7/47.4	62.4/34.4	57.3/35.4
Algorithm 3	61.4/34.6	53.0/37.1	75.2/20.6	68.5/20.5

The fourteen pairs of the numbers in the table stand for the seven different comparisons (each of the modifications separately compared to the original) at two different times of the algorithm run: after 50,000 CFEs and after 150,000 CFEs. The numerator represents the percentage of cases in which the corresponding modification yielded a better fitness value (at the precision of 6 significant digits) than the original, while the denominator speaks of the percentage of cases in which the original method performed better. The sum is generally smaller than 100, because in some cases both variants gave the same result. The counting was carried out over all runs regardless of the control parameter setting or the selected test function. In real-life problems, often the practitioner has little or no knowledge about the fitness function and consequently about the best control parameter settings. Therefore, it seems that averaging over a range of different test functions and control parameter settings, selected in the Monte-Carlo manner, is an appropriate measure of the algorithm overall performance.

In the table, the pairs of the numbers in the white cells represent the state after 50,000 CFEs. We conjectured that at this optimization stage the convergence is generally not yet fully reached. Consequently, those pairs of the numbers indicate the convergence speed rather than the overall ability to find a global minimum.

The numbers in the shaded cells represent the state after 150,000 CFEs, when we assume that the number of the cases reaching the final solution is considerably larger than of those after 50,000 CFEs. Hence we con-

sider these results to reflect the ability of the algorithm to find a good final solution.

From the table one can infer some quite interesting observations. Replacing – instead of target vector – the candidate closest to target (Algorithm 1) or closest to trial vector (Algorithm 2) without using perturbation performed just slightly better after 50,000 CFEs (1st column, 2nd and 3rd row, respectively) and slightly worse after 150,000 CFEs (2nd column, 2nd and 3rd row, respectively). That implies that the more frequent replacements in both cases speed up the convergence as expected but, in general, they cause the algorithm more often to stuck in one of the local minima or to reach stagnation in the end. That is, however, not the case with the selection strategy using Algorithm 3. This strategy outperformed the original for almost twice more cases after 50,000 CFEs and still remained much better after 150,000 CFEs (4th row, 1st and 2nd column, respectively). It is important to note that with this kind of modification the algorithm still performs more replacements than the original one, which obviously speeds-up the convergence. The main difference here is that we perform these additional replacements only on a limited number of the population members, the others still undergoing the original selection scheme. Technically, we can speak of two different schemes running in parallel.

Comparing the original method with and without perturbations gives us no noticeable difference (1st row, 3rd and 4th column). As reasonably expected, random perturbations slow down convergence to some extent (1st row, 3rd column), but in the long run no variant outperforms the other (1st row, 4th column). It is quite interesting to notice that while perturbation seems to have no observable effect when applied to the original algorithm, it improves the other three variants noticeably. It seems that in these cases perturbation not only makes up for the loss of the population variance – which might have occurred due to a too fast convergence induced by more frequent replacements – but also improves the overall performance. It seems that the changed selection schemes and random perturbations support each other. Nevertheless, comparing the results of the selection algorithms 1, 2, and 3 with perturbation after 50,000 and 150,000 CFEs shows that in all the three cases, in the long run, the original method compensates a little for the much worse performance during the first part of the run. This leaves us, possibly, some room for improvement by balancing the factors that affect the convergence speed and the rate of change in the population diversity.

A Closer Look

Let us now focus a little closer on Algorithm 3 combined with vector perturbation exhibiting the best overall improvement in the previous analysis. In order to get a more accurate picture, we made the same comparisons as before, only this time for each test function separately. The results are summarized in Table 2. The table shows comparisons of the original method with the original method with perturbation, and with Algorithm 3 with and without perturbation. The numbers in normal writing represent the state after 50,000 CFEs, while the ones in boldface the state after 150,000 CFEs.

Table 2: Comparison of different modifications by separate test functions

Test Function	Modification Compared to the Original Algorithm		
	Original with Perturbation	Algorithm 3	Algorithm 3 with Perturbation
f_1 (Quadratic)	34.72/65.28 **31.60/68.40**	70.83/29.17 **68.75/31.25**	80.21/19.79 **76.39/23.61**
f_2 (Schwefel 2.22)	33.45/66.55 **31.71/68.29**	70.73/29.27 **66.90/33.10**	74.22/25.78 **70.03/29.97**
f_3 (Schwefel 1.2)	51.04/48.26 **54.51/45.49**	75.35/23.96 **70.49/29.51**	77.08/22.57 **78.47/21.53**
f_4 (Schwefel 2.21)	49.48/48.08 **48.43/49.83**	63.07/34.49 **59.58/39.72**	83.97/12.80 **79.79/18.82**
f_5 (Generalized Rosenbrock)	49.83/50.17 **52.96/47.04**	58.19/41.81 **56.10/43.90**	77.00/23.00 **73.87/26.13**
f_6 (Step)	31.36/24.39 **29.97/9.76**	31.36/27.87 **18.12/26.48**	47.04/6.62 **35.19/3.48**
f_7 (Quartic noisy)	46.50/53.50 **49.30/50.70**	70.28/29.72 **67.83/32.17**	77.97/22.03 **77.62/22.38**
f_8 (Generalized Schwefel 2.26)	57.14/42.86 **58.54/29.62**	49.83/50.17 **36.59/58.19**	82.58/17.42 **78.75/11.15**
f_9 (Generalized Rastrigin)	50.69/48.96 **49.65/43.40**	66.67/33.33 **57.64/39.24**	80.90/19.10 **79.17/15.28**
f_{10} (Ackley)	48.26/50.69 **48.96/33.33**	60.07/39.24 **43.75/41.67**	81.60/17.36 **68.40/15.63**
f_{11} (Generalized Griewank)	32.17/61.19 **31.47/36.71**	63.99/29.72 **42.31/29.02**	73.08/20.28 **53.50/17.83**
f_{12} (Generalized penalty function 1)	43.36/56.29 **36.01/45.80**	68.53/31.12 **52.45/33.22**	81.47/18.53 **65.38/20.63**
f_{13} (Generalized penalty function 2)	45.10/54.90 **42.66/43.01**	62.59/37.06 **50.35/37.06**	82.17/17.48 **72.03/15.03**
f_{13} (Kowalik)	44.41/52.45 **45.80/46.50**	48.25/47.55 **50.70/45.10**	52.80/45.80 **49.65/45.80**

The first and foremost important observation here is that the modification combined with perturbation shows noticeably and consistently better performance

in all cases except for the Kowalik test function where there is no observable difference. Again we see that perturbation alone does not really improve performance of the original method, two notable exceptions being the Schwefel 2.26 and Step functions.

The Schwefel function is somehow tricky in that the global minimum is placed geometrically remote from the next few best local minima. The original method exhibits quite a good convergence at the beginning, while later on perturbations help find the global minimum as without them the original method would be stuck in a local minimum (see the 1st column, Schwefel 2.26 function). Interestingly enough, modification without perturbation in that case performs much worse than the original method. This probably stems from the fact that this method replaces candidates of one half of the population excessively, thus additionally forcing the population in one of the local minima. The modified method with perturbation, however, performs much better in this case.

The same goes for the step function. This function, too, poses some difficulties for the original algorithm because it consists of many plateaus and discontinuities. All points within a small neighborhood will have the same fitness value, making it very difficult for the process to move from one plateau to another. Perturbations seem to help here significantly.

Up to now we were only interested in the number of cases in which one method is better than the other. What about the fitness values they actually produce? In Table 3 one finds the best fitness values averaged over all runs, comparing the original algorithm with the one using Algorithm 3 with perturbations. Notice that the values are quite large. One must not forget that these values were obtained by running the optimization using completely random optimization parameters. So many of the parameter values were used that were not even close to the recommendations in the literature. But as we are interested only in the differences between different algorithm variants, this is not an issue.

In the first and second column of Table 3 one finds the average minimuns for each of the two variants, while in the last column there are the results of paired two--sampled two-tailed t-test of comparing the modified approach with the original.

Table 3: Comparison of the average best fitness obtained by the original algorithm and Algorithm 3 with perturbation

Test Function	Original (Average Best Fitness)	Algorithm 3 with Perturbation (Average Best Fitness)	T-Test Values	Analytical (actual) best fitness
f_1 (Quadratic)	1.694659×10^{3}	2.338428×10^{2}	p<0.01 t=5.09	0
f_2 (Schwefel 2.22)	6.908752	2.819474	p<0.01 t=5.47	0
f_3 (Schwefel 1.2)	1.429065×10^{4}	8.331654×10^{3}	p<0.01 t=13.95	0
f_4 (Schwefel 2.21)	17.24446	7.910773	p<0.01 t=10.69	0
f_5 (Generalized Rosenbrock)	2.407282×10^{6}	2.566153×10^{5}	p<0.01 t=3.17	0
f_6 (Step)	1.315575×10^{3}	2.352474×10^{2}	p<0.01 t=4.11	0
f_7 (Quartic noisy)	1.747322	0.1379149	p<0.01 t=4.40	0
f_8 (Generalized Schwefel 2.26)	-1.069594×10^{4}	-1.167699×10^{4}	p<0.01 t=10.75	-12569.5
f_9 (Generalized Rastrigin)	71.73974	49.26490	p<0.01 t=11.05	0
f_{10} (Ackley)	6.395496	2.192476	p<0.01 t=10.63	0
f_{11} (Generalized Griewank)	10.83875	1.257148	p<0.01 t=4.22	0
f_{12} (Generalized penalty function 1)	4.372791×10^{6}	7.482104×10^{5}	p=0.02 t=2.27	0
f_{13} (Generalized penalty function 2)	9.736573×10^{6}	1.050585×10^{6}	p<0.01 t=2.786	0
f_{15} (Kowalik)	1.358227×10^{-3}	1.836926×10^{-3}	p=0.11 t=-1.61	0.0003075

The test further confirms our speculations about the modification bringing significant advantages over the original DE. In twelve out of the fourteen test functions the modification performs significantly better at a 99% significance level. Again, an exception is the Kowalik function where the test actually indicates a degradation of performance (observe the negative t-value), although not a significant one.

Parameter Impact

In our experiments so far we didn't pay any attention to the actual control-parameter or population-size selection. The values were picked up completely randomly

within the set intervals. In this section we want to investigate the effect of different parameter settings on the algorithm performance with the proposed modifications. We compare the original algorithm to the one using Algorithm 3 with perturbation. In this paper we summarize the results only for the case of Generalized Schwefel 2.26 function. It should be noted, however, that we observed a similar behavior elsewhere as well [28].

We started out by choosing the control parameter settings most commonly found in literature, i.e. $F = 0.5$ and $CR = 0.9$. Our experimenting showed that at these values the best fitness (assuming a fixed number of 150,000 CFEs) is generally obtained at the population size $NP = 40$. The results in this section are obtained by changing one of the three values while keeping the other two fixed. The best fitness values were averaged over 25 independent runs.

Figure 1 shows that the original method completely failed to reach the global minimum in the Schwefel 2.26 function (at –12569.5) safe for the lowest values of *CR*. Interesting, however, is that the modification enables DE to find the global minimum at lower and higher values of *CR*, but not at the values around 0.7. A similar behavior can be observed in other functions as well [28], where the modification brings an improvement at the smaller and bigger values of *CR*.

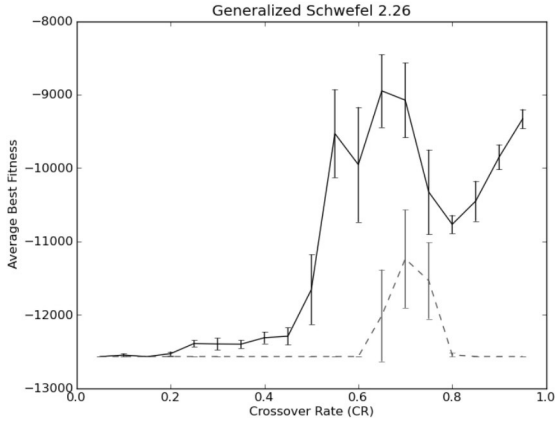

Figure 1: Impact of the crossover rate (*CR*) on the algorithm performance. The solid (black) line shows the best fitness values averaged over 25 runs, as obtained by the original DE. The dashed (red) line shows the results produced by DE with Algorithm 3 and perturbation. The graph shows a situation at the fixed *F*=0.5 and *NP*=40.

Figure 2 shows the results for the same test function using constant *CR*=0.9 and *NP*=40, while changing *F* from 0.05 to 0.95. The general pattern that is to be observed is somewhat different from the observations with the changing parameter *CR*. We observe the major improvement at the smaller *F* values. It has been mentioned in the literature that *F* must not be too small in order to be able to find a minimum [25]. A small *F* means a small difference vector and hence a small displacement of a mutated vector. It seems that too big a displacement is not good for the convergence either. An interesting observation is that our modification improves the results more at the end of smaller *F* values, and often to the extent that outperforms the original algorithm at any other *F* value. It seems that small displacements – which seem to produce a slow but stable convergence – go hand in hand with the anticipated speed up caused by our modification, together producing a more stable and faster convergence.

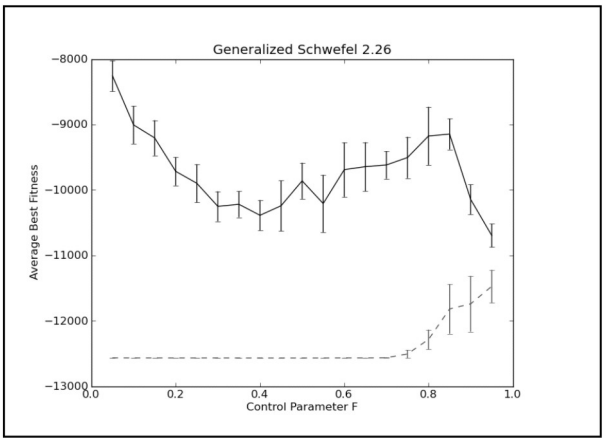

Figure 2: Impact of the control parameter *F* in the Generalized Schwefel 2.26 function. The graph shows a situation at the fixed *CR*=0.9 and *NP*=40.

In Figure 3, which depicts the impact of the population size, we can see that the major improvement is achieved at lower population sizes. This effect is especially evident in case of Schwefel 2.26 function when the minimum is reached with our modification but not with the original method.

The large population size in DE usually guarantees a larger probability of finding the global minimum and, originally, the proposed population size was $NP = 10D$ [24]. Other sizes were proposed later but were all considerably greater than the fitness function dimensionality *D*. As clearly seen from Figure 3 and the graphs in [28], at larger population sizes our modification does not bring any improvement over the original method whatsoever. That is somehow expected since the DE should be quite stable at larger *NP*. The problem however is that the stability is of no great practical use if after a relatively large number of CFEs the algorithm is still very far from the actual solution. We see one of the

strongest values of our modification in having instead of one large population many smaller ones running in parallel which could bring together the ability to actually find the global minimum and speed up of convergence.

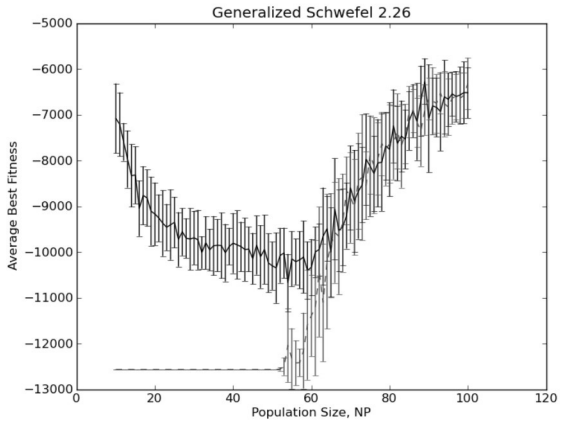

Figure 3: Impact of the population size in the Generalized Schwefel 2.26 function. The graph shows a situation at the fixed *CR*=0.9 and *F*=0.5.

5. Test runs on a simple operating amplifier circuit

The Circuit

After successfully testing the proposed algorithm on standard benchmark functions, we wanted to see whether the method behaved in the same fashion on a real circuit as well. As an example we look at a simple two-stage operating amplifier, whose circuit diagram is shown in Fig 4. The amplifier is designed in a 0.18 µm CMOS process with 1.8 V supply voltage.

The operating point of the amplifier is determined by the input bias current flowing into the drain terminal of Xmn1b. Xmn1 and Xmn4 mirror this current to set the bias of the differential pair (Xmn2 and Xmn3) and output amplifier (Xmp3). Transistors Xmp1 and Xmp2 act as active load to the differential pair. Frequency compensation is introduced by R_{out} and C_{out}. Ports *inp*, *inn*, and *out* represent the noninverting input, inverting input, and output of the amplifier, respectively. Transistors Xmn1s and Xmp1s power down the amplifier when signals slp (slpx) are pulled high (low).

The testbench circuit shown in Fig 5 provides supply voltage (V_{dd}) to the amplifier along with a 100 µA bias current. Feedback is introduced by resistors R_{fb} and R_{in}. R_{load} and C_{load} represent the load resistance and capacitance. Because the supply voltage is single-ended, the input signal (V_{in}) requires a common mode bias ($V_{com}=V_{dd}/2$).

During the optimization we simulated the circuit across three corners: nominal (nominal PMOS and NMOS model, 25° C, V_{dd}=1.8 V), worst power (fast NMOS and PMOS model, 100° C, V_{dd}=2.0 V), and worst speed (slow NMOS and PMOS model, 0° C, V_{dd}=1.8 V). In each of the corners we performed the following analyses: operating point, DC sweep of input voltage (from –2 V to 2 V), DC sweep of common mode bias (from 0.7 V to V_{dd}

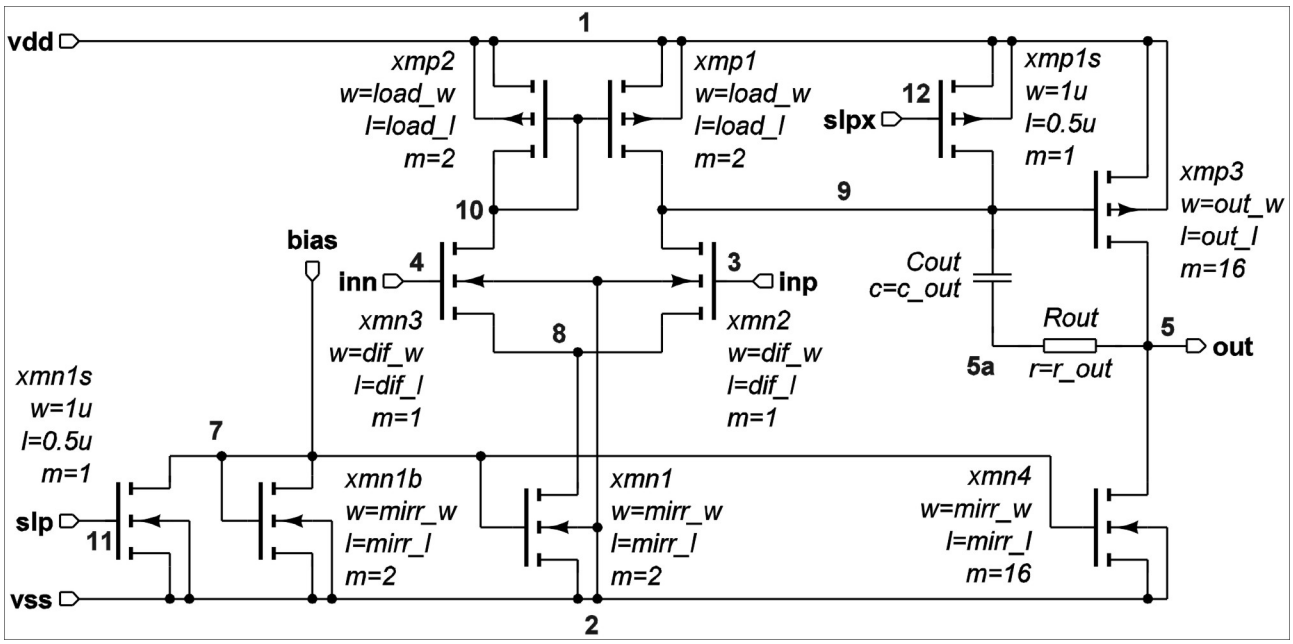

Figure 4: The circuit diagram of a simple two stage operating amplifier.

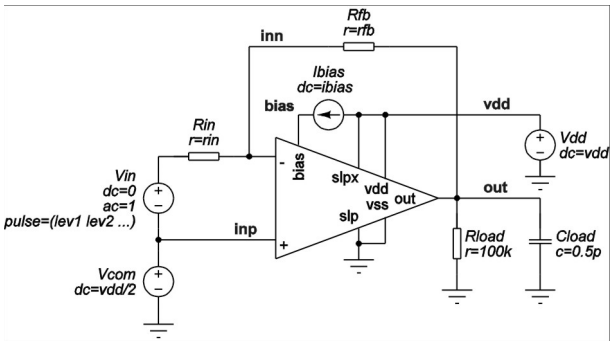

Figure 5: The testbench circuit for the operating amplifier from Fig 4.

– 0.1 V), small signal AC analysis, and transient analysis of the response to a ±0.5 V step in input voltage. R_{fb} and R_{in} were both set to 1 MΩ, except in transient analysis where R_{in} was set to 100 kΩ.

From the obtained results of above analyses the following performance measures were derived:

- V_{gs} and V_{ds} overdrive voltage (i.e. V_{gs}–V_t and V_{ds}–V_{dsat}) for all transistors, except Xmn1s and Xmp1s, at operating point and at all points analyzed in the DC sweep of common bias,
- output voltage swing where DC gain is above 50% of maximal gain,
- gain, unity-gain bandwidth (UGBW), and phase margin,
- overshoot, undershoot, 10% - 90% rise and fall time, 5% settling time, and slew rate of transient response,
- total gate area of all transistors (except Xmn1s and Xmp1s).

Next, from these performance measures, we constructed the cost function [26] consisting of the following requirements:

- overdrive voltage should be at least 1m V,
- output voltage swing must be greater than 1.6 V,
- gain, UGBW, and phase margin must lie above 75 dB, 60 MHz, and 60°, respectively,
- overshoot and undershoot must be below 10%, rise and fall time below 100 ns, settling time below 300 ns, and slew rate above 10 V/µs,
- total gate area should be less than 1500 µm².

Thus constructed cost function was used for guiding the optimization algorithm which should find optimal adjustment of the 10 circuit parameters listed in Table 4.

Table 4: Definitions of the circuit parameters subject to optimization

Parameter	Description	Range	Unit
dif_w	channel width of Xmn2 and Xmn3	[1, 95]	µm
dif_l	channel length of Xmn2 and Xmn3	[0.18, 4]	µm
load_w	channel width of Xmp1 and Xmp2	[1, 95]	µm
load_l	channel length of Xmp1 and Xmp2	[0.18, 4]	µm
mirr_w	channel width of Xmn1b, Xmn1 and Xmn4	[1, 95]	µm
mirr_l	channel length of Xmn1b, Xmn1 and Xmn4	[0.18, 4]	µm
out_w	channel width of Xmp3	[1, 95]	µm
out_l	channel length of Xmp3	[0.18, 4]	µm
c_out	capacitance of C_{out}	[0.01, 10]	pF
r_out	resistance of R_{out}	[0.001, 200]	kΩ

Performance measures were evaluated in all corners, except for the V_{gs} and V_{ds} overdrive voltages, which were evaluated only in the nominal corner. The cost function was expressed as a sum of contributions

$$F = \sum_{i=1}^{m} f_i$$

where m is the number of performance measures. Every performance measure resulted in one contribution f_i, which was obtained by transforming its worst value observed across all corners ($y_i(\boldsymbol{x})$) using a piecewise-linear function $f(y_i(\boldsymbol{x}), g_i, n_i, p_i, c_i)$. Here g_i, n_i, p_i, and c_i denote the goal, the norm, the penalty factor, and the tradeoff factor, respectively. For requirements of the form $y_i(\boldsymbol{x}) \le g_i$ the contribution was computed as

$$f_i = \begin{cases} \frac{p_i(y_i(\boldsymbol{x}) - g_i)}{n_i} & \text{if } y_i(\boldsymbol{x}) > g_i \text{ (i.e. a requirement is not fulfilled)} \\ f(y_i(\boldsymbol{x}), g_i, n_i, p_i, c_i) = \frac{t_i(y_i(\boldsymbol{x}) - g_i)}{n_i} & \text{if } y_i(\boldsymbol{x}) \le g_i \text{ (i.e. a requirement is fulfilled)} \end{cases}$$

For requirements of the form $y_i(\boldsymbol{x}) \ge g_i$ the contribution was

$$f_i = \begin{cases} \frac{p_i(g_i - y_i(\boldsymbol{x}))}{n_i} & \text{if } y_i(\boldsymbol{x}) < g_i \text{ (i.e. a requirement is not fulfilled)} \\ f(y_i(\boldsymbol{x}), g_i, n_i, p_i, c_i) = \frac{t_i(g_i - y_i(\boldsymbol{x}))}{n_i} & \text{if } y_i(\boldsymbol{x}) \ge g_i \text{ (i.e. a requirement is fulfilled)} \end{cases}$$

The tradeoff factor for overdrive voltages was set to $t_i = 0$, because we are not interested in improving them beyond their respective goals. All other tradeoff factors were set to $t_i = 0.001$ and the penalty factors were set to $p_i = 1$. The norms were set to g_i / 10 for all performance measures, except for the area, where $n_i = 100$ µm² was used.

Optimization Results

We ran the circuit optimization using the same DE parameters as with the test functions in the previous section. Simple circuits such as the one in question often exhibit unimodal objective function contaminated with numerical noise. Therefore we expected the results to be reminiscent of those obtained with the Quartic noisy test function [28].

During each optimization run we performed, as with test functions, 150,000 CFEs. We run the optimizations on 20 2.66 Ghz Core i5 (4 cores per machine) machines, and it took approximately 3 weeks to complete the computation. Unlike with most test functions, the study of the results showed us that already after 20,000 CFEs there were no observable changes neither in obtained average fitness values nor standars errors. From relatively flat lines with almost zero standard error at fitness value of 0.56 we conclude that the minimum actually lies at that value (cf. Figs 6 to 8).

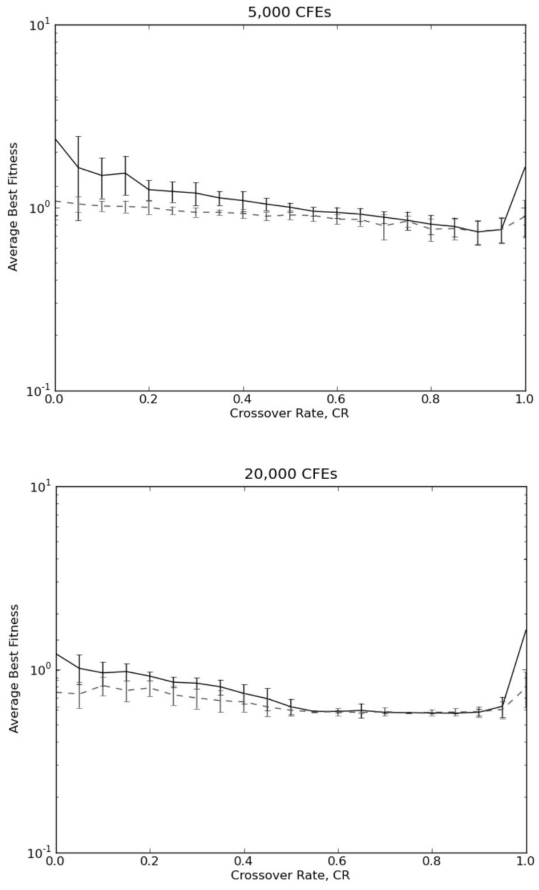

Figure 6: Impact of the crossover in optimizing the simple operating amplifier circuit, after 5,000 CFEs (left) and 20,000 CFEs (right). The graph shows a situation at the fixed *F*=0.5 and *NP*=40.

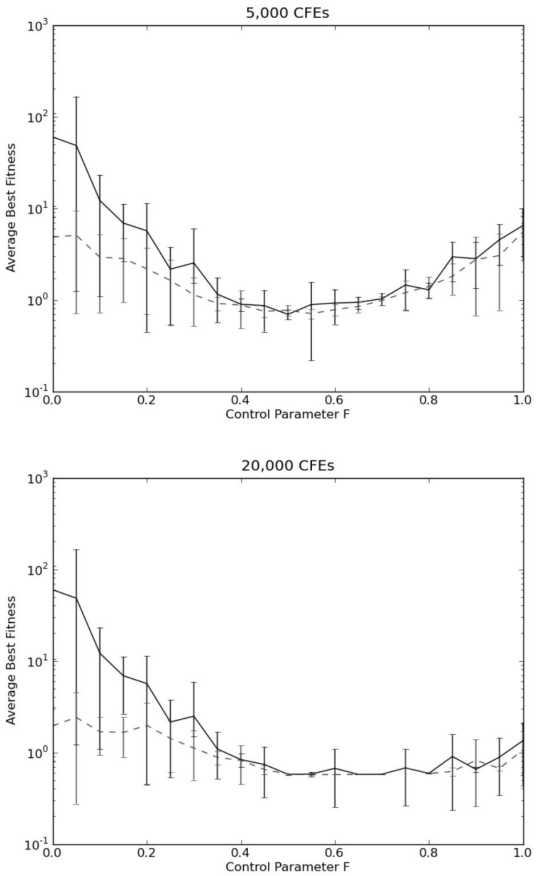

Figure 7: Impact of the control parameter *F* in optimizing the simple operating amplifier circuit, after 5,000 CFEs (left) and 20,000 CFEs (right). At the values of *F*=0.2 and below there was practically no change after 5,000 CFEs with the original algorithm, while some further improvement could be observed with Algorithm 3 with perturbation. The graph in this case is comparatively flat, showing lessened sensitivity to control parameter *F*. The graph shows a situation at the fixed *CR*=0.9 and *NP*=40.

Similarly to the results with Quartic noisy function, our modification does not improve the best results obtained with the original DE. The very important observation however, is the fact that it does not worsen the best results either, and the resulting fitness values are quite better at the control parameter values where the original DE did not perform very well. In that sense one can argue that applying Algorithm 3 with perturbatuion lessens the algorithm sensitivity to control parameter values. Although the differential evolution algorithm itself is surprisingly simple to implement there is still much bewilderment among scientists about setting the values of the control parameters. So any step towards parameter insensitiveness of the DE is welcome.

As seen in Fig 8, one still needs a relatively large population in order to stand a fair chance of finding the glo-

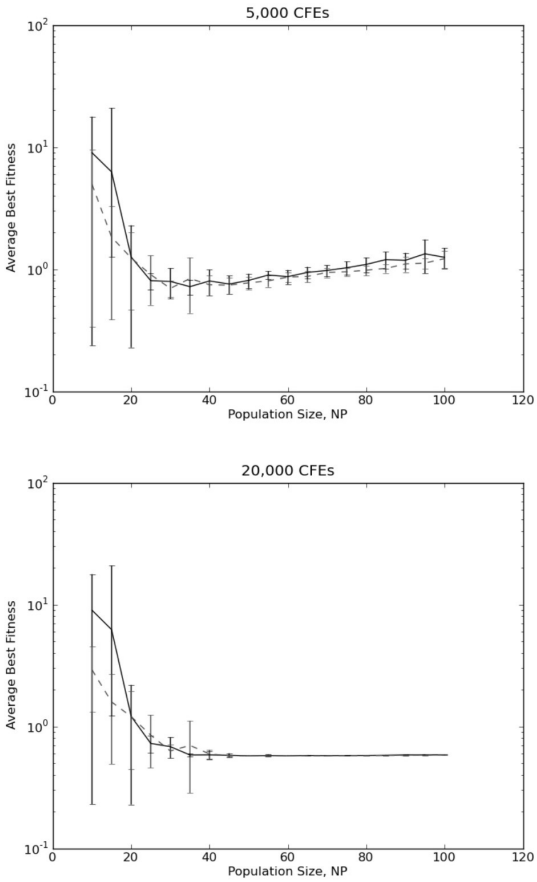

Figure 8: Impact of the population size in optimizing the simple operating amplifier circuit, after 5,000 CFEs (left) and 20,000 CFEs (right). At the smaller population sizes, the original algoritm was soon stuck (no change from 5,000 CFEs to 20.000 CFEs), while Algorithm 3 with perturbation still improved the outcome. The graph shows a situation at the fixed *CR*=0.9 and *F*=0.5.

bal minimum. It seems that innumerable local minima introduced by the intensive numerical noise could only be overcome with relatively large populations. At the same time we observe that our proposed modification quite improves results at small populations, i.e. *NP* < 20. Eventhough the improvement itself does not lead us to the global minimum at that small populatuion sizes, it could be crucial for reducing the sizes of sub-populations in the multi-population model of parallel differential evolution [27 and the references within].

6. Conclusion

In the paper we studied different replacement schemes in the DE algorithm combined with additional random perturbation of vector parameters. By experimenting with a suite of standard test functions we observed that only one replacement scheme provided observably better results than the original algorithm. It was somehow surprising to note that perturbation did not improve behavior of the original replacement scheme while it improved all the others.

Studying the performance of Algorithm 3 combined with random perturbation showed a statistically significant improvement in all higher dimensional test functions. We also saw that the improvement is greater at certain values of the control parameters and population sizes, i.e. at lower values of *F* and *NP*, and at lower as well as higher values of *CR*.

Especially outstanding was the improvement in smaller population sizes. According to the outcomes of the experiments with both the Quartic noisy function and the real-world circuit we beleive it is worthwile to aim some research effort in the direction of DE parallelization using sub-populations of sizes below 20.

One of the advantages of the approach proposed in this paper is the fact that its intervention with the original method does not interfere with any other operation and can therefore be applied independently and combined with many other approaches proposed in literature.

All in all, the beauty of the original DE algorithm is its utmost implementation simplicity. Our research tried not to stray away from this simplicity and we showed that it is possible to improve the algorithm performance by only changing the rule for replacing the population members combined with simple random perturbation. We believe that further work in this direction is worthwhile.

Acknowledgement

This work has been supported by the Ministry of Education, Science, Culture and Sport of Republic of Slovenia within the research program P2-0246 – Algorithms and optimization methods in telecommunications.

References

1. R. Storn and K. Price, "Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces," J. Glob. Optim., vol. 11, pp. 341–359, 1997.
2. K. Price, "An introduction to differential evolution," in New ideas in optimization, D. Corne, M. Dorigo, and F. Glover, Eds. London (UK): McGraw-Hill Ltd., 1999, pp. 79–108.

3. D. Jiang, H. Wang, and Z. Wu, "A variant of differential evolution based on permutation regulation mechanism," in International Symposium on Intelligence Computation and Applications (ISICA): 2010, pp. 76–85.
4. E. Mezura-Montes, J. Velázquez-Reyes, and C. A. Coello Coello, "A comparative study of differential evolution variants for global optimization, " in Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation (GECCO): 2006, vol. 1, pp. 485–492.
5. Y. Wang, Z. Cai, and Q. Zhang, "Differential Evolution With Composite Trial Vector Generation Strategies and Control Parameters," IEEE Trans. Evol. Comput., vol. 15, pp. 55–66, 2011.
6. J. Brest, S. Greiner, B. Bošković and M. Mernik, "Self-adapting control parameters in differential evolution: a comparative study on numerical benchmark problems," IEEE Trans. Evol. Comput., vol. 10, pp. 646–657, 2006.
7. J. Liu and J. Lampinen, "A fuzzy differential evolution algorithm," Soft Comput., vol. 9, pp. 448–462, 2005.
8. D. Zaharie, "Control of population diversity and adaptation in differential evolution algorithms," in Proceedings of 9th International Conference on Soft Computing, R. Matoušek, P. Ošmera, Eds. Brno (Czech Republic): Mendel 2003, pp. 41–46.
9. J. Zhang and A. C. Sanderson, "JADE: Adaptive Differential Evolution With Optional External Archive," IEEE Trans. Evol. Comput., vol. 13, pp. 945–958 , 2009.
10. J. Brest and M. Sepesy Maučec, "Population size reduction for the differential evolution algorithm," Appl. Intell., vol. 29, pp. 228–247, 2008.
11. J. Teo, "Exploring dynamic self-adaptive populations in differential evolution," Soft Comput., vol. 10, pp. 673–686, 2006.
12. C. Zhang, J. Chen, B. Xin, T. Cai, and C. Chen, "Differential evolution with adaptive population size combining lifetime and extinction mechanisms," in Proceedings of 8th Asian Control Conference (ASCC): 2011, pp. 1221–1226.
13. H. Wang, S. Rahnamayan, and Z. Wu, "Adaptive differential evolution with variable population size for solving high-dimensional problems," in Proceedings of the IEEE Congress on Evolutionary Computation (CEC): 2011, pp. 2626–2632.
14. J. Brest and M. S. Maučec, "Self-adaptive differential evolution algorithm using population size reduction and three strategies," Soft Comput., vol. 15, pp. 2157–2174, 2011.
15. H. Y. Fan and J. Lampinen, "A trigonometric mutation operation to differential evolution," J. Glob. Optim., vol. 27, pp. 105–129, 2003.
16. S. Das, A. Abraham, U. K. Chakraborty, and A. Konar, "Differential evolution using a neighborhood-based mutation operator," IEEE Trans. Evol. Comput., vol. 13, pp. 526–553, 2009.
17. D. Zaharie, "Influence of crossover on the behavior of differential evolution algorithms," Appl. Soft Comput., vol. 9, pp. 1126–1138, 2009.
18. S. M. Islam, S. Das, S. Ghosh, S. Roy, and P. N. Suganthan, "An adaptive differential evolution algorithm with novel mutation and crossover strategies for global numerical optimization," IEEE Trans. Syst., Man and Cybern. (SMC), part B, vol. 42, pp. 482–500, 2012.
19. Z. Yang, J. He, and X. Yao, "Making a difference to differential evolution," in Advances in metaheuristics for hard optimization, Z. Michalewicz and P. Siarry, Eds.: Springer, 2007, pp. 415–432.
20. J. Olenšek, Á. Bűrmen, J. Puhan, and T. Tuma, "DESA: a new hybrid global optimization method and its application to analog integrated circuit sizing," J. Glob. Optim., vol. 44, pp. 53–77, 2009.
21. J. Brest, S. Greiner, B. Bošković, M. Mernik, and V. Žumer, "Self-adapting control parameters in differential evolution: a comparative study on numerical benchmark problems," IEEE Trans. Evol. Comput., vol. 10, pp. 646–657, 2006.
22. S. Das, A. Konar, and U. K. Chakraborty, "Two improved differential evolution schemes for faster global search," in Proceedings of GECCO, Washington D.C.: 2005, pp. 991–998.
23. X. Yao, Y. Liu, and G. Lin, "Evolutionary programming made faster," IEEE Trans. Evol. Comput., vol. 3, pp. 82–102, 1999.
24. R. Storn, "On the usage of differential evolution for function optimization," in Biennial Conference of the North American Fuzzy Information Processing Society (NAFIPS), Berkeley: 1996, pp. 519–523.
25. R. Gämperle, S. D. Müller, and P. Koumoutsakos, "A parameter study for differential evolution," in Proc. WSEAS NNA-FSFS-EC, Interlaken, Switzerland: 2002, pp. 293–298.
26. A. Bűrmen, D. Strle, F. Bratkovič, J. Puhan, I. Fajfar, and T. Tuma, "Automated robust design and optimization of integrated circuits by means of penalty functions," AEÜ Int. J. Electron. Commun., vol. 57, pp. 47–56, 2003.
27. W. Zhu, "Massively parallel differential evolution—pattern search optimization with graphics hardware acceleration: an investigation on bound constrained optimization problems," J. Glob. Optim., vol. 50, pp. 417–437, 2011.
28. I. Fajfar, "Selection strategies and random perturbations in differential evolution," in Proc. IEEE Congress on Evolutionary Computation, Brisbane, Australia: 2012.

Arrived: 25. 07. 2012
Accepted: 09. 10. 2012