Experimental Comparison of Methods to Handle Boundary Constraints in Differential Evolution

Jaroslaw Arabas[1], Adam Szczepankiewicz[2], and Tomasz Wroniak[2]

[1] Institute of Electronic Systems,
Warsaw University of Technology, Poland
jarabas@elka.pw.edu.pl
[2] Faculty of Electronics and Information Technology,
Warsaw University of Technology, Poland

Abstract. In this paper we show that the technique of handling boundary constraints has a significant influence on the efficiency of the Differential Evolution method. We study the effects of applying several such techniques taken from the literature. The comparison is based on experiments performed for a standard DE/rand/1/bin strategy using the CEC2005 benchmark. The paper reports the results of experiments and provides their simple statistical analysis. Among several constraint handling methods, a winning approach is to repeat the differential mutation by resampling the population until a feasible mutant is obtained. Coupling the aforementioned method with a simple DE/rand/1/bin strategy allows to achieve results that outperform in many cases results of almost all other methods tested during the CEC2005 competition, including the original DE/rand/1/bin strategy.

1 Introduction

Differential Evolution (DE) [1, 2] has proved an efficient and a very simple algorithm from the Evolutionary Computation (EC) family. In particular, the DE scheme overcomes one of the basic difficulties that are met when tuning an Evolutionary Algorithm (EA), namely, the issue of a proper setting of the mutation range. In DE, mutation is based on differences between chromosomes contained in the population, and, since their distribution is influenced by the shape of the fitness function, the distribution of mutants reflects that shape as well. This effect has been called by Storn a "contour matching property".

The basic DE scheme has been introduced for an unconstrained optimization task, whereas in most of the engineering problems, there may appear additional constraints that any solution must satisfy. In general, handling constraints in EC is a problem itself, and a lot of research has been conducted to elaborate efficient methods of solving it. In this contribution we focus on applying DE in optimization problems with boundary constraints, where each coordinate of the solution must fit into the range between a lower and an upper bound. This form of constraints allows for very easy check for feasibility, and it is also very easy to define constraint handling strategies that are based on repairing. For

these reasons, the choice of a technique to handle boundary constraints does not appear to be a serious problem, and maybe this explains why we were unable to find a systematic study that addresses the issue of boundary constraints handling methods. We give an experimental evidence that this issue cannot be treated too easily, since it may have a surprisingly big influence on the DE effectiveness.

In the research papers devoted to DE we can find several techniques to handle boundary constraints. Price *et al.* [1] suggest that random reinitialization (that is, replacing an infeasible solution by a randomly initialized one) is "the most unbiased approach". This method has been also used e.g. in [3, 4]. Price *et al.* [1] defined also a "bounce back" strategy, where an infeasible solution y, generated by mutating a feasible solution x, is replaced by a new feasible solution located on a line between x and y. This approach has been applied e.g. in [5, 6]. Another approach to repairing infeasible solutions is to reflect them back from the bound [7]. Yet another possibility [8] is to project infeasible solutions on bounds, which consists in changing each parameter that exceeds a boundary value to a new value which equals the boundary.

Some optimization tasks, e.g. digital filter design, are periodic in nature. Then, search can be constrained to parameter values from a certain base interval covering a single period, which are treated feasible. Values outside that range can be shifted by an integer multiple of the interval length to fit it the feasible area [9]. We shall call this strategy a wrapping approach.

The motivation for this paper was given by studying the results of the CEC2005 competition of optimization algorithms [10]. In the CEC2005 proceedings, 18 papers took part in the competition. Among the submitted papers, some authors did not report details of handling constraints, and those who reported used various techniques. In the group of algorithms that performed quite well, one can find a very basic DE method [7]. In this paper we investigate if the results of this method can be improved by changing the method of handling boundary constraints.

2 Differential Evolution algorithm

The outline of the DE algorithm is depicted in Fig. 1. With P^t we denote the population in the generation number t, and P_i^t stands for the i-th chromosome in the population P^t; finally, with $P_{i,j}^t$ and x_j we denote the j-th coordinate of the chromosome $P_{i,j}^t$ and x, respectively. All chromosomes are n-dimensional vectors of real numbers, and μ is the size of the population P^t.

The algorithm minimizes the fitness function $f : R^n \to R$. We assume that for each dimension, a pair of numbers l_i, u_i is defined which are lower and upper bound of the feasible area in the i-th dimension. Thus, the feasible area is defined by two vectors composed of lower bound values l and upper bound values u.

In our study we considered a simple DE/rand/1/bin strategy which implements DE steps in the following way. In the initialization phase, population P^0 is filled with chromosomes that are generated with uniform distribution in the feasible area $[l, u]$. The selection rule returns a chromosome $x = P_i^t$ where the

```
algorithm DE with constraint handling
P^0 ← initialize(l, u)
repeat until stop condition met
    for  i ∈ 1...μ
        x ← selection (P^t)
        v ← differential mutation(x, P^t)
        if  v is feasible then w ← v
                else w ← repair(v, x, P^t)
        y ← crossover(P_i^t, w)
        P_i^{t+1} ← replacement(P_i^t, y)
    end for
    t ← t + 1
end repeat
```

Fig. 1. Outline of a basic DE algorithm

index i is a random variable with uniform distribution in $\{1, ..., \mu\}$. Differential mutation uses two chromosomes P_j^t and P_k^t whose indexes are also random variables with uniform distribution in $\{1, ..., \mu\}$. In the result, a chromosome v is generated according to the formula

$$v = x + F(P_j^t - P_k^t) \tag{1}$$

where F is a scaling factor defined by the user. When a constrained optimization problem is considered, chromosome v may become infeasible, therefore a repairing strategy is applied that generates a new feasible chromosome w instead of v. In this paper we study several versions of the repair procedure, and those versions are discussed in the next subsection. The new chromosome w, which resulted from repairing the chromosome v, undergoes crossover with the chromosome P_i^t. In our study we focus on the binary crossover which is defined as follows

$$y_j = \begin{cases} w_j & \text{with probability } CR \\ P_{i,j}^t & \text{with probability } 1 - CR \end{cases} \tag{2}$$

where CR is a user-defined parameter. In the replacement step, i-th chromosome for the next generation results from the tournament of chromosomes P_i^t and y:

$$P_i^{t+1} = \begin{cases} y & f(y) < f(P_i^t) \\ P_i^t & \text{otherwise} \end{cases} \tag{3}$$

Considered constraint handling methods In the presented study we compare the following methods of handling boundary constraints which we have found in the literature.

- *Conservatism:* If the differential mutation resulted in an infeasible chromosome, it is rejected and $w = x$. This strategy is equivalent to the assumption

that the infeasible chromosome will not be repaired, but it will be rejected in the replacement phase.
- *Reinitialization:* Chromosome w is generated with the uniform distribution in the feasible area.
- *Reflection:* Chromosome w is generated by reflecting coordinate values from the exceeded boundary values

$$w_i = \begin{cases} v_i & l_i \leq v_i \leq u_i \\ 2u_i - v_i & v_i > u_i \\ 2l_i - v_i & v_i < l_i \end{cases} \tag{4}$$

If the resulting chromosome is still infeasible, it is reflected again, and the procedure is repeated until feasibility is obtained.
- *Projection:* All coordinate values that exceed bounds are trimmed to the boundary values

$$w_i = \begin{cases} v_i & l_i \leq v_i \leq u_i \\ u_i & v_i > u_i \\ l_i & v_i < l_i \end{cases} \tag{5}$$

- *Wrapping:* All coordinate values that exceed the admissible range are shifted by an integer multiple of the range such that they become feasible

$$w_i = \begin{cases} v_i & l_i \leq v_i \leq u_i \\ v_i + k_i(u_i - l_i) & \text{otherwise} \end{cases} \tag{6}$$

where k_i is an integer number that guarantees feasibility for the i-th dimension.
- *Resampling:* Selection of a random chromosome from P^t and its differential mutation is repeated until a feasible chromosome is obtained:

$$w = \text{differential mutation}(\text{selection}(P^t), P^t) \tag{7}$$

This means that chromosomes x, P_j^t and P_k^t are selected anew by picking them with the uniform distribution from the population P^t.

3 Experiments and results

Conditions of testing We performed an experimental comparison using the suite of fitness functions that have been defined for the CEC2005 competition. Detailed description of functions and conditions of testing are defined in [11], therefore we provide only a brief information about the benchmark.

The benchmark is a compilation of 25 minimization problems that can be divided into four groups (in brackets we refer to function numbers assumed originally in the benchmark definition): 1) unimodal functions (f_1 up to f_5), 2) basic multimodal functions (f_6 up to f_{12}) which include functions by Ackley,

Griewank, Rastrigin, Rosenbrock, Schwefel, and Weierstrass, 3) expanded functions (f_{13}, f_{14}) which are combinations of Griewank, Rosenbrock and Shaffer functions, 4) composition functions (f_{15} up to f_{25}) which resulted from combining functions from groups 1) – 3). Most of the problems from that benchmark are defined for a shifted, rotated, and scaled coordinate system. With two exceptions (f_7 and f_{25}), all other problems have boundary constraints. Some of the benchmark functions have their global minimum on the boundary. Tests are performed for 10-dimensional and 30-dimensional problems.

For each optimization problem, the algorithm is run 25 times, and each time it returns the best value found in that run. For each benchmark function its global minimum is known, so the difference between the fitness value of the best chromosome and the true global minimum can be treated as the error of a single run. Thus, for 25 independent runs one obtains populations of 25 error values whose statistics are reported. For a better readability and compactness, in our study we decided to reduce the suggested set of reported error statistics to the mean value and standard deviation. They are computed for populations of solutions obtained by each of 25 independent runs after $n \cdot 10000$ evaluations of the fitness function.

All experiments were performed using a plain DE/rand/1/bin strategy assuming the following parameter values: population size $\mu = 10n$, scaling factor $F = 0.8$, binary crossover rate $CR = 0.9$. We applied these settings and performed testing with each of the aforementioned constraint handling techniques.

Results Results obtained by DE/rand/1/bin with various constraint handling methods are reported in Tab. 1, 2 for all bounded CEC2005 problems. To facilitate the interpretation of results, for each test function we use the mean error values to assign ranks to all constraint handling methods under comparison. Then, for each constraint handling method we average its ranks over all test functions and we report these averaged rank values.

In addition to the mean value and standard deviation of the fitness function, we indicate the percentage of chromosomes that have been generated outside the admissible area and required repairing. We provide the mean values of the percentage of repaired chromosomes for each constraint handling method (averaged over all test functions). This indicates how effectively a constraint handling method avoids generation of infeasible chromosomes. We also report for each function how frequently a constraint handling method was used (averaging over all constraint handling methods), which informs about the importance of a proper choice of a constraint handling method for that particular function.

A general conclusion is that the choice of the constraint handling technique may significantly influence the final result. Still, for 10-dimensional problems it is not clear which strategy is the most efficient. For some functions, like f_1, f_2, f_{13}, f_{14}, application of any constraint handling technique yields similar results. Projection and reflection work well when the global minimum is located on bounds or close to them, which is the case of functions f_5,$f_{18} - f_{20}$. Reinitialization appears extremely effective for the Schwefel 1.2 problem (f_{12}). Resampling and conservatism are clearly winning strategies only for the Weier-

Table 1. Mean and standard deviation of results yielded by DE/rand/1/bin after 100000 fitness function evaluations for 10-dimensional problems from CEC2005

		projection	reflection	resampling	conservative	wrapping	reinitialize	% repaired
f_1	mean	7.89E-09	8.14E-09	7.73E-09	8.59E-09	**7.59E-09**	8.37E-09	8.1
	std	1.85E-09	1.26E-09	1.59E-09	9.57E-10	1.50E-09	1.10E-09	
f_2	mean	8.14E-09	8.65E-09	8.24E-09	8.15E-09	**7.73E-09**	8.52E-09	11.6
	std	1.62E-09	1.05E-09	1.61E-09	1.41E-09	1.39E-09	1.24E-09	
f_3	mean	1.61E-08	3.58E-08	**8.58E-09**	1.35E-07	5.33E-08	5.18E-08	15.3
	std	8.23E-09	3.81E-08	1.10E-09	1.21E-07	3.41E-08	4.10E-08	
f_4	mean	**8.18E-09**	8.21E-09	8.75E-09	8.24E-09	8.28E-09	**8.18E-09**	11.4
	std	1.34E-09	1.48E-09	1.12E-09	1.31E-09	1.26E-09	1.48E-09	
f_5	mean	**1.23E-08**	4.91E-05	3.59E-06	1.63E+00	1.33E+00	1.39E+00	67.9
	std	8.72E-09	3.88E-05	1.84E-06	6.88E-01	4.92E-02	6.01E-01	
f_6	mean	3.81E-06	1.59E-01	**1.69E-06**	1.74E-05	4.66E-06	1.59E-01	7.75
	std	6.10E-06	7.81E-01	2.00E-06	1.84E-05	3.76E-06	7.81E-01	
f_8	mean	**2.05E+01**	**2.05E+01**	2.06E+01	2.07E+01	**2.05E+01**	**2.05E+01**	15.0
	std	7.25E-02	7.35E-02	1.55E-01	1.22E-01	9.82E-02	9.41E-02	
f_9	mean	2.75E+01	3.34E+01	2.46E+01	2.45E+01	**2.32E+01**	2.52E+01	13.3
	std	1.94E+01	1.30E+01	1.32E+01	1.66E+01	1.71E+01	1.45E+01	
f_{10}	mean	2-19E+01	3.39E+01	2.42E+01	2.43E+01	2.49E+01	**1.95E+01**	11.7
	std	1.36E+01	1.45E+01	1.61E+01	1.60E+01	1.66E+01	1.55E+01	
f_{11}	mean	6.58E+00	7.35E+00	**4.12E+00**	4.71E+00	8.79E+00	8.11E+00	16.4
	std	2.78E+00	2.30E+00	3.55E+00	3.79E+00	1.38E+00	2.90E+00	
f_{12}	mean	1.04E+02	5.22E+00	6.23E+01	1.73E+02	5.54E+01	**4.00E-01**	13.3
	std	4.05E+02	8.11E+00	3.05E+02	4.61E+02	2.64E+02	1.96E+00	
f_{13}	mean	**2.76E+00**	2.77E+00	2.97E+00	3.00E+00	2.85E+00	2.79E+00	3.6
	std	1.21E+00	1.22E+00	1.11E+00	9.62E-01	1.12E+00	1.22E+00	
f_{14}	mean	3.80E+00	3.69E+00	3.78E+00	**3.61E+00**	3.94E+00	3.84E+00	17.5
	std	6.37E-01	5.20E-01	4.98E-01	6.86E-01	1.39E-01	4.36E-01	
f_{15}	mean	3.84E+02	3.02E+02	2.08E+02	2.59E+02	2.05E+02	**1.88E+02**	11.6
	std	9.23E+01	1.31E+02	1.05E+02	1.09E+02	1.13E+02	9.87E+01	
f_{16}	mean	1.55E+02	1.53E+02	1.47E+02	**1.35E+02**	1.48E+02	1.50E+02	13.0
	std	3.52E+01	3.16E+01	3.54E+01	2.90E+01	3.08E+01	2.95E+01	
f_{17}	mean	1.69E+02	1.78E+02	1.56E+02	1.60E+02	**1.55E+02**	1.61E+02	16.1
	std	2.81E+01	2.60E+01	2.62E+01	2.41E+01	2.35E+01	3.51E+01	
f_{18}	mean	**3.82E+02**	4.00E+02	8.00E+02	7.40E+02	8.00E+02	8.00E+02	12.0
	std	1.71E+02	2.00E+02	2.64E-11	1.62E+02	7.61E-10	7.15E-10	
f_{19}	mean	3.63E+02	**3.40E+02**	7.80E+02	7.40E+02	7.40E+02	8.00E+02	11.6
	std	1.50E+02	1.36E+02	9.80E+01	1.62E+02	1.62E+02	4.80E-10	
f_{20}	mean	**3.52E+02**	3.60E+02	8.00E+02	7.45E+02	7.64E+02	8.00E+02	11.5
	std	1.44E+02	1.62E+02	2.16E-11	1.66E+02	1.38E+02	1.35E-09	
f_{21}	mean	5.00E+02	5.00E+02	**4.36E+02**	4.52E+02	4.92E+02	4.92E+02	7.9
	std	8.28E-12	3.20E-12	9.33E+01	8.54E+01	3.92E+01	3.92E+01	
f_{22}	mean	7.83E+02	7.04E+02	6.80E+02	6.84E+02	**6.27E+02**	7.43E+02	12.0
	std	2.20E+01	1.76E+02	1.93E+02	1.92E+02	2.24E+02	1.31E+02	
f_{23}	mean	**5.59E+02**	**5.59E+02**	5.98E+02	6.34E+02	5.85E+02	5.72E+02	8.1
	std	1.44E-12	1.68E-12	1.05E+02	1.01E+02	5.93E+01	4.39E+01	
f_{24}	mean	2.14E+02	**2.00E+02**	**2.00E+02**	**2.00E+02**	**2.00E+02**	**2.00E+02**	8.5
	std	6.64E+01	3.15E-12	3.00E-12	9.25E-11	2.86E-11	3.94E-11	
avg. rank		2.87	3.35	2.74	3.39	3.00	3.43	
% repaired		15.5	12.4	14.7	15.0	13.6	13.7	

strass function f_{11}. Despite of local differences, average percentage of repaired chromosomes stays on roughly similar level for all methods under comparison.

In contrast to the 10-dimensional case, for 30-dimensional problems a clear picture is obtained. Resampling appears the most effective constraint handling strategy. Projection and reflection seem to be the methods of the second choice. Reinitialization performs surprisingly bad. As for the average percentage of infeasible chromosomes, its clear minimum value is recorded for the reinitialization, and for the other methods under comparison, this value is roughly similar. Thus we state a hypothesis that the good performance of the resampling strategy results from its efficiency in avoiding generation of infeasible chromosomes.

Resampling method is the only one where the repairing cost cannot be guaranteed. To estimate it, for each test function we computed the average number of trial chromosomes that are generated to repair a single infeasible chromosome. For 10-dimensional problems, the average repairing cost ranged from 2.3 to 6.8 (the mean value was 3.3), and for 30-dimensional problems, it ranged from 3.8 to 29.3 (the mean value was 6.7).

Comparison with other CEC2005 competitors Interpretation of the presented results becomes easier after looking at Tab. 3 which contains a comparison of the results between DE/rand/1/bin with constraint handling by resampling and the algorithms that were found superior in the summary of the CEC2005 competition [10]. Those algorithms are: DE variants — plain DE/rand/1/bin[7] (treated as a reference method), DE-535[12] (modified DE/rand/1/bin, scaling factor generated randomly) L-SaDE[13] (DE with adaptive scaling factor and local search incorporated); Memetic EA — BLX-GL50[14], BLX-MA[15], SPC-PNX[16]; Steady state EA — K-PCX[17]; Coevolutionary EA — CoEVO[18]; simple Particle Swarm Optimization DMS-L-PSO[19]; Estimation of Distribution — EDA[20] and CMA-ES using two restart strategies — G-CMA-ES[21], L-CMA-ES[22]. For L-SaDE and DMS-L-PSO, no results have been reported for $n = 30$, and for DE-520, only results for $f_1 - f_{14}$ are available.

Each cell of the table, which corresponds to a function f_i and the competing algorithm number j, indicates if the mean error achieved by DE/rand/1/bin with resampling was smaller (symbol '+') or greater (symbol '–') than the mean error achieved by the algorithm number j for the function f_i. Since the compared methods are stochastic, we applied a generalized t-Student's test of equal means. This generalization, called Welch test, is applicable for samples driven from normal distributions with different standard deviations. Symbols '–' or '+' appear when the difference was statistically significant, i.e., when the probability of accepting the null hypothesis that both mean values were equal was smaller than 0.05. If the null hypothesis could not be rejected, it is indicated with '·'.

Analysis of Tab.3 indicates that, for several test functions, the combination of the DE with the resampling method could have allowed the DE/rand/1/bin to perform significantly better than the DE version with reinitialization [7], which might have allowed to even outperform the winners of the CEC2005 competition for some functions. This effect is clearly visible in 30 dimensions on multimodal composition functions, where [7] reported rather poor results. For unimodal and

Table 2. Mean and standard deviation of results yielded by DE/rand/1/bin after 300000 fitness function evaluations for 30-dimensional problems from CEC2005

		projection	reflection	resampling	conservative	wrapping	reinitialize	% repaired
f_1	mean	6.82E+01	6.30E+01	**1.53E+01**	7.07E+02	3.00E+02	3.63E+02	50.0
	std	2.61E+01	2.06E+01	7.12E+00	1.98E+02	9.75E+01	1.01E+02	
f_2	mean	1.61E+02	2.06E+02	**1.57E+01**	1.23E+03	5.49E+02	6.37E+02	60.7
	std	7.11E+01	5.59E+01	6.65E+00	6.20E+02	1.70E+02	2.29E+02	
f_3	mean	4.96E+05	6.04E+05	**5.97E+03**	3.93E+06	1.48E+06	1.53E+06	76.8
	std	1.93E+05	1.78E+05	2.68E+03	1.75E+06	4.96E+05	4.44E+05	
f_4	mean	5.57E+02	7.09E+02	**7.61E+01**	4.18E+03	1.65E+03	1.97E+03	67.3
	std	1.93E+02	2.33E+02	3.98E+01	2.54E+03	5.30E+02	5.87E+02	
f_5	mean	3.93E+03	4.43E+03	**9.09E+02**	1.17E+04	8.13E+03	9.80E+03	88.1
	std	8.23E+02	5.49E+02	4.99E+02	1.50E+03	9.92E+02	1.37E+03	
f_6	mean	5.14E+04	3.97E+04	**1.16E+04**	3.90E+06	7.66E+05	1.67E+06	51.4
	std	3.26E+04	2.41E+04	9.02E+03	2.38E+06	5.11E+05	1.11E+06	
f_8	mean	2.10E+01	2.10E+01	2.11E+01	2.11E+01	**2.09E+01**	2.10E+01	46.3
	std	5.22E-02	4.26E-02	2.26E-01	2.18E-01	1.08E-01	5.22E-02	
f_9	mean	1.16E+02	1.30E+02	**8.97E+01**	1.84E+02	1.59E+02	1.68E+02	67.3
	std	3.36E+01	4.19E+01	1.14E+01	2.94E+01	2.91E+01	3.35E+01	
f_{10}	mean	1.41E+02	1.48E+02	**9.91E+01**	2.41E+02	1.98E+02	2.17E+02	70.5
	std	2.58E+01	3.52E+01	4.24E+01	1.60E+01	3.02E+01	1.76E+01	
f_{11}	mean	3.98E+01	3.74E+01	**2.20E+01**	2.56E+01	3.87E+01	3.72E+01	54.2
	std	2.73E+00	5.29E+00	5.89E+00	2.50E+00	3.72E+00	5.59E+00	
f_{12}	mean	1.00E+05	4.79E+04	**1.02E+04**	7.65E+04	8.85E+04	6.34E+04	65.7
	std	4.45E+04	2.12E+04	6.64E+03	3.00E+04	2.96E+04	2.58E+04	
f_{13}	mean	1.85E+01	1.82E+01	**1.57E+01**	1.92E+01	1.77E+01	1.80E+01	30.0
	std	1.07E+00	1.31E+00	1.94E+00	1.44E+00	1.41E+00	1.61E+00	
f_{14}	mean	1.35E+01	1.34E+01	**1.31E+01**	**1.31E+01**	1.36E+01	1.36E+01	55.9
	std	2.46E-01	4.19E-01	9.91E-01	6.40E-01	2.37E-01	2.07E-01	
f_{15}	mean	4.48E+02	4.50E+02	**4.30E+02**	5.11E+02	4.87E+02	4.84E+02	60.9
	std	2.59E+01	2.06E+01	3.56E+01	3.22E+01	1.94E+01	4.42E+00	
f_{16}	mean	1.65E+02	1.70E+02	**1.36E+02**	3.01E+02	2.46E+02	2.60E+02	76.3
	std	3.13E+01	3.76E+01	2.83E+01	4.05E+01	1.95E+01	1.99E+01	
f_{17}	mean	2-47E+02	2.56E+02	**1.89E+02**	3.36E+02	2.97E+02	2.99E+02	85.8
	std	1.87E+01	2.59E+01	5.95E+01	5.34E+01	2.10E+01	1.92E+01	
f_{18}	mean	9.22E+02	9.33E+02	**8.61E+02**	9.93E+02	9.62E+02	9.68E+02	90.6
	std	4.52E+00	3.76E+00	5.34E+01	2.80E+01	1.70E+01	2.75E+01	
f_{19}	mean	9.22E+02	9.33E+02	**8.44E+02**	9.95E+02	9.49E+02	9.63E+02	89.9
	std	4.52E+00	5.23E+00	5.29E+01	2.40E+01	2.65E+01	2.90E+01	
f_{20}	mean	9.22E+02	9.32E+02	**8.39E+02**	9.94E+02	9.53E+02	9.66E+02	89.8
	std	4.51E+00	5.71E+00	5.13E+01	2.41E+01	2.19E+01	2.70E+01	
f_{21}	mean	5.25E+02	5.09E+02	**5.04E+02**	6.76E+02	5.79E+02	6.15E+02	56.1
	std	3.28E+01	3.15E+00	1.96E+00	6.92E+01	3.44E+01	5.10E+01	
f_{22}	mean	**8.88E+02**	9.26E+02	9.29E+02	1.02E+03	9.94E+02	9.96E+02	88.0
	std	1.01E+01	1.12E+01	1.80E+01	1.47E+01	1.26E+01	1.50E+01	
f_{23}	mean	5.50E+02	5.40E+02	**5.35E+02**	7.52E+02	5.97E+02	6.73E+02	57.3
	std	1.09E+01	5.35E+00	2.99E+00	1.24E+02	2.57E+01	1.28E+02	
f_{24}	mean	5.62E+02	2.20E+02	**2.05E+02**	5.45E+02	3.40E+02	4.01E+02	61.9
	std	3.61E+02	5.20E+00	1.72E+00	1.16E+02	5.19E+01	7.65E+01	
avg. rank		2.74	2.65	1.09	5.30	4.09	4.48	
% repaired		73.8	66.2	51.7	71.8	68.6	68.5	

basic multimodal functions, DE/1/rand/bin with any constraint handling technique is relatively ineffective. In our opinion this indicates a poor performance in exploitation which can be overcome by intelligent restarting and/or by the hybridization with some local optimization method, like in several other methods that took part in the competition [13–16, 21, 22].

Table 3. Comparison of the DE/rand/1/bin with resampling versus leading optimization methods from CEC2005

$n = 10$	1	2	3	4	5	6	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
DE/rand/1/bin	−	−	+	−	−	−	−	−	+	·	+	−	−	·	+	·	−	−	−	+	·	·	·
DE.535	−	·	+	+	+	+	−	−	−	−	·	−	−	·	−	−	−	·	−	+	+	+	−
L.SaDE	−	−	+	·	+	−	−	−	−	·	·	−	−	−	−	−	·	·	−	·	·	·	·
BLX.GL50	·	·	+	·	−	−	−	−	−	−	·	−	−	+	−	−	−	−	−	+	·	·	·
BLX.MA	+	+	+	+	+	+	−	−	−	·	·	−	−	+	−	−	+	·	·	+	·	+	−
SPC.PNX	+	+	+	+	−	+	+	−	−	−	·	−	−	·	−	−	−	−	−	+	·	·	·
K.PCX	+	+	+	·	+	·	−	−	−	+	·	−	−	+	−	−	·	·	·	+	·	+	−
CoEVO	+	·	·	·	+	+	−	·	·	+	+	−	·	+	+	+	+	·	+	+	+	+	−
DMS.L.PSO	−	−	−	+	−	−	−	−	−	·	·	−	−	−	−	−	·	·	+	+	·	+	·
EDA	−	−	−	−	+	−	−	+	+	+	·	·	·	+	+	+	−	−	−	+	+	·	·
G.CMA.ES	−	−	−	−	−	−	−	−	−	−	·	−	−	·	−	−	−	−	−	+	·	·	·
L.CMA.ES	−	−	−	·	−	−	−	+	+	·	·	−	·	·	−	+	−	−	−	·	·	+	−

$n = 30$	1	2	3	4	5	6	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
DE/rand/1/bin	−	−	+	−	−	−	−	−	−	+	·	−	·	+	+	+	+	+	+	+	+	+	−
DE.535	−	+	+	+	+	−	·	−	−	·	−	−	·	−	+	−	·	·	+	+	−	+	−
BLX.GL50	−	−	−	−	−	−	−	−	−	·	·	−	−	−	−	−	+	+	+	−	−	+	−
BLX.MA	−	−	+	−	+	−	−	−	·	+	−	−	−	−	+	+	·	+	+	−	−	+	−
SPC.PNX	−	−	+	−	+	−	−	−	−	−	·	−	·	−	−	−	+	+	+	−	−	·	−
K.PCX	−	−	−	+	+	−	−	−	−	+	−	−	+	+	−	·	−	·	·	+	+	+	−
CoEVO	−	−	−	+	+	−	−	+	+	+	+	−	·	·	+	+	+	+	+	+	+	+	−
EDA	+	+	+	+	+	+	−	+	+	+	+	+	+	+	+	+	+	+	+	·	−	+	−
G.CMA.ES	−	−	−	·	−	−	−	−	−	·	·	−	·	−	−	+	+	+	+	−	−	·	−
L.CMA.ES	−	−	−	+	−	−	−	+	+	−	·	−	+	−	−	+	+	+	+	−	−	·	−

4 Summary

We showed that the results obtained by the DE/rand/1/bin algorithm were significantly influenced by the method to handle boundary constraints. A strategy that repeats the differential mutation until a feasible solution is found appeared to be a winning one, and in particular, it appeared superior to reinitialization which is nowadays quite commonly used in DE. This observation might indicate yet another possibility to improve efficiency of the DE by choosing an appropriate boundary constraint handling technique.

References

1. Price, K., et al.: Differential Evolution: A Practical Approach to Global Optimization. Springer (2005)
2. Neri, F., Tirronen, V.: Recent advances in Differential Evolution: a survey and experimental analysis. Artificial Intelligence Rev. **33**(1-2) (2010) 61–106

3. Qin, A.K., et al.: Differential Evolution algorithm with strategy adaptation for global numerical optimization. IEEE Trans. Evolutionary Computation **13**(2) (2009) 398–417
4. Liu, J., Lampinen, J.: A Fuzzy Adaptive Differential Evolution algorithm. Soft Computing **9**(6) (2005) 448–462
5. Zhang, J., Sanderson, A.C.: JADE: adaptive Differential Evolution with optional external archive. IEEE Trans. Evolutionary Computation **13**(5) (2009) 945–958
6. Doumpos, M., Marinakis, Y., Marinaki, M., Zopounidis, C.: An evolutionary approach to construction of outranking models for multicriteria classification: The case of the ELECTRE TRI method. Eur. J. of Operational Research **199**(2) (2009) 496 – 505
7. Rönkkönen, J., et al.: Real-parameter optimization with differential evolution. In: CEC2005, IEEE (2005)
8. Brest, J., et al.: Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. IEEE Trans. Evolutionary Computation **10**(6) (2006) 646–657
9. Karabogal, N., Cetinkayal, B.: Design of digital FIR filters using Differential Evolution algorithm. Circuits, Systems, Signal Processing **25**(5) (2006) 649 – 660
10. Hansen, N.: Compilation of results on the 2005 CEC benchmark function set. http://www.ntu.edu.sg/home/epnsugan/index_files/CEC-05/compareresults.pdf
11. Suganthan, P.N., et al.: Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization. Technical report, Nanyang Tech. Univ. (2005)
12. Bui, L.T., et al.: Comparing two versions of differential evolution in real parameter optimization. In: CEC2005, IEEE (2005)
13. Qin, A., Suganthan, P.: Self-adaptive differential evolution algorithm for numerical optimization. In: CEC2005, IEEE (2005)
14. Martines, C.G., Lozano, M.: Hybrid real-coded genetic algorithms with female and male differentiation. In: CEC2005, IEEE (2005)
15. Molina, D., et al.: Adaptive local search parameters for real-coded memetic algorithms. In: CEC2005, IEEE (2005)
16. Ballester, P., et al.: Real-parameter optimization performance study on the CEC-2005 benchmark with SPC-PNX. In: CEC2005, IEEE (2005)
17. Sinha, A., et al.: A population-based, steady-state procedure for real-parameter optimization. In: CEC2005, IEEE (2005)
18. Posik, P.: Real parameter optimization using mutation step co-evolution. In: CEC2005, IEEE (2005)
19. Liang, J., Suganthan, P.: Dynamic multi-swarm particle swarm optimizer with local search. In: CEC2005, IEEE (2005)
20. Yuan, B., Gallagher, M.: Experimental results for the special session on real-parameter optimization at CEC 2005: A simple, continuous EDA. In: CEC2005, IEEE (2005)
21. Auger, A., et al.: A restart CMA evolution strategy with increasing population size. In: CEC2005, IEEE (2005)
22. Auger, A., et al.: Performance evaluation of an advanced local search evolutionary algorithm. In: CEC2005, IEEE (2005)