Evaluation of Bound Constraints Handling Methods in Differential Evolution using the CEC2017 Benchmark

Vinicius Kreischer[1], Thiago Tavares Magalhães[1], Helio J. C. Barbosa[1,2], and Eduardo Krempser[3]

[1] Laboratório Nacional de Computação Científica, Petrópolis, RJ, 25651-075, Brazil
{valmeida,thiagotm,hcbm}@lncc.br
[2] Universidade Federal de Juiz de Fora, Juiz de Fora, MG, 36036-330, Brazil
[3] Fundação Oswaldo Cruz – Plataforma Institucional Biodiversidade e Saúde Silvestre, Rio de Janeiro, RJ, 21040-361, Brazil
eduardo.krempser@fiocruz.br

Abstract. This paper presents an experimental analysis of some of the most popular methods for handling boundary constraints in the Differential Evolution algorithm. We also propose an additional method were an infeasible mutant vector is scaled back to the allowable bounds through the selection of an adequate scale factor. The selected methods are applied to the CEC2017 benchmark suite for single objective real-parameter numerical optimization. We present a statistical analysis using the Wilcoxon test and Performance Profiles. The experimental results show the superiority of the method known as Resampling and that, for some scenarios, our proposed method might be a good option.

1 Introduction

Differential Evolution (DE) is a stochastic and population-based metaheuristic proposed in [1, 2], originally to solve unconstrained optimization problems with continuous variables. It is a well-know algorithm characterized, among other features, by the simplicity, robustness and the ability to handle problems that are not easily solved by classical optimization methods. The results obtained by DE in practical real-world problems have drawn the attention of a large number of researchers resulting in a growing number of publications.

Although DE was initially proposed for unconstrained problems, several subsequent publications have been addressing adaptations in the original algorithm in order to tackle problems with constraints, such as follows:

$$\min f(\boldsymbol{x}) \tag{1}$$

subject to:

$$u_i \leq x_i \leq l_i \tag{2}$$

$$g_l(\boldsymbol{x}) \geq \alpha_l, l = 1, ..., p \tag{3}$$

$$h_k(\boldsymbol{x}) = \beta_k, k = 1, ..., q \tag{4}$$

where $\boldsymbol{x}$ denotes the solution vector for the objective function $f(\boldsymbol{x})$.

The majority of works in the literature is related to the study of techniques used for handling constraints defined by (3) and (4). Works related exclusively to the type of constraint defined by (2), also called boundary constraints, seem to be overlooked despite the impact that the selected method for handling such type of constraint might have over the efficiency of the Differential Evolution algorithm. As argued in [3], the handling of such constraints is simple, being the repair or substitution of infeasible solutions some of the most popular methods.

In [4], a systematic study of different boundary constraints handling methods found in the literature is carried out. The authors of the study show that the efficiency of the Differential Evolution algorithm on the CEC2005 benchmark suite [5] is significantly influenced by the chosen method. It is also shown that, for the considered benchmark, the best approach is to repeat the differential mutation by resampling individuals from the population until a feasible mutation is obtained.

As in the aforementioned paper, the present work has the objective of evaluating different boundary constraints handling methods. Nevertheless, we add three approaches to the set of analyzed methods, two found in the literature and one new proposal. Then, they are evaluate using the CEC 2017 special session on single objective real-parameter numerical optimization [6]. Besides, in order to establish a comparative analysis, Performance Profiles [7, 8] are used.

The rest of the paper is organized as follows: Sect. 2 presents an overview of the basic DE algorithm; in Sect. 3 the boundary repair methods found in the literature and an additional proposal are described; details about the computational experiments and the results obtained are given in Sect. 4; some conclusions drawn from the experiments are given in Sect. 5.

2 Differential Evolution Algorithm

The iterative process that characterizes the DE algorithm starts with the generation of a random population of NP individuals with D design variables, being NP one of the control parameters of the algorithm to be defined by the user. The generation of the j-th design variable, for the individual with population index i, is performed according to:

$$x_{i,j,G=0} = l_j + rand(0,1)(u_j - l_j) \tag{5}$$

where the function $rand(0,1)$ is responsible for returning a real pseudo-random number uniformly distributed in the interval $[0,1]$, while u_j and l_j stand for the upper and lower bounds, respectively, of the j-th design variable.

After the step above, an objective value is computed for each individual present in the population. For unconstrained minimization problems, individuals with lower values of the objective function correspond to better solutions for the problem, while in maximization problems, larger values denote better solutions. For the purposes of the following explanations, only the former case is considered.

In the next step of the algorithm, for each member of the current population, called a target vector ($\boldsymbol{x_i}$), a new candidate solution, called a mutant vector ($\boldsymbol{m_i}$), is produced by means of the mutation operator. This new vector is obtained by adding to a candidate solution in the current population one or more scaled differences between vectors in the population.

Several variants of the original mutation scheme have been proposed since the inception of DE, the difference between them given by the way that the vectors employed are selected and the number of scaled differences used. This work considers the mutation scheme named *target-to-best/1*, proposed in [1], which has been largely employed in several works concerning the DE algorithm. In this scheme, a mutation vector is generated according to:

$$\boldsymbol{m_i} = \boldsymbol{x_i} + F_1(\boldsymbol{x_{best}} - \boldsymbol{x_i}) + F_2(\boldsymbol{x_{r1}} - \boldsymbol{x_{r2}}) \tag{6}$$

where the control parameters F_1 and F_2 are scale factors, with $F_1, F_2 \in (0, 1+)$ for the difference vectors with the possibility that $F_1 = F_2$. The subscript *best* denotes the population member with the best fitness (lowest objective function value), while r_1 and r_2 indicate randomly selected population members.

It is easy to see that the mutation operator may sometimes produce values that fall outside the prescribed range $[\boldsymbol{l_j}, \boldsymbol{u_j}]$ for the j-th design variable. The next section present some of the methods employed to address such cases.

The next step of the algorithm is given by the crossover operator, which generates a trial vector ($\boldsymbol{t_i}$) through the exchange of components of the mutant and target vectors. The scheme employed here and in the majority of works related to the DE is known as bionomial (or uniform) crossover and is given by:

$$t_{i,j} = \begin{cases} m_{i,j}, \text{ if } rand_j(0,1) \leq CR \text{ or } j = J \\ x_{i,j}, \text{ otherwise} \end{cases} \tag{7}$$

where $CR \in [0, 1]$ is the user-defined crossover probability, and the value J is an integer randomly generated in $[1, D]$ in order to ensure that the trial vector is never an exact copy of its corresponding target vector.

Finally, after a fitness value is assigned to the trial vector, a replacement operator takes place. The elitist replacement depicted in Eq. 8 is considered, where the trial vector will replace its corresponding target vector if it has a smaller or equal objective function value.

$$\boldsymbol{x}_{i,G+1} = \begin{cases} \boldsymbol{t}_{i,G}, \text{ if } f(\boldsymbol{t}_{i,G+1}) \leq f(\boldsymbol{x}_{i,G}) \\ \boldsymbol{x}_{i,G}, \text{ otherwise.} \end{cases} \tag{8}$$

A pseudo-code for the Differential Evolution algorithm can be seen in Alg. 1. The mutation, crossover, and replacement operators are performed for each individual in the population, for several iterations, also called generations. The algorithm is terminated when a maximum number (GEN) of generations is met.

Algorithm 1 Basic DE algorithm

```
1: procedure DE(NP, CR, F, GEN)
2:     g=0
3:     CreateRandomInitialPopulation(NP)
4:     Evaluate f(x_{i,g})                                   ▷ ∀i, i = 1, ..., NP
5:     for g = 1 : GEN do
6:         for i = 1 : NP do
7:             m_{i,g} = Mutation (x_{i,g}, x_{best,g}, x_{r1,g}, x_{r2,g})
8:             t_{i,g} = Crossover (m_{i,g}, x_{i,g})
9:             Evaluate f(t_{i,g})
10:            x_{i,g+1} = Replacement (t_{i,g}, x_{i,g})
11:        end for
12:    end for
13: end procedure
```

a

3 Methods for Handling Bound Constraints

The mutation operator of the DE algorithm may generate individuals where one or more components fall outside the corresponding bounds. In such cases, a method to repair or substitute an infeasible individual is usually employed. This section presents the approaches compared in our experiments and the novel method for handling bound constraints suggested as part of this paper. The two techniques added to those in [4] are named *Midpoint Base* and *Midpoint Target*.

3.1 Methods found in the literature

– **Projection [9]:** infeasible components are assigned the value of the violated bound.

$$m_{i,j,g} = \begin{cases} u_j, & \text{if } m_{i,j,g} > u_j \\ l_j, & \text{if } m_{i,j,g} < l_j \\ m_{i,j,g}, & \text{otherwise} \end{cases} \tag{9}$$

– **Reinitialization [3]:** a new value is randomly generated inside the allowable range for infeasible components.

$$m_{i,j,g} = \begin{cases} l_j + rand_j(0,1)(u_j - l_j), & \text{if } u_j < m_{i,j,g} < l_j \\ m_{i,j,g}, & \text{otherwise} \end{cases} \tag{10}$$

– **Rand Base [3]:** out-of-bounds components are assigned a new value that lies between the respective base vector's component and the violated bound.

$$m_{i,j,g} = \begin{cases} base_{j,g} + rand_j(0,1)(u_j - base_{j,g}), & \text{if } m_{i,j,g} > u_j \\ base_{j,g} + rand_j(0,1)(l_j - base_{j,g}), & \text{if } m_{i,j,g} < l_j \\ m_{i,j,g}, & \text{otherwise} \end{cases} \tag{11}$$

– **Midpoint Base [3]:** infeasible components are replaced by the midway point between the respective base vector's component and the violated bound.

$$m_{i,j,g} = \begin{cases} \frac{1}{2}(base_{j,g} + u_j), & \text{if } m_{i,j,g} > u_j \\ \frac{1}{2}(base_{j,g} + l_j), & \text{if } m_{i,j,g} < l_j \\ m_{i,j,g}, & \text{otherwise} \end{cases} \tag{12}$$

– **Midpoint Target [10]:** similar to the above method, except this time infeasible components are replaced by the midway point between the respective target vector's component and the bound being violated.

$$m_{i,j,g} = \begin{cases} \frac{1}{2}(target_{j,g} + u_j), & \text{if } m_{i,j,g} > u_j \\ \frac{1}{2}(target_{j,g} + l_j), & \text{if } m_{i,j,g} < l_j \\ m_{i,j,g}, & \text{otherwise} \end{cases} \tag{13}$$

– **Reflection [11]:** out-of-bounds components are replaced by a value that denotes the reflection of the violation relatively to the violated bound.

$$m_{i,j,g} = \begin{cases} 2u_j - m_{i,j,g}, & \text{if } m_{i,j,g} > u_j \\ 2l_j - m_{i,j,g}, & \text{if } m_{i,j,g} < l_j \\ m_{i,j,g}, & \text{otherwise} \end{cases} \tag{14}$$

– **Conservatism [4]:** if one or more trial vector's components fall outside the allowable ranges, the trial vector becomes a copy of the base vector.

$$m_{i,g} = \begin{cases} base_{i,g}, & \text{if } u_j < m_{i,j,g} < l_j \\ m_{i,g}, & \text{otherwise} \end{cases} \tag{15}$$

– **Resampling [4]:** different individuals are randomly sampled from the population to take part in the mutation operator, which is performed again until a feasible individual is yielded or a maximum number of tries is reached. Here, we selected 100 as the maximum number of tries. If this number is not sufficient to produce a feasible individual, the target vector is kept in the population.

3.2 An Additional Proposal

Here we propose a simple bound handling method, called Scaled Mutant, for the case when the origin is an interior point of the feasible set: an infeasible mutant vector is scaled back to the allowable bounds via a multiplier α selected according to:

$$\alpha = \min\{1, \alpha_1, ..., \alpha_D\} \tag{16}$$

where α_j is given by:

$$\alpha_j = \begin{cases} \frac{u_j}{m_{i,j}}, & \text{if } m_{i,j} > 0 \\ \frac{l_j}{m_{i,j}}, & \text{if } m_{i,j} < 0 \end{cases} \tag{17}$$

When $m_{i,j} = 0$, the corresponding α is not computed.

4 Experiments and analysis tools

In order to study the performance of the methods described in Section 3, we employed each of them to solve a set of problems via Differential Evolution algorithm. Then, we analyse the results with the support of a statistical and a visualization tools. This methodology and its items are explained in this Section.

4.1 Numerical experiments

We used the benchmark designed for the CEC 2017 special session on single objective real-parameter numerical optimization [6]. While the set provided in [5] is composed of 25 problems, this test-suit offers 30 scalable minimization problems, which can be categorized into: unimodal functions ($f1, f2, f3$), simple multimodal functions ($f4, f5, \ldots, f10$), hybrid functions ($f11, f12, \ldots, f20$), and composition functions ($f21, f22, \ldots, f30$). For all cases, we choose 30-dimension configuration and the search range is given by $[-100, 100]^{30}$.

The implemented Differential Evolution algorithm employed the *target-to-best/1/bin* mutation scheme and the binomial crossover operator described in Section 2. The crossover rate (CR) and the scaling factor (F) were set to 0.9 and 0.8, respectively. These values were chosen in order to increase the utilization of the bound constraint handling methods while not disrupting the search process. The population size was set to $NP = 100$ and the maximum number of generations to 3,000, what results in a total of 300,000 evaluations per execution.

Each problem was executed 25 times, for each bound constraint handling method, with different random seeds to extract statistical information. To make it easier the visualization and interpretation of these numerical experiments results, we employed a statistical test known as Wilcoxon Signed-rank Test [12] and an analytical tool known as Performance Profiles [7].

4.2 Wilcoxon Test and Performance Profiles

The Wilcoxon test is a non-parametric statistical hypothesis test used to compare two numerical samples, giving the significance of the difference between them. It indicates, according to a chosen confidence level, here selected as 95%, if it might be asserted or not that one sample is statistically different from another.

In turn, to use performance profiles, first it is necessary to select a measure ($t_{p,a}$) representative of the relative performance of the algorithm $a_i \in A = \{a_1, ..., a_n\}$ applied to the test-problem $p_i \in P = \{p_1, ..., p_m\}$. Given the definition of $t_{p,a}$, the performance ratio $r_{p,a}$ can be described as

$$r_{p,a} = \frac{t_{p,a}}{\min\{t_{p,a} : a \in A\}} \tag{18}$$

The performance profiles present in a compact graphical form the performance of the solvers in A on a large set of problems P. Denoting the cardinality of a

set A by $|A|$ and given the definition of $\rho(\tau)$ as

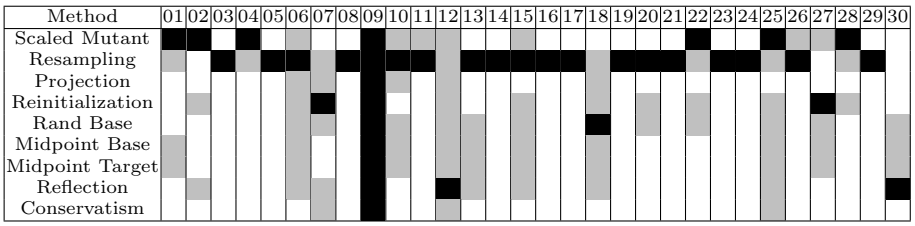
$$\rho_{p,a} = \frac{1}{n_p}|\{p \in P : r_{p,a} \leq \tau\}| \tag{19}$$

then $\rho(\tau)$ is the fraction of the problems in P such that the solver $a_j \in A$ is able to find solutions in a factor $r_{p,a} \leq \tau$ compared to the best performance observed. It is suggested in [8] that the area under the ρ_a curve ($AUC_a = \int \rho_a(t)dt$) is an overall performance/measure for solver a in the problem set P: the larger the AUC the higher the solver efficiency. This property is employed to assess the results obtained in the numerical experiments.

4.3 Results and Discussion

Initially, Tables 1 and 3 list results derived from the Wilcoxon tests, discriminating the test functions. For each of the 30 test functions from the benchmark, the cell related to the method that achieved the minimum median regarding the optima found along the 25 independent runs is highlighted in black. Then, to the same problem, if the Wilcoxon test does not allow us to consider the results obtained by another method statistically worse than the best, but the method also presents the lower median, the corresponding cell is marked in gray. Otherwise, the corresponding cell remains white. In turn, these observations are grouped in Tables 2 and 4, which present the sum of Wilcoxon tests without discriminating the problems. In Tables 3 and 4 we recorded results with 100,000 function evaluations, while Tables 1 and 2 present the results related to the total budget of 300,000 function evaluations.

Table 1. Wilcox results for all algorithms and 300,000 function evaluations



With respect to the Wilcoxon test, it is possible to note the superiority of the Resampling method, in general. Among the other techniques, the Scaled Mutant method, suggested here, is highlighted achieving the best results. With 100,000 function evaluations, this pattern is maintained. However, in this case the performance difference between the Resampling method and all other is increased. On the other hand, Projection and Conservatism present the worst results, also emphasized in the 100,000 function evaluations configuration. A relevant aspect of this observation is that the Projection method is specially widely employed

Table 2. Sum of Wilcox results for all algorithms and 300,000 function evaluations

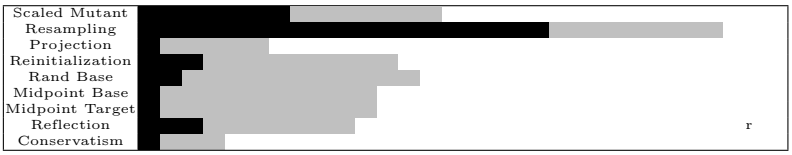
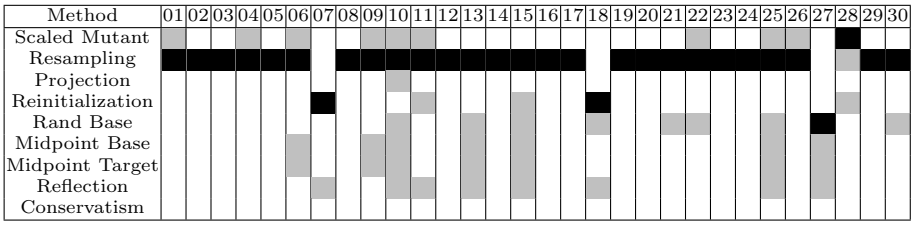


Table 3. Wilcox results for all algorithms and 100,000 function evaluations

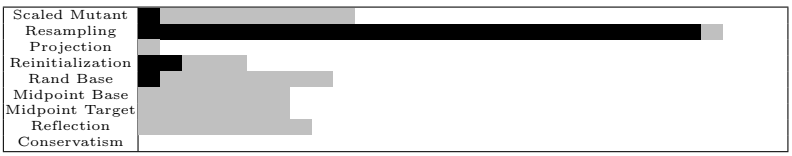


in the literature. According to these results, the influence of this approach in optimization algorithms and the existence of alternative strategies deserves a greater attention by the researchers. The other methods beyond the two best and the two worst present more close results, when compared the amount of promising results through Tables 2 and 4. Some large standard deviations that we recorded associated with each set of 25 independent runs give rise to the number of times that the statistical test did not ensure a real difference between the samples (gray cells in Wilcoxon tables). Despite this, the statistical results show that the choice of an appropriate boundary constraint handling method leads to relevant differences in the optimization process.

Regarding the Performance Profiles, Figs. 1 and 2 show the results for 300,000 and 100,000 function evaluations, respectively. A performance profile including only hybrid and composite functions is presented in Fig. 3, while another including only unimodal and multimodal simple functions is presented in Fig. 4. These classifications are according to the published benchmark definitions.

In both Figs. 1 and 2 the curve related to the Resampling method already presents the higher values for the first values of τ, and is the first to reach the maximum possible value in these Performance Profiles. There is a difference in chart scales due to the difference in budget, since the treatment of the problems with 100,000 function evaluations is a harder task. Moreover, in both cases the

Table 4. Sum of Wilcox results for all algorithms and 100,000 function evaluations



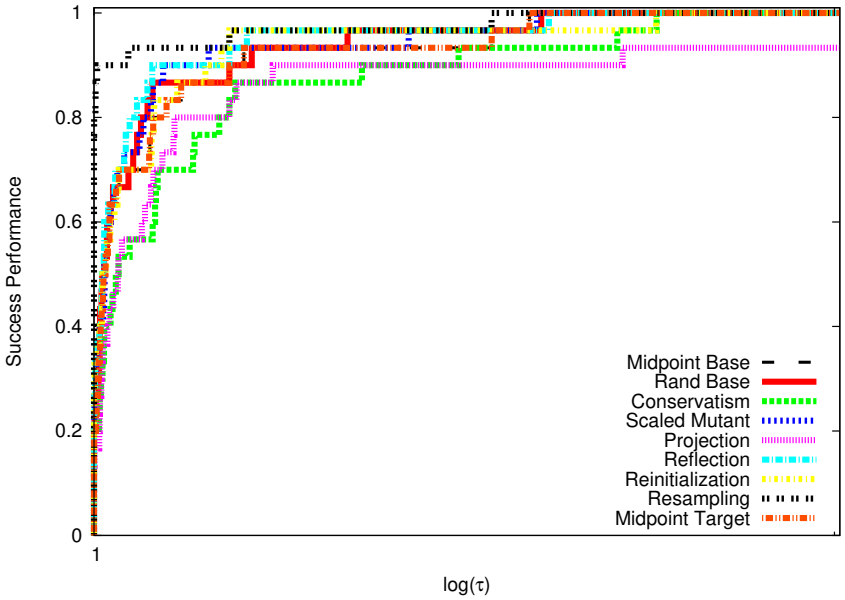


Fig. 1. Performance profile for all problems and 300,000 function evaluations

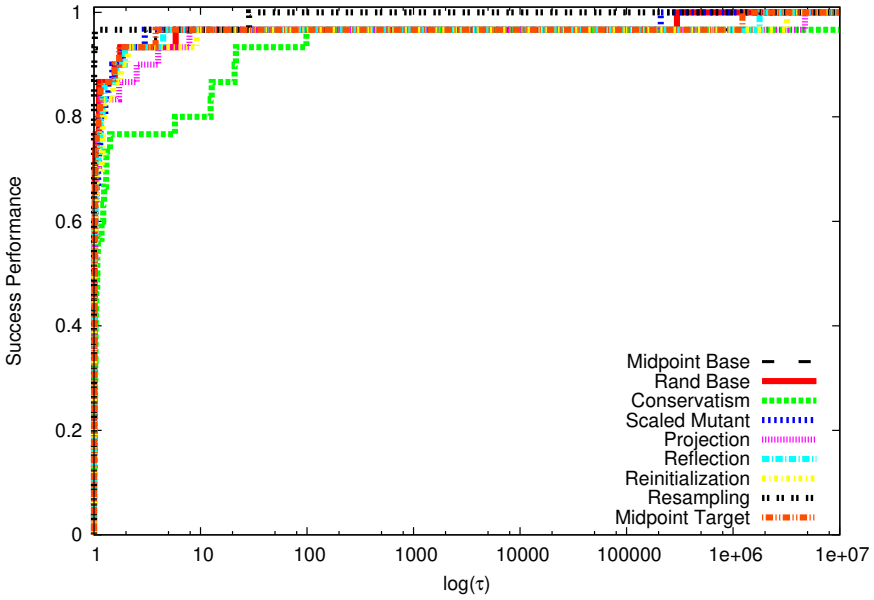


Fig. 2. Performance profile for all problems and 100,000 function evaluations

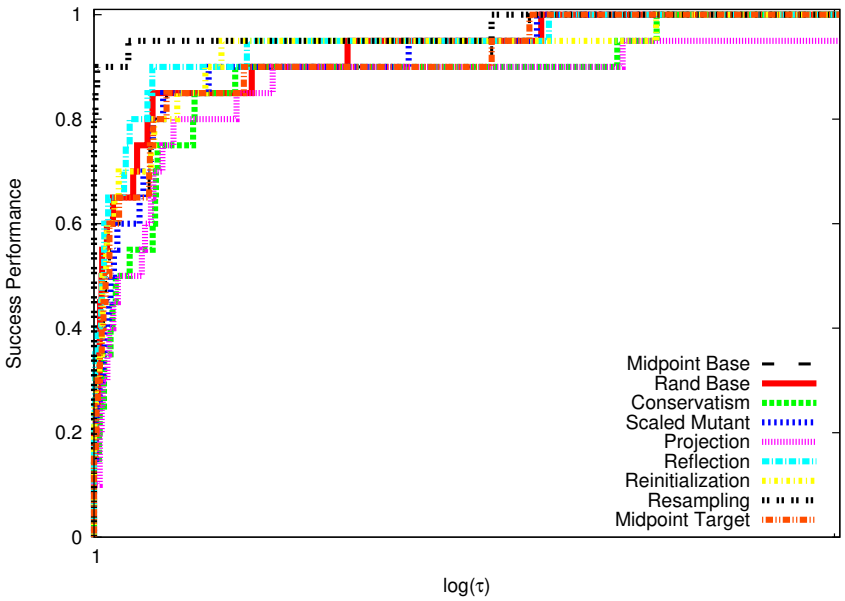


Fig. 3. Performance profile for hybrid and composition problems and 300,000 function evaluations

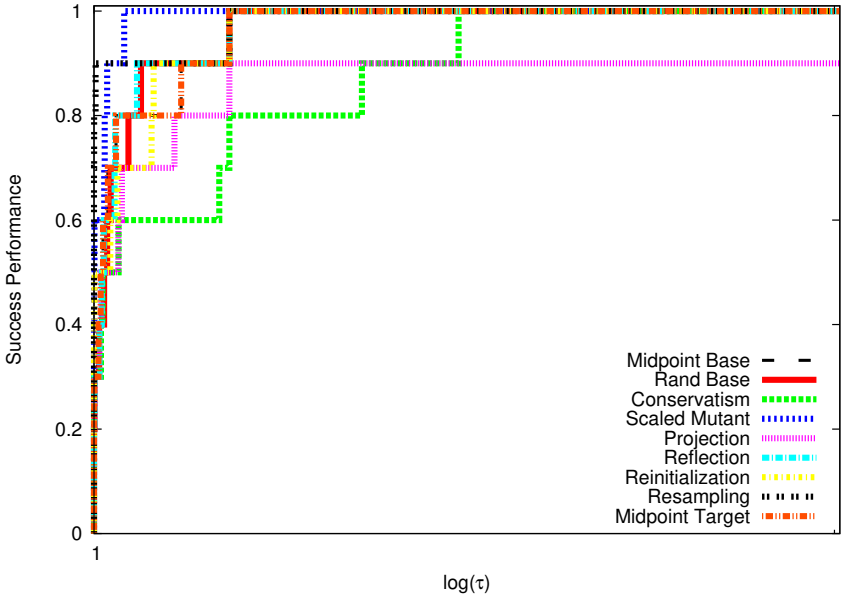


Fig. 4. Performance profile for simple unimodal and multimodal problems and 300,000 function evaluations

curves related to the Conservatism and Projection approaches are highlighted as lower values of success performance. All other curves are relatively close, in a simple visual observation.

The distinction provided in Figs. 3 and 4 allows relating different performances to different types of problems. Regarding the Scaled Mutant approach, for hybrid and composition functions a prominent performance is not observed. However, when comparing only simple unimodal and multimodal functions it is easy to see the superiority of this method. Also, for both performance profiles the worst performances correspond to the Conservatism and Projection methods.

The Table 5 lists the normalized areas under the curves, for each one of the performance profiles. For 3 performance profiles the Resampling method achieves the greater area under the curve, while the Scaled Mutant presented the greater area in 1 performance profile. In addition, the second greater area under the curve is attributed to the methods Reflection (Figs. 1 and 3), Scaled Mutant (Fig. 2) and Resampling (Fig. 4). The smaller areas correspond to Projection (Figs. 1, 3 and 4) and Conservatism (2), as expected after the analysis of the medians. In addition, the difference in success performance for the Scaled Mutant method when comparing the two budget configurations can indicate that this approach holds an additional capability to solve problems in lower budget conditions.

Table 5. Normalized areas under all the performance profile curves

	Fig. 1	Fig. 2	Fig. 3	Fig. 4
Scaled Mutant	0.9736	0.9993	0.9564	**1.0000**
Resampling	**1.0000**	**1.0000**	**1.0000**	0.9924
Projection	0.8958	0.9842	0.9015	0.8779
Reinitialization	0.9699	0.9892	0.9624	0.9774
Rand Base	0.9717	0.9990	0.9635	0.9804
Midpoint Base	0.9632	0.9959	0.9516	0.9787
Midpoint Target	0.9632	0.9959	0.9516	0.9787
Reflection	0.9813	0.9940	0.9770	0.9823
Conservatism	0.9189	0.9666	0.9216	0.9066

5 Conclusions

Firstly, the results obtained in this work show that the boundary constraint handling strategy has a relevant influence on the results obtained by the Differential Evolution algorithm. Due to this fact, it is possible to indicate the usefulness of the efforts in this theme, which is not one of the most studied in the literature.

The Resampling method achieved the best results, with a significant difference with respect to other methods. In general, the Scaled Mutant method presented the second best results also with a very relevant difference to other methods. However, it is important to note that the Scaled Mutant vector does not produce a large number of mutant vectors whenever an infeasible individual is found, in contrast with the Resampling Method. In addition, it is possible to cite the Reflection as the third best technique.

Considering only the simple unimodal and multimodal functions, Scaled Mutant presents the best results, suggesting that this proposal can be an interesting alternative in similar cases. Thus, a promising approach may be to try both the Resampling and the Scaled Method, whenever one needs to solve a new problem.

An important finding is related to the strategies that achieved the worst results. Specially regarding Projection, one of the most employed strategies in the literature, which presented one of the two worst performances in general. An interesting future analysis concerns the use of benchmarks in which the optima have active constraints. For these cases, it is possible that methods not positively highlighted in this work may achieve good results, according to their pattern of treatment of infeasible solutions.

Acknowledgements

We thank the support from CNPq (grant 310778/2013-1) and CAPES.

References

1. Storn, R., Price, K.: Differential Evolution - A simple and efficient adaptive scheme for global optimization over continuous spaces. TR-95-012 (1995)
2. Storn, R., Price, K.: Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. J. of Global Optimization 11(4), 341–359 (2010)
3. Price, K., et al.: Differential Evolution: A Practical Approach to Global Optimization (Natural Computing Series). Springer-Verlag New York, Inc. (2005)
4. Arabas, J., et al.: Experimental Comparison of Methods to Handle Boundary Constraints in Differential Evolution. Parallel Problem Solving from Nature - PPSN XI, Part II, 411–420 (2010)
5. Suganthan, P. N., et al.: Problem Definitions and Evaluation Criteria for the CEC 2005 Special Session on Real-Parameter Optimization. Nanyang Tech. Univ. (2005)
6. Awad, N.H., et al.: Problem Definitions and Evaluation Criteria for the CEC 2017 Special Session and Competition on Single Objective Real-Parameter Numerical Optimization. Technical Report, Nanyang Tech. Univ., Singapore and Jordan Univ. of Science and Tech., Jordan and Zhengzhou Univ., Zhengzhou China (2016)
7. Dolan, E. D., Moré, J. J.: Benchmarking optimization software with performance profiles. Mathematical Programming 91(2), 201–213 (2002)
8. Barbosa, H. J. C., et al.: Using performance profiles to analyze the results of the 2006 CEC constrained optimization competition. In: IEEE Congress on Evolutionary Computation (2010)
9. Brest, J., et al.: Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. IEEE Trans. Evolutionary Computation 10(6), 646–657 (2006)
10. Price, K.: An Introduction to Differential Evolution. In: Corne, D., Dorigo, M., Glover (Eds.), New Ideas in Optimization. McGraw Hill, 109–125 (1999)
11. Ronkkonen, J., et al.: Real-parameter optimization with differential evolution. In: IEEE Congress on Evolutionary Computation, Vol. 1 , 506–513 (2005)
12. Wilcoxon, F.: Individual Comparisons by Ranking Methods. Biometrics Bulletin 1(6), 80–83 (1945)