Underperformance of Current State of the Art Differential Evolution Algorithms as a result of implementation differences

Carlos A. Ramirez
Graduate School of Arts and Sciences
The University of Tokyo

Alex S. Fukunaga
Graduate School of Arts and Sciences
The University of Tokyo

***Abstract*—Differential evolution (DE) is a powerful stochastic search technique for solving numerical optimization problems. Success of DE in solving a specific problem depends on appropriately chosen parameter settings.**

Current state-of-the-art DE algorithms implement adaptive and self-adaptive methods to control parameter settings and mutation strategies automatically in DE. Unfortunately, minor implementation differences in these algorithms drastically affect their behavior. In this paper, we evaluated the performance of slightly different algorithm implementations on a set of well-known benchmark functions. Computational results show that DE and state-of-the-art DE algorithms with certain minor differences in their implementations under-perform on the majority of test problems.

1. Introduction

Differential Evolution (DE) is an Evolutionary Algorithm (EA) that was proposed by Price and Storn [1], [2] for solving numerical optimization problems. It is simple but robust and effective in solving a broad range of optimization problems. Its performance greatly depends on the control parameter settings and the mutation strategies implemented.

In DE, different mutation strategies and control parameter settings may result in important differences of performance. Additionally, optimal settings for these parameters are problem-dependent. Since tunning the control parameters is a significant problem in practice, novel adaptive and self-adaptive mechanisms for automatic parameter settings selection have been developed in the past decade [3], [4].

The above mentioned novel solutions implement different strategies not only to adjust the control parameters but also to tweak settings in the mutation mechanisms and crossover operations during the search process. Because these implementations tend to become more and more complex in newer DE algorithms, authors may implement versions of the algorithm components that are slightly different from the original specification. In some cases, these unnoticed deviations may introduce serious performance issues.

In this paper we evaluate the performance of different algorithm implementations of the standard DE and the state of the art SHADE [7] algorithms. The primary comparison is based on the CEC2014 benchmarks. Additionally, we designed a test function that introduces plateaus to the problem search space, allowing us to test cases where multiple vector parents could share the same best fitness value.

First, in section 2 we review differential evolution briefly, including state of the art algorithms. In section 3 we study different implementation strategies that may be used in different components of DE and SHADE algorithms. Section 4 presents the empirical evaluation of the different DE and SHADE implementations. Finally the work is concluded in section 5.

2. A Brief Review of Differential Evolution

2.1. Differential Evolution (DE)

DE starts with a random generated vector population called parameter vectors or gnomes. These vectors are used to explore the problem landscape. A process of trial vector generation and selection is repeated until some termination criterion is encountered. Subsequent generations in DE are usually denoted by $G = 0, 1, ..., G_{MAX}$. Additionally, the ith individual in population $P(G)$ is usually denoted by ${X_i}^G (i = 1, 2, ..., NP)$ where NP is the population size.

2.1.1. Mutation. In each generation G, a mutant vector $v_{i,G}$ is generated from an existing population member $x_{i,G}$ by applying some mutation strategy. The most common mutation strategies in DE are:

rand/1:

$$v_{i,G} = x_{r_1,G} + F \cdot (x_{r_2,G} - x_{r_3,G}) \tag{1}$$

rand/2:

$$v_{i,G} = x_{r_1,G} + F \cdot (x_{r_2,G} - x_{r_3,G}) + F \cdot (x_{r_4,G} - x_{r_5,G}) \tag{2}$$

best/1:

$$v_{i,G} = x_{best,G} + F \cdot (x_{r_1,G} - x_{r_2,G}) \tag{3}$$

2.1.2. Crossover. After generating the mutant vector $v_{i,G}$, it is crossed with the parent $x_{i,G}$ in order to generate trial vector $u_{i,G}$. Binomial Crossover, the most commonly used crossover operator in DE, is implemented as follows:

$$u_{j,i,G} = \begin{cases} v_{j,i,G} & \text{if rand}[0,1) \leq CR \text{ or } j = j_{rand} \\ x_{j,i,G} & \text{otherwise} \end{cases} \tag{4}$$

2.1.3. Crossover. After all of the trial vectors $u_{i,G}, 0 \leq i \leq N$ have been generated, a selection process determines the survivors for the next generation. The selection operator in standard DE compares each individual $x_{i,G}$ against its corresponding trial vector $u_{i,G}$, keeping the better vector in the population.

$$x_{i,G+1} = \begin{cases} u_{i,G} & \text{if } f(u_{i,G}) \leq f(x_{i,G}) \\ x_{i,G} & \text{otherwise} \end{cases} \tag{5}$$

2.2. State of the art DE algorithms

2.2.1. SHADE. Success-History based Adaptive DE (SHADE), is an enhancement to JADE [5], [6] which uses a history based parameter adaptation scheme [7], [8]. SHADE uses a historical memory M_{CR}, M_F which stores sets of CR, F values that have been found to perform well in the past. SHADE maintains a diverse set of parameters to guide control parameter adaptation as search progresses.

In addition, SHADE implements a randomized approach to setting the control parameter value p in the current-to-$p$best/1 mutation strategy. This parameter is used to adjust the greediness of the mutation strategy. While this parameter is static and manually adjusted in JADE, in SHADE it is generated for each individual x_i, according to the equation by generation:

$$p_i = \text{rand}[p_{min}, 0.2] \tag{6}$$

Experimentally, SHADE was shown to outperform previous DE variants, including JADE, on a large set of benchmark problems [7].

3. Different DE and SHADE implementations

In this section, we study different implementation strategies that may be used in two different components of DE and SHADE algorithms. One is when implementing the generation alternation code (i.e., selection operations or survival mechanisms), and the other is when writing the trial vector generation strategy (i.e., mutation and crossover operations). Below we describe each of these in detail.

3.1. Selection Operations

The selection mechanism for both standard DE and state-of-the-art DE algorithms is straightforward:

$$x_{i,G+1} = \begin{cases} u_{i,G} & \text{if } f(u_{i,G}) \leq f(x_{i,G}) \\ x_{i,G} & \text{otherwise} \end{cases} \tag{7}$$

This general strategy dictates that an individual is to be compared to the trial vector and the one with better fitness value should be kept. When the actual implementation of this mechanism takes place (i.e., when writing the code) some authors implement it as:

$$x_{i,G+1} = \begin{cases} u_{i,G} & \text{if } f(u_{i,G}) < f(x_{i,G}) \\ x_{i,G} & \text{otherwise} \end{cases} \tag{8}$$

After implementation, these two strategies only differ from each other by one single symbol. Because this difference is so small, it may go unnoticed or simply ignored by code authors. One reason authors may choose to ignore the difference is because the implementation still keeps the vector with the better fitness value, which is what the general strategy dictates. However, in some cases replacing individuals with trial vectors that have exactly the same fitness value as them and no better may still improve the general performance of the algorithm. One such case is when the test function used to obtain the fitness value has plateaus in its landscape.

Although rare, test functions in real-world optimization problems may contain plateaus in their search landscape. One example of this is when rounding numeric values. In these cases, it is likely that multiple trial vectors will share the same fitness value, including the best fitness value for the current generation. When it is the case that multiple vectors contain the highest fitness in the generation, excluding them from the gene pool during the selection process may result in underperformance issues during algorithm execution.

3.2. Mutation Operations

In the standard DE, for each vector x_i in generation G, a mutant vector v_i is defined by:

$$v_{i,G} = x_{r_1,G} + F \cdot (x_{r_2,G} - x_{r_3,G}) \tag{9}$$

Another mutation strategy is $best/1$, often used for faster convergence. It includes genes from the current trial vector with the highest fitness value:

$$v_{i,G} = x_{best,G} + F \cdot (x_{r_1,G} - x_{r_2,G}) \tag{10}$$

An interesting case to consider is when the best fitness value in generation G is shared among multiple trial vectors. This could happen under different circumstances, especially in test functions that include plateaus in their search space. When this is the case, choosing a random parent from all available choices is a better strategy than considering a

single option in the mutation strategy. Algorithms that do not consider this particular case fail to feed their genes from multiple parents and this results in restriction of the search operations in the problem space, with lower performance as the ultimate consequence.

4. Experimental Study

4.1. Impact of $<$ and $\leq$ in Selection Operations

In order to verify the performance impact of implementing the selection operations of the algorithms with "$<$" vs. "$\leq$", we employ the step function as it generates plateaus in the search space and whose mathematical definition is:

$$f(x) = \sum_{i=1}^{D} \lfloor z_i + 0.5 \rfloor^2 \tag{11}$$

In the experiments, "$<$" is compared with the original "$\leq$" for both standard DE and SHADE. The exact same parameter settings are used in both algorithms, the only actual difference is changing "<=" to "$<$" in the programming code. The total number of runs was ten for each algorithm.

The control parameters for both of the DE implementations are the following:

- Problem size D = 50 and population size N = 50.
- The crossover rate CR = 0.90 and the weighting factor WF = 0.50.
- The maximum number of evaluations before giving up was set to 60000.

The control parameters for both of the SHADE implementations are the following:

- Problem size D = 50, population size N = 50 and memory size M = 50.
- The maximum number of evaluations before giving up was set to 60000.

4.1.1. Results for the DE algorithm. Figure 1 shows the convergence performance of "<=" and "$<$" for the DE algorithm. As can be seen, both versions have the same convergence speed. However, only four out of the ten runs reach the optimal value in the "$<$" implementation, while all ten runs reach the optimal value in "<=". In other words, implementing the selection operations with "$<$" causes the algorithm to underperform about 60% of the time.

4.1.2. Results for the SHADE algorithm. Figure 2 shows the convergence performance of "<=" and "$<$" for the SHADE algorithm. The results are similar to those of DE. For SHADE, in the "$<$" implementation only three out of the ten runs reach the optimal value, while in the "<=" version eight out of ten succeed.

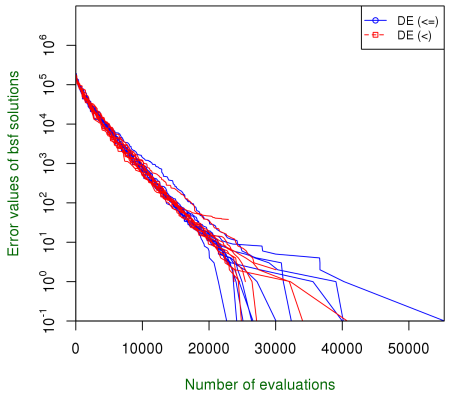

Figure 1. (<=) vs. ($<$) in DE.

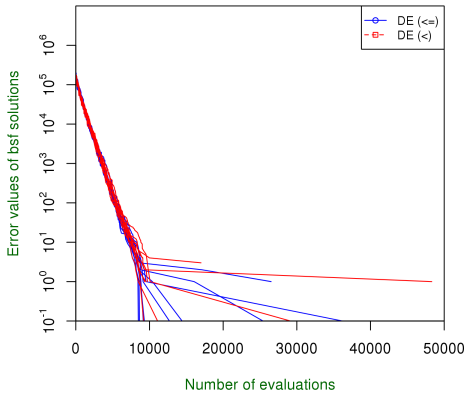

Figure 2. (<=) vs. ($<$) in SHADE.

4.2. Impact of failing to consider cases when multiple trial vectors share the best fitness value in Mutation Operations

In the following experiments, we use the step function from Eq. (11) in order to evaluate different algorithm implementations with heavy occurrence of plateaus in the search space. Moreover, we employ the original CEC2014 standard benchmark function set in order to test common and well known benchmarking functions and not only functions that guarantee the presence of multiple trial vectors containing the same fitness value.

In the first experiment, we compare the performance of an implementation of DE that uses the best/1 mutation strategy from Eq. (10) with another implementation that uses the best/rand mutation strategy. We do the equivalent comparisons with three different implementations of the SHADE algorithm. The first implementation is the standard

implementation using CurrentToPbest/random strategy. The second version is a slightly modified algorithm that accesses in a non-random way the external archive that contains the fittest vectors of previous generations. The third implementation uses CurrentToPbest without randomization.

For this experiment, our performance measurement is given by the average number of fitness evaluations in successful runs divided by the number of successes. Lower means better. In addition, the step function from Eq. (11) is used as the test function for the search space. The total number of runs was fifty for each algorithm.

The control parameters for both of the DE implementations are the following:

- Problem size D = 10, 20, 30, 40, 50.
- Population size N = 100.
- The crossover rate CR = 0.90 and the weighting factor WF = 0.70.
- The maximum number of evaluations before giving up was set to 2,000,000.

The control parameters for all three of the SHADE implementations are the following:

- Problem size D = 10, 20, 30, 40, 50, 60, 70, 80, 90, 100.
- Population size N = 100 and memory size M = 100.
- The maximum number of evaluations before giving up was set to 2,000,000.

4.2.1. Results for the DE algorithm. Figure 3 shows the average number of evaluations per success rate for both of the DE implementations, lower is better. As the graph shows, the performance of the best/1 algorithm decreases as the size of the problem increases. By the time the problem size reaches 50, the performance of the best/random algorithm is two orders of magnitude better than its best/1 counterpart.

4.2.2. Results for the SHADE algorithm. Figure 4 shows the average number of evaluations per success rate for all three of the SHADE implementations. No significant performance differences take place before problem size of $D = 30$.

For problem sizes of $D = 40$ and higher, the performance difference between the three implementations seems to grow steady. The CurrentToPbest without randomization performs the worst, at some points reaching a a difference of two orders of magnitude from the standard SHADE CurrentToPbest/random version.

4.2.3. Performance Evaluation on the CEC2014 Benchmarks. The second experiment uses the CEC2014 benchmark set to evaluate the performance of both, DE and SHADE algorithms for problem sizes of D = 10 and D = 30.

For the DE algorithm, we evaluated the performance of (<=) vs. (<) and the performance of best/1 vs best/random.

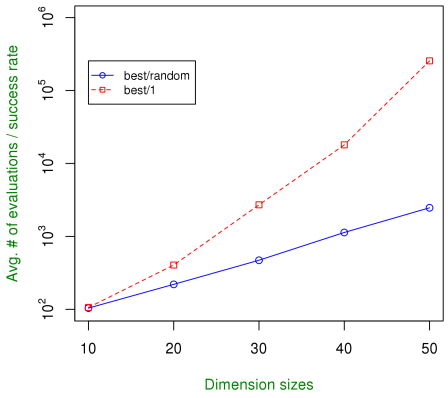

Figure 3. Best/1 vs Best/random in DE.

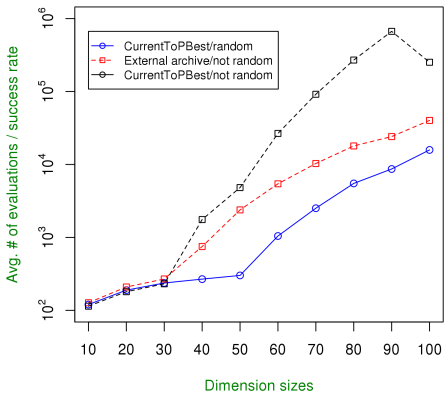

Figure 4. CurrentToPbest/random vs. CurrentToPbest/1 in SHADE.

For the SHADE algorithm we evaluated the performance of (<=) vs. (<) and also CurrentToPbest/random vs. CurrentToPbest/1. Statistical significance testing was done using the Wilcoxon Rank-Sum test with significant threshold of $p < 0.05$.

4.2.4. Results for the CEC2014 Benchmarks. No significant differences were found for the (<=) vs. (<) implementations in both standard DE and SHADE.

In the CurrentToPbest/random vs. CurrentToPbest/1 implementations in SHADE significant differences were found for both problem sizes of D = 10 and D = 30.

Table 1 shows the results for the CurrentToPbest/random vs. CurrentToPbest/1 implementations for problem size of D = 10. The table shows the mean and standard deviation of the error (difference) between the best fitness values found in each run and optimal value. The $+, -, \approx$ indicate whether the CurrentToPbest/1 implementation per-

TABLE 1. COMPARISON OF CURRENTTOPBEST/RAND WITH CURRENTTOPBEST/1 IN SHADE ON THE CEC2014 BENCHMARK FUNCTIONS (10 DIMENSIONS).

F	CurrentToPbest/Rand Mean (Std)	CurrentToPbest/1 Mean (Std)
F1	0.00e+00 (0.00e+00)	0.00e+00 (0.00e+00)≈
F2	0.00e+00 (0.00e+00)	0.00e+00 (0.00e+00)≈
F3	0.00e+00 (0.00e+00)	0.00e+00 (0.00e+00)≈
F4	2.93e+01 (1.28e+01)	2.41e+01 (1.59e+01)≈
F5	1.65e+01 (5.65e+00)	1.70e+01 (5.96e+00)≈
F6	2.67e-04 (9.33e-04)	1.99e-01 (5.11e-01)≈
F7	4.13e-03 (4.04e-03)	1.66e-02 (1.15e-02)−
F8	0.00e+00 (0.00e+00)	0.00e+00 (0.00e+00)≈
F9	2.68e+00 (8.52e-01)	2.91e+00 (1.02e+00)≈
F10	4.90e-03 (1.70e-02)	4.04e-02 (4.64e-02)−
F11	5.64e+01 (4.99e+01)	9.69e+01 (8.21e+01)−
F12	1.86e-01 (3.94e-02)	1.74e-01 (4.03e-02)≈
F13	8.01e-02 (1.42e-02)	8.42e-02 (1.51e-02)≈
F14	1.19e-01 (4.41e-02)	9.90e-02 (3.91e-02)+
F15	5.02e-01 (7.31e-02)	4.96e-01 (1.05e-01)≈
F16	1.51e+00 (2.99e-01)	1.53e+00 (3.11e-01)≈
F17	2.04e+00 (2.10e+00)	5.12e+01 (6.21e+01)≈
F18	1.13e-01 (1.10e-01)	2.83e-01 (3.66e-01)≈
F19	2.22e-01 (2.00e-01)	4.21e-01 (4.89e-01)≈
F20	2.61e-01 (1.05e-01)	2.37e-01 (9.57e-02)≈
F21	3.62e-01 (2.19e-01)	7.97e+00 (2.96e+01)≈
F22	1.63e-01 (4.53e-02)	3.51e+00 (7.08e+00)≈
F23	3.29e+02 (0.00e+00)	3.29e+02 (0.00e+00)≈
F24	1.08e+02 (1.53e+00)	1.09e+02 (1.48e+00)−
F25	1.21e+02 (1.54e+01)	1.40e+02 (3.86e+01)≈
F26	1.00e+02 (1.12e-02)	1.00e+02 (1.80e-02)≈
F27	6.55e+01 (1.32e+02)	1.79e+02 (1.92e+02)−
F28	3.83e+02 (3.86e+01)	4.11e+02 (7.21e+01)−
F29	2.22e+02 (5.43e-01)	4.03e+04 (2.86e+05)−
F30	4.69e+02 (1.94e+01)	4.95e+02 (4.39e+01)−
	+	1
	−	8
	≈	21

TABLE 2. COMPARISON OF CURRENTTOPBEST/RAND WITH CURRENTTOPBEST/1 IN SHADE ON THE CEC2014 BENCHMARK FUNCTIONS (30 DIMENSIONS).

F	CurrentToPbest/Rand Mean (Std)	CurrentToPbest/1 Mean (Std)
F1	8.64e+02 (1.74e+03)	3.40e+03 (3.74e+03)−
F2	0.00e+00 (0.00e+00)	0.00e+00 (0.00e+00)≈
F3	0.00e+00 (0.00e+00)	0.00e+00 (0.00e+00)≈
F4	0.00e+00 (0.00e+00)	2.52e+00 (1.26e+01)≈
F5	2.02e+01 (2.54e-02)	2.02e+01 (2.75e-02)+
F6	1.49e+00 (2.33e+00)	1.60e+00 (1.46e+00)−
F7	0.00e+00 (0.00e+00)	3.62e-03 (6.37e-03)−
F8	0.00e+00 (0.00e+00)	0.00e+00 (0.00e+00)≈
F9	1.95e+01 (3.34e+00)	2.61e+01 (5.84e+00)−
F10	4.49e-03 (8.65e-03)	2.43e+00 (1.66e+01)−
F11	1.55e+03 (1.79e+02)	1.56e+03 (2.50e+02)≈
F12	2.07e-01 (2.98e-02)	1.94e-01 (3.07e-02)≈
F13	1.96e-01 (2.79e-02)	2.52e-01 (4.50e-02)−
F14	2.23e-01 (3.35e-02)	2.36e-01 (3.94e-02)≈
F15	2.85e+00 (3.36e-01)	3.06e+00 (4.54e-01)−
F16	9.39e+00 (2.92e-01)	9.48e+00 (3.98e-01)≈
F17	1.11e+03 (3.70e+02)	1.50e+03 (4.65e+02)−
F18	4.80e+01 (2.58e+01)	1.50e+02 (4.83e+01)−
F19	4.53e+00 (7.62e-01)	6.59e+00 (8.45e+00)−
F20	1.00e+01 (5.86e+00)	4.37e+01 (2.73e+01)−
F21	2.15e+02 (1.24e+02)	4.42e+02 (1.65e+02)−
F22	1.16e+02 (5.80e+01)	1.66e+02 (7.83e+01)−
F23	3.15e+02 (0.00e+00)	3.15e+02 (0.00e+00)≈
F24	2.24e+02 (9.71e-01)	2.29e+02 (6.10e+00)−
F25	2.04e+02 (9.70e-01)	2.04e+02 (1.12e+00)≈
F26	1.00e+02 (3.48e-02)	1.02e+02 (1.40e+01)−
F27	3.19e+02 (3.87e+01)	3.86e+02 (4.38e+01)−
F28	8.02e+02 (3.71e+01)	8.26e+02 (8.68e+01)−
F29	7.22e+02 (8.61e+00)	2.29e+05 (1.63e+06)−
F30	1.30e+03 (4.98e+02)	1.75e+03 (8.27e+02)−
	+	1
	−	19
	≈	10

formed significantly better (+), significantly worse (−), or not significantly different better or worse (≈) compared to CurrentToPbest/random. Seven of the test functions in the CurrentToPbest/1 implementation, namely F_7, F_{10}, F_{11}, F_{24}, F_{27}, F_{28}, F_{29} and F_{30} performed significantly worse than their counterparts in CurrentToPbest/random.

Similarly, Table 2 shows the results for the CurrentToPbest/random vs. CurrentToPbest/1 implementations for problem size of D = 30. In this case nineteen of the test functions in the CurrentToPbest/1 implementation performed significantly worse and only one function performed significantly better.

5. Conclusion

In this paper, we presented empirical performance evaluations of slightly different implementations of the standard DE and state-of-the-art DE algorithms.

The experiments showed that for test functions containing plateau landscapes, both DE and SHADE implementations that only consider individuals with strictly greater health value than their parent require a considerably large number of evaluations before reaching the global minimum value. Depending on the actual number of evaluations that the algorithm is allowed to execute before giving up, a global minimum value may never be found at all.

For functions that have a plateau landscape, both DE and SHADE perform better if they consider for their gene pool individuals with the same or better health value as the current parent. In addition, our results show that these mutation strategies still under-perform in well known benchmark functions with no plateau landscapes, such as the CEC2014 set.

References

[1] R. Stron and K. Price, "Differential Evolution - A Simple and efficient adaptive scheme for global optimization over continuous spaces," *International Computer Science Institute,* Berkeley, CA, Tech. Rep. TR-95-012, March 1995.

[2] ——, "Differential Evolution - A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces," *J. Global Optimiz.,* vol. 11, no. 4, pp. 341-359, Dec. 1997.

[3] S. Das and P. N. Suganthan, "Differential Evolution: A Survey of the State-of-the-Art," *IEEE Tran. Evol. Comput.,* vol. 15, no. 1, pp. 4-31, 2011.

[4] A. K. Qin, V. L. Huang, and P.N. Suganthan, "Self-adaptive differential evolution algorithm for numerical optimization," in *IEEE Evol. Comput. CEC 2005,* vol. 2. pp. 2251-2258, Sept. 2005.

[5] J. Zhang and A. C. Sanderson, "JADE: Self-adaptive differential evolution with fast and reliable convergence performance," in *IEEE Evol. Comput. CEC 2007,* pp. 2251-2258, Sept. 2007.

[6] ——, "JADE: Adaptive Differential Evolution With Optimal External Archive," *IEEE Tran. Evol. Comput.,* vol. 13, no. 5, pp. 945-958, 2009.

[7] R. Tanabe and A. Fukunaga, "Success-History Based Parameter Adaptation for Differential Evolution," in *IEEE Evol. Comput. CEC 2013,* pp. 71-78, 2013.

[8] ——, "Improving the Search Performance of SHADE using Linear Population Size Reduction," in *IEEE Evol. Comput. CEC 2014,* pp. 1658-1665, Jul. 2014.