mathematics

MDPI

Article

Adaptation of the Scaling Factor Based on the Success Rate in Differential Evolution

Vladimir Stanovov * and Eugene Semenkin

Institute of Informatics and Telecommunication, Reshetnev Siberian State University of Science and Technology, 660037 Krasnoyarsk, Russia; eugenesemenkin@yandex.ru
* Correspondence: vladimirstanovov@yandex.ru

Abstract: Differential evolution is a popular heuristic black-box numerical optimization algorithm which is often used due to its simplicity and efficiency. Parameter adaptation is one of the main directions of study regarding the differential evolution algorithm. The main reason for this is that differential evolution is highly sensitive to the scaling factor and crossover rate parameters. In this study, a novel adaptation technique is proposed which uses the success rate to replace the popular success history-based adaptation for scaling factor tuning. In particular, the scaling factor is sampled with a Cauchy distribution, whose location parameter is set as an nth order root of the current success rate, i.e., the ratio of improved solutions to the current population size. The proposed technique is universal and can be applied to any differential evolution variant. Here it is tested with several state-of-the-art variants of differential evolution, and on two benchmark sets, CEC 2017 and CEC 2022. The performed experiments, which include modifications of algorithms developed by other authors, show that in many cases using the success rate to determine the scaling factor can be beneficial, especially with relatively small computational resource.

Keywords: differential evolution; parameter adaptation; numerical optimization

MSC: 68W50; 68T20; 65K10

Citation: Stanovov, V.; Semenkin, E. Adaptation of the Scaling Factor Based on the Success Rate in Differential Evolution. *Mathematics* **2024**, *12*, 516. https://doi.org/10.3390/math12040516

Academic Editors: Jonathan Blackledge and Liliya Demidova

Received: 7 December 2023
Revised: 16 January 2024
Accepted: 29 January 2024
Published: 7 February 2024

1. Introduction

Over the last several years, the differential evolution (DE) [1] algorithm has become one of the most commonly used numerical optimization techniques within the evolutionary computation (EC) community. DE is used in unconstrained, constrained, multi-objective, dynamic, multimodal and other cases [2], when the problem being solved has numerical parameters. Hence, developing more advanced and efficient DE variants is an important research direction for many evolutionary algorithms.

The main problem of DE stems from its main advantage: only three main parameters, scaling factor F, crossover rate Cr and population size N, control most of the optimization process, which makes DE highly sensitive to settings. This problem has been significantly mitigated by parameter adaptation techniques proposed in jDE [3], JADE [4], SHADE [5] and some other schemes [6]. However, there still seems to be room for further improvement. The existing adaptation techniques rely on subsequent immediate improvements, which may constitute a greedy approach. In particular, the popular success history-based adaptation tunes the scaling factor F and crossover rate Cr values based on the most successful values in the last generations, but this does not guarantee that these values will result in good coverage of the search space and good convergence in the long term. This problem was considered in [7], where the effects of bias in parameter adaptation were studied. Hence, some other sources of information could be used to determine parameter values, for example, the number of improved solutions at every generation.

In this study, the scaling factor F adaptation technique is proposed, which is based on the success rate (SR) value, i.e., the ratio of the number of successful new solutions

divided by the total number of individuals. This value is not used explicitly in any of the DE variants to the best of our knowledge, although it is part of the averaging in the JADE and SHADE (Success History Adaptive Differential Evolution) algorithms. The idea of using the success rate came from analyzing the results of the genetic programming (GP) algorithm applied to design novel parameter adaptation techniques [8]. In particular, it was observed that GP heavily relies on the success rate in many of its solutions. Here, a refined variant of this idea is presented, more specifically, a cth order root of the success rate is utilized. To evaluate the efficiency of the new approach for F sampling, it is tested on several algorithms, including L-SHADE-RSP [9], NL-SHADE-RSP [10], NL-SHADE-LBC [11] and L-NTADE [12]. The experiments are performed on two benchmark sets, namely the Congress on Evolutionary Computation (CEC) competitions from 2017 [13] and 2022 [14].

The main features of this study can be outlined as follows:

1. The success rate adaptation of the scaling factor improves the performance of most DE variants and requires the same settings independent of the algorithm;
2. The proposed adaptation scheme shows small dependence on the computational resource or problem dimension;
3. Compared to success history adaptation, success rate adaptation performs better with relatively small computational resource.

The Section 2 contains a short overview of parameter adaptation in differential evolution, the Section 3 describes the proposed approach, the Section 4 contains the experimental setup and results; after that, in Section 5, a discussion of the results is provided, and Section 6 concludes the paper.

2. Related Work

2.1. Differential Evolution

The main focus of our study is on differential evolution, so we only consider its variants and modifications. The reason for this is that, as latest competitions (such as CEC 2017, CEC 2021, CEC 2022) show, other techniques, such as genetic algorithms, particle swarm optimization, and other biology inspired algorithms are not competitive in a single-objective black-box setup. The only exclusion may be the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) algorithm and its hybrids with DE.

The main idea of the differential evolution algorithm proposed in [15] is to use difference vectors between individuals to produce new solutions during mutation. The algorithm starts with the initialization of N individuals, $x_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,D})$, $i = 1, \ldots, N$, within the bounds:

$$S = \{x_i \in R^D | x_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,D}) : x_{i,j} \in [x_{lb,j}, x_{ub,j}]\}, \quad (1)$$

where D is the number of variables. Most of the studies use the uniform random generation of individuals:

$$x_{i,j} = x_{lb,j} + rand \times (x_{ub,j} - x_{lb,j}). \quad (2)$$

After initialization and target function evaluation, the mutation step begins. Many mutation strategies have been proposed for DE, but the two most popular ones are rand/1 and current-to-pbest/1:

$$v_{i,j} = x_{r1,j} + F \times (x_{r2,j} - x_{r3,j}), \quad (3)$$

$$v_{i,j} = x_{i,j} + F \times (x_{pbest,j} - x_{i,j}) + F \times (x_{r1,j} - x_{r2,j}), \quad (4)$$

where x_i is called the target vector, v_i is the mutant or donor vector, F is the scaling factor, $pbest$ is the index of one of the p% best individuals, and indexes $r1$, $r2$, $r3$ and $pbest$ are chosen randomly so that they are different from each other and the current individual with index i. The mutation is performed for all solutions in the population, $i = 1, 2, \ldots, N$.

After a set of donor vectors is generated, the crossover step is performed. The most widely used method is binomial crossover, described as follows:

$$u_{i,j} = \begin{cases} v_{i,j}, & \text{if } rand(0,1) < Cr \text{ or } j = jrand \\ x_{i,j}, & \text{otherwise} \end{cases}, \tag{5}$$

where u_i is called the trial vector, Cr is the crossover rate parameter, and $jrand \in [1, D]$, generated randomly. In other words, binomial crossover combines the information in target vector x_i and mutant vector v_i to produce the trial vector, and $jrand$ is required to make sure that the trial vector is different from the target one.

The mutation may generate vectors which are outside the $[x_{lb,j}, x_{ub,j}]$ range, so a bound constraint handling technique is required in DE. One of the widely used ones is the midpoint-target method [16], which works as follows:

$$u_{i,j} = \begin{cases} \frac{x_{lb,j}+x_{i,j}}{2}, & \text{if } v_{i,j} < x_{lb,j} \\ \frac{x_{ub,j}+x_{i,j}}{2}, & \text{if } v_{i,j} > x_{ub,j} \end{cases}. \tag{6}$$

If some of the components of the trial vector overshoot the boundaries, then the coordinate of the parent (target vector) is used to move towards the boundary without reaching it. We note that this step can be applied after mutation or after crossover.

The last step in DE is the selection, which here plays the role of a replacement mechanism. If the trial vector, u_i, is better in terms of the target function value compared to target vector x_i, then replacement occurs:

$$x_i = \begin{cases} u_i, & \text{if } f(u_i) \leq f(x_i) \\ x_i, & \text{if } f(u_i) > f(x_i) \end{cases}. \tag{7}$$

Although this selection mechanism is known to be simple and efficient, there are some attempts to modify it, for example, by using information about neighbuorhood [17] or replacing other individuals [12].

2.2. Parameter Adaptation in Differential Evolution

Most of the modern studies which are focused on DE or apply DE to some problem use one of the previously proposed parameter adaptation techniques. Here, it would be impractical to cover all of the existing parameter adaptation techniques, so interested readers are advised to refer to surveys such as [18,19] as well as some recent papers considering the problem [7].

Some of the earliest successful experiments on parameter adaptation were presented in the jDE algorithm [3], where, on each iteration t before the mutation and crossover steps of the search loop, new control parameters are generated in the following way:

$$F_{i,t+1} = \begin{cases} random(F_l, F_u), & \text{if } random(0,1) < \tau_1 \\ F_{i,t}, & \text{otherwise} \end{cases}, \tag{8}$$

$$CR_{i,t+1} = \begin{cases} random(0,1), & \text{if } random(0,1) < \tau_2 \\ CR_{i,t}, & \text{otherwise} \end{cases}. \tag{9}$$

In the equations above, values $F_l = 0.1$ and $F_u = 0.9$ represent the lower and upper boundaries for F, and the parameters controlling the frequency of change τ_1 and τ_2 are set to 0.1. If, during selection, improvement occurs, then the new values for F and Cr are saved. This technique was successfully applied in several highly efficient versions of jDE, including the jDE100 [20] and j2020 [21] algorithms, which have shown competitive results in CEC 2019 and 2020 competitions, respectively.

Another important algorithm is JADE, proposed in [4]. Despite the fact that JADE introduced one of the most used mutations, current-to-pbest/1, it also proposed sampling of F and Cr in a way similar to the SaDE algorithm [22]. However, instead of using fixed means, in JADE, F and Cr are sampled with Cauchy and normal distributions around μ_F and μ_{Cr}:

$$\begin{cases} F = randc(\mu_F, 0.1) \\ Cr = randn(\mu_{Cr}, 0.1) \end{cases}, \tag{10}$$

where $randc(\mu, s)$ and $randn(\mu, s)$ are random numbers sampled from Cauchy and normal distribution with location parameter μ and scale parameter $s = 0.1$. The update of μ values is performed as follows:

$$\begin{cases} \mu_F = (1-c) \cdot \mu_F + c \cdot mean_L(S_F) \\ \mu_{Cr} = (1-c) \cdot \mu_{Cr} + c \cdot mean_A(S_{Cr}) \end{cases}, \tag{11}$$

where $mean_A()$ is the arithmetic mean, c is the update parameter, S_F and S_{Cr} contain the successful values of F and Cr, i.e., values which produce individuals better than parents, and $mean_L()$ is the Lehmer mean:

$$mean_L(S_F) = \frac{\sum_{F \in S_F} F^2}{\sum_{F \in S_F} F}. \tag{12}$$

The success of JADE's parameter adaptation has led to the development of the even more efficient Success History Adaptation (SHA), proposed in the SHADE algorithm [5]. Unlike JADE, in SHADE, there are H memory cells, each containing a pair $(M_{F,h}, M_{Cr,h})$, i.e., mean values to be used for sampling F and Cr. These values are used in a similar way as in Equation (10):

$$\begin{cases} F = randc(M_{F,h}, 0.1) \\ Cr = randn(M_{Cr,h}, 0.1) \end{cases}. \tag{13}$$

The h index is randomly sampled from $[1, H]$ before each mutation and crossover. At the end of the generation, the new means are calculated using a weighted Lehmer mean [23], where weights are set based on the improvements $\Delta f = |f(u_j) - f(x_j)|$, stored in $S_{\Delta f}$:

$$mean_{wL,F} = \frac{\sum_{j=1}^{|S_F|} w_j S_{F,j}^2}{\sum_{j=1}^{|S_F|} w_j S_{F,j}}, \; mean_{wL,Cr} = \frac{\sum_{j=1}^{|S_{Cr}|} w_j S_{Cr,j}^2}{\sum_{j=1}^{|S_{Cr}|} w_j S_{Cr,j}}, \tag{14}$$

where $w_j = \frac{S_{\Delta f_j}}{\sum_{k=1}^{|S|} S_{\Delta f_k}}$. Next, one of the memory cells with index k, iterated every generation, is updated:

$$\begin{cases} M_{F,k}^{t+1} = 0.5(M_{F,k}^t + mean_{wL,F}) \\ M_{Cr,k}^{t+1} = 0.5(M_{Cr,k}^t + mean_{wL,Cr}) \end{cases}, \tag{15}$$

where t is the current generation number. If $k > H$, then it is set to $k = 1$.

As for the population size, in L-SHADE [24], the Linear Population Size Reduction (LPSR) was proposed, which is a widely used technique nowadays. The LPSR works by determining new population size at every generation as follows:

$$N_{g+1} = round\left(\frac{N_{min} - N_{max}}{NFE_{max}} NFE\right) + N_{max}, \tag{16}$$

where NFE is the current number of target function evaluations, NFE_{max} is the total available computational resource, N_{max} and N_{min} are the initial and final number of individuals, g is the generation number. The main idea of LPSR consists in spreading across the search space at the beginning and concentrating at the end of the search. LPSR allows achievement

of significant improvements in performance if the computational resource limit is known. However, some other studies modify it and use non-linear reduction; for example, [10].

In [9], it was shown that adding tournament or rank-based selection strategies to sample the indexes of individuals for further mutation may be beneficial. The exponential rank-based selection was implemented by selecting an individual depending on its fitness in a sorted array, with the ranks assigned as follows:

$$rank_i = e^{\frac{-kp \cdot i}{N}}, \tag{17}$$

where kp is the parameter controlling the pressure and i is the individual number. Larger ranks are assigned to better individuals, and discrete distribution is used for selection.

It is hard to underestimate the importance of the L-SHADE (SHADE with LPSR) algorithm: the success history adaptation was further developed and improved in many studies [6]. To provide some examples, the jSO algorithm [25] proposed specific rules for parameter adaptation, limiting the F and Cr values depending on computational resource, and used in L-SHADE-RSP [9] with rank-based selection, proposed in [26], DB-LSHADE proposed distance-based adaptation, where weights are based on the Euclidean distance instead of fitness improvement [27,28]. In [7], the effect of modified Lehmer means was considered. Nevertheless, most of the adaptive DE variants tend to apply small modifications to the SHA general scheme without changing it dramatically. The main problem of SHA is that it follows immediate improvements in fitness and tunes F and Cr to them, which is a greedy approach. The biased adaptation introduced with the Lehmer mean in the JADE algorithm was, in fact, proposed to eliminate this effect by sampling higher F values than they should be. A more detailed consideration of the effects of bias on parameter adaptation was performed in [7].

Some other modifications of DE include population regeneration in case of premature convergence [29], modifications for binary search space [30], Gaussian–Cauchy mutation [31] and using an ensemble of mutation and crossover operators [32]. Some works focus on proposing new mutation strategies, such as the triple competitive approach recently proposed in [33]. In [34], the Unbounded DE (UDE) was proposed, where all the individuals generated throughout the whole search space are saved and used in the search. Such an approach shows some interesting effects of using old solutions in the search process.

3. Proposed Approach

Developing new algorithmic schemes, including parameter adaptation techniques, could be a tedious process requiring prolonged experimentation and brand new ideas. In a recent study [8], an attempt to use genetic programming solving symbolic regression as a knowledge extraction technique was performed, and as a result, it was discovered that the success rate can be an important source of information for the adaptation of scaling factor F. Based on this idea, as well as some additional experiments, success rate-based adaptation is proposed.

Success rate is calculated as the ratio of the number of successful solutions NS to the current population size N:

$$SR = \frac{NS}{N}. \tag{18}$$

In other words, NS is equal to the amount of elements in S_F or S_{Cr}. Success rate shows, in general, how well the algorithm performs: if the search is efficient, then NS is quite high, and if the population is stuck in a local optimum or optima, then it is low. Success rate is used to calculate the MF value, which acts as a mean for sampling F:

$$MF = SR^{1/c}, \tag{19}$$

where c is the parameter value. Some of the solutions found by GP used the square root of the success rate. However, further experiments have shown that using the cube root or even larger c values may produce better results. The MF value is then used as follows:

$$F = randc(MF, 0.1). \tag{20}$$

Joining together all the above equations, the scaling factor for every mutation can be calculated as follows:

$$F = randc\left(\left(\frac{NS}{N}\right)^{1/c}, 0.1\right). \tag{21}$$

As in many other methods, if $F > 1$, it is set to 1, and if $F < 0$, it is sampled again until positive. Such a technique is very simple compared to SHADE, or even JADE and jDE, as it does not require additional values to be stored or the average to be calculated. Moreover, it can be applied to any DE algorithm without significant effort.

Figure 1 shows the dependence of MF on SR with different c parameter values.

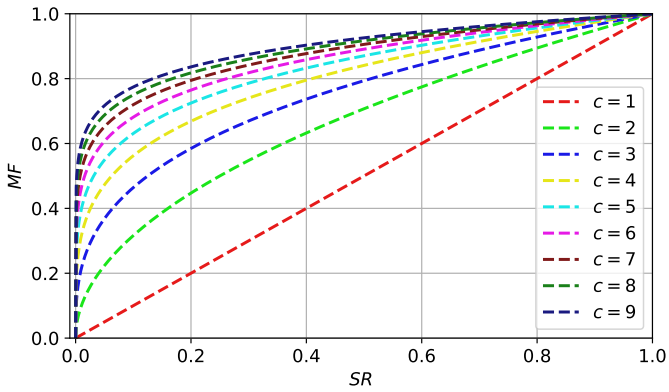

Figure 1. MF values for different c parameter values.

As shown in Figure 1, small success rate values (<0.05) lead to MF values between 0 and 0.5, while larger success rates result in MF being larger than 0.5.

The main advantage of the proposed technique is its simplicity: to apply it, a single Equation (21) is required, whereas SHA needs to store memory cells, successful parameter values, calculate weights, Lehmer means and update memory cells (Equations (13)–(15)). This creates additional computational effort. To illustrate this, in Figure 2, the two techniques are compared. In the next section, when the modifications of DE variants are considered, the steps of success history adaptation are replaced by a single step of success rate adaptation.

$$F = randc\left(\left(\frac{NC}{N}\right)^{1/c}, 0.1\right)$$

$$F = randc(M_{F,k}, 0.1)$$
$$mean_{wL} = \frac{\sum_{j=1}^{|S|} w_j \cdot S_j^2}{\sum_{j=1}^{|S|} w_j \cdot S_j}$$
$$w_j = \frac{\Delta f_j}{\sum_{k=1}^{|S|} \Delta f_k}$$
$$\Delta f_j = |f(u) - f(x_j)|$$
$$M_{F,k}^{t+1} = 0.5(M_{F,k}^{t} + mean_{(wL,F)})$$
H memory cells

Figure 2. Comparison of steps required for success rate-based adaptation and success history-based adaptation.

The proposed technique was implemented for L-SHADE-RSP, NL-SHADE-RSP, NL-SHADE-LBC, L-NTADE and other algorithms, and the results are considered in the next section.

4. Experimental Setup and Results

4.1. Benchmark Functions and Parameters

The main purpose of the experiments in this study was to determine the efficiency of success rate-based adaptation in different scenarios. For this, two benchmark sets were chosen, namely the CEC 2017 [13] and 2022 [14] Single Objective Bound Constrained Numerical Optimization problems. These benchmarks have different numbers of functions, 30 and 12, respectively, different dimensions and computational resource. For CEC 2017, the test functions were defined for $D = 10$, 30, 50 and 100, and for CEC 2022 $D = 10$ and 20. The number of available function evaluations was set to NFE_{max} = 10,000D for CEC 2017, and for CEC 2022 it was set to $NFE_{max} = 2 \times 10^5$ for 10D and $NFE_{max} = 1 \times 10^6$ for 20D. All the experiments with algorithms and their modifications were performed according to competition rules.

All the tested algorithms were implemented in C++, and ran on eight AMD Ryzen 3700 PRO processors with eight cores each under Ubuntu Linux 20.04; the experiments were paralleled using OpenMPI 4.0.3, and post-processing was performed using Python 3.8.5.

To compare different results, two main techniques were applied. The first was the Mann–Whitney rank sum statistical test with normal approximation and tie-breaking, with significance level set to $p = 0.01$. In addition to the result of the test (win/tie/loss), the standard Z score values were calculated and summed over all test functions. We note that $Z = \pm 2.58$ corresponds to $p = 0.01$. The second is the Friedman ranking procedure, applied in the Friedman statistical test. Here, it was used to compare a set of algorithms or their variants, and the ranks were assigned to the results obtained on every run and function independently, and then summed together. We note that unlike CEC 2017, in CEC 2022, the ranking of results included not only the best achieved values, but also the total computational resource spent, and this ranking was used in both the Mann–Whitney test and Friedman ranking.

4.2. Numerical Results

In order to consider the proposed modification from different points, in the following subsections, several aspects were considered, including the effects of SR-based adaptation on the performance of different existing algorithms when success history-based adaptation is replaced; the influence of the available computational resource and mutation strategies; comparison with the best methods on two sets of benchmark functions. Also, a visualization of the parameter adaptation process was considered for a better understanding of the effects on the differential evolution algorithm.

4.2.1. Modification of Existing Algorithms

As was mentioned above, the success rate-based adaptation for F replaced the success history adaptation in four tested algorithms. As for the crossover rate Cr adaptation, it was unchanged. In the case of the L-SHADE-RSP algorithm, which uses a specific mutation strategy with F and $F2$, when using the success rate, $F2$ was set equal to F, and specific rules based on the current resource from jSO algorithm were removed. The parameters for all the algorithms were set according to those used in corresponding papers.

The majority of modern DE algorithms rely on the success history adaptation method introduced in SHADE, so the purpose of the study was to compare the adaptation methods, and not specific algorithms. For this purpose, in Tables 1–10, we modified L-SHADE-RSP, NL-SHADE-RSP, NL-SHADE-LBC, L-NTADE, jSO, LSHADE-SPACMA, APGSK-IMODE, MLS-LSHADE and MadDE with success rate-based adaptation and compared the results.

Tables 1 and 2 present the comparison of the original L-SHADE-RSP with the modified L-SHADE-RSP (SR) with different c values. Each cell in the tables contains the number of

wins/ties/losses and the total standard score in brackets. Larger standard score means that the modified algorithm is better.

Table 1. L-SHADE-RSP (SR) vs. L-SHADE-RSP, CEC 2017.

c	$10D$	$30D$	$50D$	$100D$
1	3/18/9 (−45.16)	0/9/21 (−157.01)	0/7/23 (−163.79)	2/5/23 (−183.13)
2	3/23/4 (−12.68)	4/11/15 (−74.72)	4/8/18 (−87.52)	4/9/17 (−91.82)
3	5/23/2 (4.32)	5/16/9 (−28.41)	5/15/10 (−25.87)	8/12/10 (−16.28)
4	4/26/0 (19.57)	2/23/5 (−18.11)	5/20/5 (2.37)	11/11/8 (11.60)
5	5/23/2 (14.58)	3/20/7 (−31.34)	1/25/4 (−11.09)	12/10/8 (16.57)
6	6/19/5 (8.65)	2/18/10 (−41.83)	3/20/7 (−33.75)	12/11/7 (4.25)

Table 2. L-SHADE-RSP (SR) vs. L-SHADE-RSP, CEC 2022.

c	$10D$	$20D$
1	7/3/2 (29.61)	3/3/6 (−11.30)
2	7/3/2 (30.91)	4/4/4 (4.29)
3	6/3/3 (21.43)	4/6/2 (18.58)
4	4/2/6 (−8.13)	2/8/2 (−3.00)
5	4/4/4 (−8.23)	2/6/4 (−14.21)
6	3/5/4 (−9.17)	1/7/4 (−16.18)

As can be seen from Tables 1 and 2, the c parameter significantly influences performance, and if for CEC 2017 $c = 1$, i.e., $MF = SR$, is a bad choice; for CEC 2022 it is a reasonable setting for $10D$. The setting which leads to good performance is $c = 4$ for CEC 2017. In this case, for $10D$, $50D$ and $100D$, there are more improvements than losses, but for $30D$ the success history adaptation works better. In the case of the CEC 2022 benchmark, the best choice is$c = 3$.

In Tables 3 and 4, the comparison of the basic NL-SHADE-RSP and the modified NL-SHADE-RSP (SR) is presented.

Table 3. NL-SHADE-RSP (SR) vs. NL-SHADE-RSP, CEC 2017.

c	$10D$	$30D$	$50D$	$100D$
1	0/13/17 (−77.34)	1/6/23 (−134.10)	1/9/20 (−128.17)	6/6/18 (−91.88)
2	0/25/5 (−18.32)	7/21/2 (17.65)	14/13/3 (45.81)	20/6/4 (89.04)
3	2/27/1 (8.46)	14/14/2 (64.24)	15/13/2 (100.78)	22/7/1 (134.60)
4	7/22/1 (28.38)	15/14/1 (81.37)	17/11/2 (111.83)	22/6/2 (141.35)
5	8/21/1 (33.83)	16/13/1 (73.35)	18/10/2 (111.43)	21/7/2 (139.24)
6	6/24/0 (29.73)	13/14/3 (66.20)	17/11/2 (111.59)	22/5/3 (135.73)

Table 4. NL-SHADE-RSP (SR) vs. NL-SHADE-RSP, CEC 2022.

c	$10D$	$20D$
1	4/4/4 (2.46)	0/6/6 (−39.78)
2	5/6/1 (14.26)	1/8/3 (−8.21)
3	6/2/4 (14.37)	5/4/3 (5.05)
4	6/2/4 (8.26)	5/3/4 (4.18)
5	5/3/4 (9.20)	5/3/4 (6.02)
6	6/2/4 (11.19)	5/4/3 (8.10)

The situation with NL-SHADE-RSP is quite different as this method was not tuned for CEC 2017 or CEC 2022. Nevertheless, small values of the c parameter lead to poor

performance on both benchmarks, and increasing c up to 3 or 4 produces much better results. After $c = 4$, performance gains become smaller.

Tables 5 and 6 show the results of NL-SHADE-LBC and its modified version.

Table 5. NL-SHADE-LBC (SR) vs. NL-SHADE-LBC, CEC 2017.

c	10D	30D	50D	100D
1	0/17/13 (−63.46)	2/8/20 (−138.49)	5/8/17 (−103.11)	7/4/19 (−108.71)
2	2/24/4 (−13.13)	7/13/10 (−15.57)	18/3/9 (62.92)	16/8/6 (63.66)
3	4/24/2 (21.91)	14/12/4 (73.17)	23/6/1 (161.08)	23/5/2 (161.72)
4	8/21/1 (40.55)	16/13/1 (97.39)	25/4/1 (185.04)	25/5/0 (190.54)
5	10/20/0 (53.79)	16/13/1 (106.44)	25/5/0 (186.03)	27/3/0 (199.10)
6	10/20/0 (56.07)	17/12/1 (106.95)	25/4/1 (176.66)	26/4/0 (193.90)

Table 6. NL-SHADE-LBC (SR) vs. NL-SHADE-LBC, CEC 2022.

c	10D	20D
1	6/3/3 (26.91)	3/3/6 (-19.44)
2	2/3/7 (−23.34)	1/4/7 (−37.09)
3	3/2/7 (−25.30)	2/5/5 (−20.46)
4	2/3/7 (−26.51)	3/4/5 (−17.18)
5	2/3/7 (−28.37)	3/4/5 (−20.26)
6	2/3/7 (−28.53)	3/4/5 (−18.72)

As can be seen from Tables 5 and 6, the situation with NL-SHADE-LBC is quite different. As this method was specifically developed for the CEC 2022 benchmark, its parameter adaptation was not tuned for the CEC 2017 benchmark, and applying success rate-based adaptation significantly improves results, with up to 27 improvements out of 30 functions in 100D. However, for CEC 2022, the success rate adaptation was not able to deliver better performance, but still $c = 4$ or higher is a reasonable choice for both benchmarks. The reason why SR-based adaptation worked worse with NL-SHADE-LBC is that the parameter adaptation of this algorithm was specifically tuned for the CEC 2022 benchmark.

Tables 7 and 8 contain the results of the L-NTADE algorithm and its modification.

Table 7. L-NTADE (SR) vs. L-NTADE, CEC 2017.

c	10D	30D	50D	100D
1	2/10/18 (−105.68)	0/8/22 (−192.68)	0/4/26 (−218.98)	1/2/27 (−219.12)
2	6/15/9 (−11.94)	3/14/13 (−65.01)	4/8/18 (−108.84)	5/3/22 (−124.60)
3	9/15/6 (22.08)	12/15/3 (43.24)	8/17/5 (19.67)	8/9/13 (-6.39)
4	11/14/5 (37.30)	14/15/1 (75.39)	15/11/4 (69.26)	15/11/4 (70.63)
5	10/16/4 (44.53)	14/15/1 (73.95)	17/10/3 (67.77)	15/11/4 (65.51)
6	11/17/2 (36.40)	9/19/2 (48.88)	13/11/6 (25.88)	11/12/7 (24.52)

Table 8. L-NTADE (SR) vs. L-NTADE, CEC 2022.

c	10D	20D
1	5/3/4 (11.08)	3/2/7 (−22.99)
2	6/4/2 (25.53)	3/3/6 (−6.99)
3	6/4/2 (25.45)	3/5/4 (1.07)
4	5/5/2 (26.17)	2/6/4 (−1.82)
5	3/8/1 (13.29)	2/6/4 (−10.64)
6	2/6/4 (−10.49)	2/7/3 (−12.62)

Although L-NTADE has two populations and has a different algorithmic scheme, applying success rate-based adaptation improved the results in both benchmarks. As before, small c values are inefficient. However, setting $c = 4$ for CEC 2017 or probably even larger values offers significant performance benefits.

In order to test the applicability of the proposed approach to other algorithms, several of them for which the source codes were available were modified and the modifications were compared to the original results. For the CEC 2017 benchmark, in the jSO (derived from L-SHADE) [25], the mutation step, which uses two factors, F and $F2$, was modified to have only a single one, and the control rules for F were deactivated. For LSHADE-SPACMA [35], the scaling factor sampling in the DE part was replaced by Equation (19). The comparison is shown in Table 9.

Table 9. Comparison of other modified approach vs. original versions, CEC 2017.

Algorithm	10D	30D	50D	100D
jSO (SR, $c = 4$) vs. jSO [25]	1/21/8 (−37.08)	8/16/6 (21.04)	6/14/10 (−25.27)	11/9/10 (−2.29)
LSHADE-SPACMA (SR, $c = 4$) vs. LSHADE-SPACMA [35]	3/22/5 (−22.61)	1/24/5 (−41.67)	0/22/8 (−64.32)	0/23/7 (−42.11)

As can be seen from Table 9, the modification of jSO with $c = 4$ may be beneficial in some cases, for example, in the 30D scenario, but in 10D the number of losses was much larger than the number of wins. As for LSHADE-SPACMA, using success rate-based adaptation leads to decreased performance. The main reason why SR-based adaptation failed to improve the performance in these cases could be that both jSO and LSHADE-SPACMA were specifically tuned for the CEC 2017 benchmark, and the SR-based adaptation is a more general approach, which was not tuned for these particular algorithms.

In Table 10, the same modification was applied to APGSK-IMODE (only in IMODE part) [36], MLS-LSHADE [37] and MadDE [38]. Due to a simpler benchmark in terms of computation time, the experiments were performed with different c values.

Table 10. Comparison of other modified approaches vs. original versions, CEC 2022.

Algorithm	10D	20D
APGSK-IMODE (SR, $c = 2$) vs. APGSK-IMODE [36]	4/5/3 (3.66)	1/5/6 (−24.33)
APGSK-IMODE (SR, $c = 3$) vs. APGSK-IMODE [36]	4/4/4 (12.88)	1/8/3 (−14.19)
APGSK-IMODE (SR, $c = 4$) vs. APGSK-IMODE [36]	3/4/5 (−8.81)	1/8/3 (−12.24)
APGSK-IMODE (SR, $c = 5$) vs. APGSK-IMODE [36]	3/4/5 (−8.81)	1/8/3 (−12.24)
MLS-LSHADE (SR, $c = 2$) vs. MLS-LSHADE [37]	4/3/5 (−2.05)	0/4/8 (−49.53)
MLS-LSHADE (SR, $c = 3$) vs. MLS-LSHADE [37]	2/3/7 (−29.50)	0/6/6 (−41.34)
MLS-LSHADE (SR, $c = 4$) vs. MLS-LSHADE [37]	3/1/8 (−32.78)	0/5/7 (−41.91)
MLS-LSHADE (SR, $c = 5$) vs. MLS-LSHADE [37]	1/3/8 (−37.41)	0/5/7 (−40.96)

Table 10. *Cont.*

Algorithm	10D	20D
MadDE (SR, $c = 2$) vs. MadDE [38]	5/4/3 (12.46)	2/6/4 (−9.56)
MadDE (SR, $c = 3$) vs. MadDE [38]	4/5/3 (14.78)	3/8/1 (13.48)
MadDE (SR, $c = 4$) vs. MadDE [38]	4/5/3 (13.10)	3/8/1 (15.39)
MadDE (SR, $c = 5$) vs. MadDE [38]	3/6/3 (4.88)	3/6/3 (0.68)

The effect of success rate-based scaling factor adaptation is different for different algorithms. For example, the performance of APGSK-IMODE was improved only in the 10D case, with $c = 2$ and $c = 3$, but for MLS-LSHADE the effect was mainly negative. As for the MadDE algorithm, it was mostly improved by the modification, especially for $c = 3$ and $c = 4$.

4.2.2. Effect of the Available Computational Resource

The main difference between the two considered benchmarks is the number of computations available for every function. The experiments shown above demonstrated that there is a possibility that amount of resource may influence the performance of success rate-based adaptation. To test this hypothesis, a set of experiments was performed for the L-NTADE algorithm on CEC 2022, where the number of function evaluations NFE_{max} was decreased to 10%, 20% and so on up to 100%. Figure 2 shows the Friedman ranking of the results in the form of heatmaps.

In Figure 3, the ranking was performed independently in every column, as there is no sense in comparing algorithms with different available resource. The best ranks are shown in red. In the 10D case, if the resource is relatively small, the success rate adaptation is significantly better than SHA. However, as the NFE_{max} grows, the SHA may take the lead. In the 20D case, however, the SR adaptation is always better, and the best settings are $c = 5$ or $c = 6$, although $c = 4$ is very similar. From these results, it can be concluded that, first of all, success rate-based adaptation is not highly influenced by available resource, but there is a difference in performance compared to SHA.

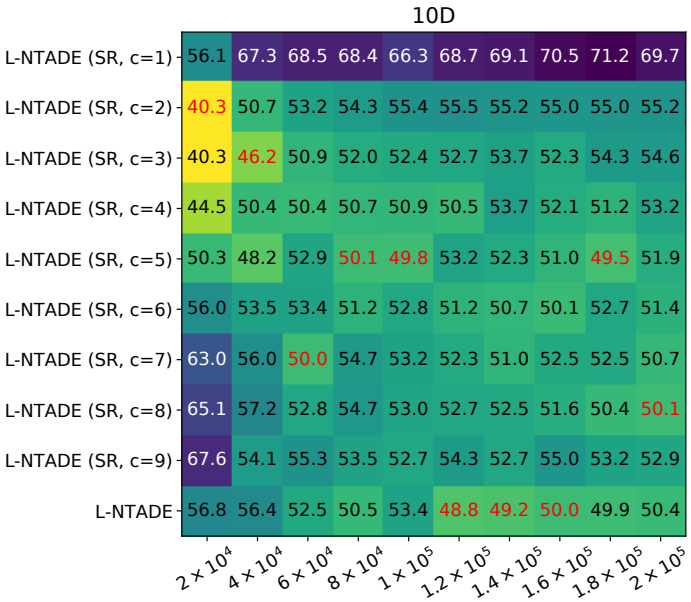

Figure 3. *Cont.*

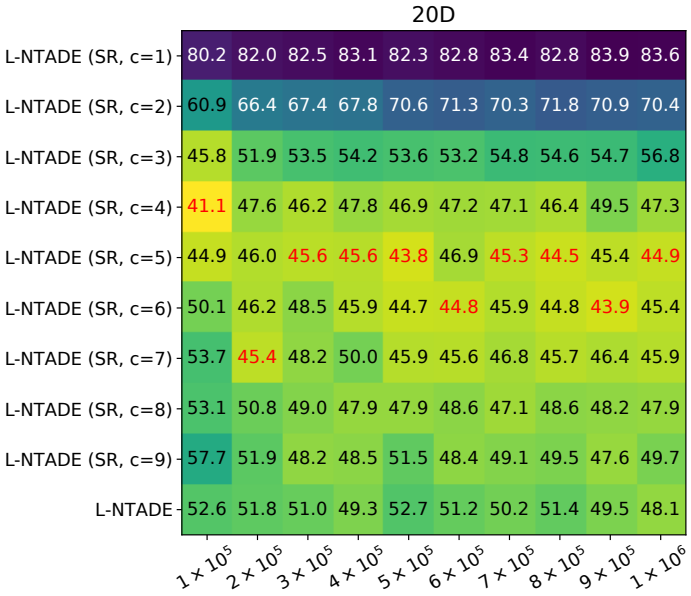

Figure 3. Comparison of L-NTADE with and without success rate adaptation with different available resource NFE_{max}, Friedman ranking.

4.2.3. Visualization of Parameter Adaptation Process

For a better understanding of what SR actually does during the search process, Figures 4 and 5 show the graphs of F values set by the SHA and SR methods in the case of the L-SHADE-RSP algorithm.

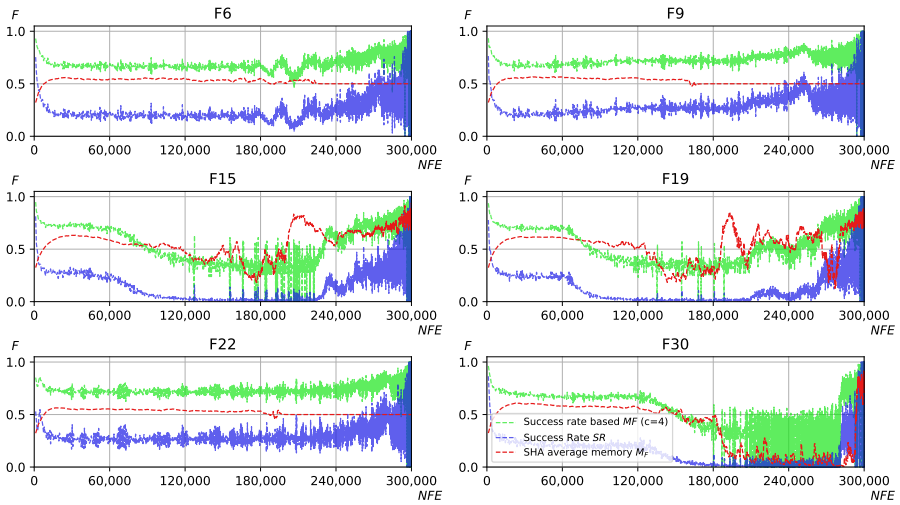

Figure 4. Graphs of parameter adaptation of L-SHADE-RSP with and without success rate adaptation, CEC 2017, 30D, selected functions.

The graphs in Figure 4 show that the behavior of the two considered adaptation methods is quite different, and sometimes even opposite. For example, at the beginning of the search, the success rate is high, and so is the MF value, and they both decrease, while SHA makes larger F values. If the success rate drops close to 0, this makes the MF values oscillate between 0 and 0.5, while SHA slowly tunes values in the memory cells, as seen on functions F15 and F19. If the search process is going well, SR tends to maintain relatively high F values, and switches to an oscillating mode if the success rate is low.

Figure 5 shows similar trends, but they are seen much better due to larger resource. For example, for functions F4, F7 and F8, the success history adaptation leads to memory cell values close to zero at some points in the middle of the search, but success rate

adaptation tries every possible F value from 0 to around 0.4 when the success rate drops low. Also, oscillation tends to become higher closer to the end of the search. This is simply the effect of smaller a population size.

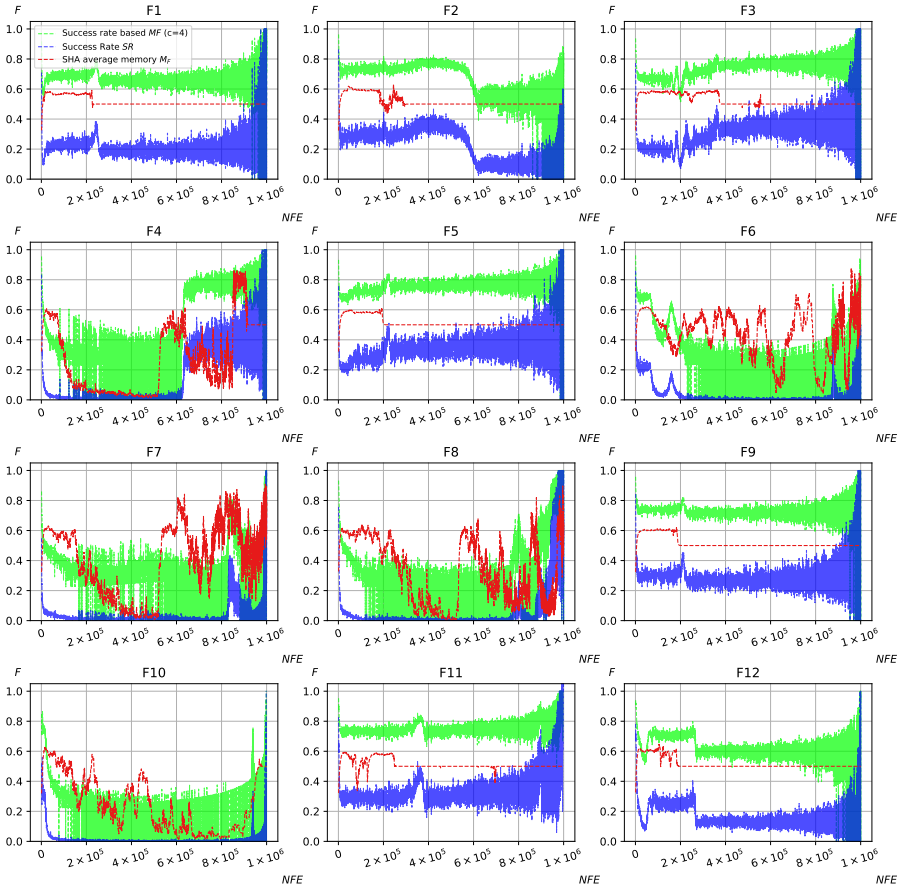

Figure 5. Graphs of parameter adaptation of L-SHADE-RSP with and without success rate adaptation, CEC 2022, 20D.

4.2.4. Success Rate-Based Adaptation With Different Mutation Strategies

One of the possible explanations of the high performance of success rate-based adaptation for F could be that it works in combination with the current-to-pbest strategy. If the success rate is low, smaller F values force more exploration, while F closer to 1 leads to a move towards one of the p% best solutions. To test this effect of success rate-based adaptation, a set of experiments was performed on the standard L-SHADE algorithm with different mutation strategies, such as rand/1, rand/2, current-to-best/1, current-to-rand/1, best/1 and best/2. The c parameter was also altered from 1 to 6, and the original L-SHADE was tested. Next, the Friedman ranking procedure was applied to compare the performance of SHA with SR. The ranks were assigned for each mutation strategy (column) independently. The experiments were repeated for both CEC 2017 and CEC 2022 benchmarks. The parameters of L-SHADE were the following: $N = 60 \times D^{\frac{2}{3}}$, $pbest = 0.2$, $H = 5$, initial $M_F = 0.5$, $M_{Cr} = 0.5$, archive was not used. Figures 6 and 7 contain the heatmaps of the comparison.

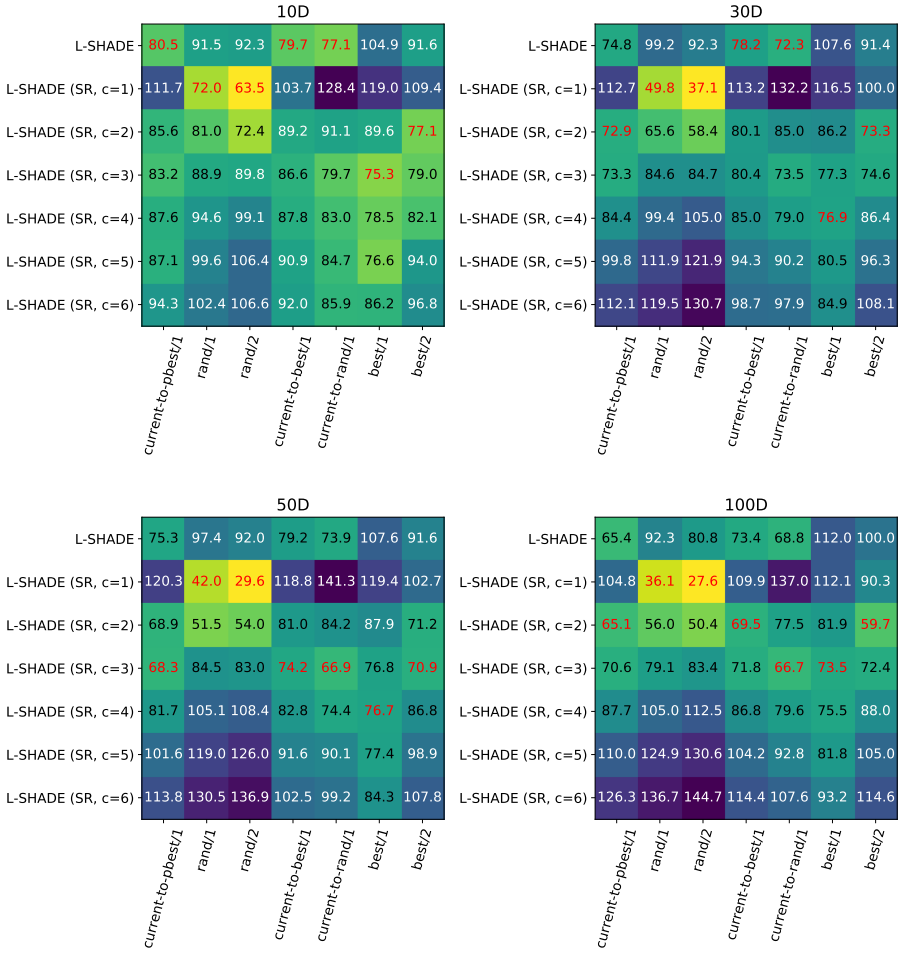

Figure 6. Comparison of L-SHADE with different strategies, with and without success rate adaptation, CEC 2017, Friedman ranking.

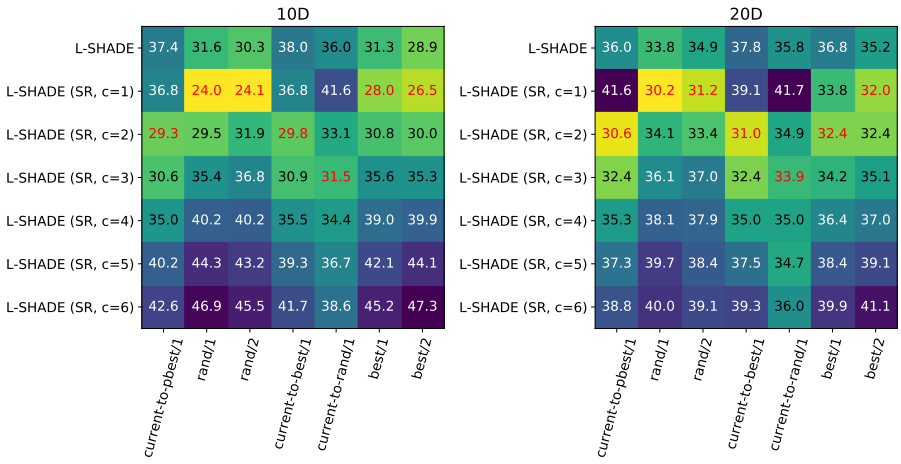

Figure 7. Comparison of L-SHADE with different strategies, with and without success rate adaptation, CEC 2022, Friedman ranking.

The heatmaps for the CEC 2017 benchmark show that in low-dimensional cases like 10D and 30D, the SHA may perform better than SR, but only for current-to-pbest/1, current-to-best/1 and current-to-rand/1. As for the other strategies, rand/1 and rand/2 performed much better with the modified approach, and so did best/1 and best/2, although the difference here was smaller. In high-dimensional cases, SR is always performing better, especially when the rand/2 strategy is used. Different strategies prefer different c values;

for example, rand/1 and rand/2 only use $c = 1$, i.e., linear dependence. In fact, this means that the success rate is used directly. Other strategies, such as best/1 and best/2, prefer $c = 2$, $c = 3$ or $c = 4$, but this value is always smaller for best/2 compared to best/1. The same is true when comparing rand/1 and rand/2: the effect of smaller c values is larger in the case of rand/2. The current-to-best and current-to-rand strategies are similar to current-to-pbest ones, and the best settings for them are close. The comparison on CEC 2022 demonstrates the same trends, but in general smaller c values are preferred for all strategies except current-to-rand ones. Also, on this benchmark, the SHA is never the best choice.

To compare different strategies with current-to-pbest, the best performing ones from every dimension and every benchmark were chosen and compared to current-to-pbest with SR. The standard SHA was not considered here. The Mann–Whitney statistical tests for both benchmarks are shown in Tables 11 and 12.

Table 11. L-SHADE (SR) with different mutation strategies, current-to-pbest strategy vs. best variants, CEC 2017.

Strategy	10*D*	30*D*	50*D*	100*D*
current-to-pbest/1 vs. rand/1	10/11/9 (26.18)	19/9/2 (110.36)	24/4/2 (150.35)	18/8/4 (133.91)
current-to-pbest/1 vs. rand/2	11/16/3 (56.51)	20/9/1 (134.71)	25/3/2 (174.02)	20/6/4 (146.97)
current-to-pbest/1 vs. current-to-best/1	15/10/5 (45.26)	5/22/3 (17.98)	4/22/4 (3.20)	11/10/9 (19.94)
current-to-pbest/1 vs. current-to-rand/1	11/17/2 (52.97)	19/11/0 (119.34)	19/9/2 (135.04)	22/7/1 (164.06)
current-to-pbest/1 vs. best/1	9/17/4 (20.28)	15/11/4 (80.90)	23/4/3 (108.31)	20/6/4 (107.48)
current-to-pbest/1 vs. best/2	11/16/3 (27.58)	17/11/2 (80.53)	24/4/2 (123.93)	20/6/4 (108.48)

Table 12. L-SHADE (SR) with different mutation strategies, current-to-pbest strategy vs. best variants, CEC 2022.

Strategy	10*D*	20*D*
current-to-pbest/1 vs. rand/1	4/5/3 (1.77)	3/5/4 (−5.97)
current-to-pbest/1 vs. rand/2	4/5/3 (8.87)	2/7/3 (−1.72)
current-to-pbest/1 vs. current-to-best/1	1/7/4 (−15.58)	1/8/3 (−8.00)
current-to-pbest/1 vs. current-to-rand/1	7/4/1 (35.73)	5/6/1 (22.00)
current-to-pbest/1 vs. best/1	5/3/4 (1.91)	5/4/3 (12.18)
current-to-pbest/1 vs. best/2	3/4/5 (−10.48)	6/2/4 (13.10)

In the case of the CEC 2017 benchmark in Table 11, the current-to-pbest strategy dominates other strategies in terms of performance in all cases. However, the closest competitor is the current-to-best which performed similar in the 50D case, which might indicate a non-optimal choice of the *pbest* parameter for current-to-best. As for the results on CEC 2022, the current-to-pbest outperformed most of the strategies but the current-to-best had several wins in both the 10D and 30D cases. Also, the rand/1 strategy has shown some competitive performance in 20D, as well as best/2 in 10D. Considering the improvements that success rate-based adaptation was able to deliver for these strategies

compared to success history adaptation (Figures 6 and 7), one may conclude that SR can be efficiently applied to other strategies, and even make them competitive.

4.2.5. Comparison With Alternative Approaches

Tables 13–15 contain the comparison of the L-NTADE (SR) algorithm with $c = 4$ to other modern DE algorithms on the CEC 2017 benchmark, and Tables 16–18 on the CEC 2022 benchmark. Table 13 shows the results of Mann–Whitney statistical tests, Table 14 contains the Friedman ranking results, the ranking that is used in a Friedman statistical test to compare several sets of measurements; smaller total ranks are better. Table 15 shows the recently proposed U-scores [39]. Tables 16–18 contain the same results for CEC 2022. The U-scores represent a trial-based dominance method for comparing several optimization methods using Mann–Whitney U tests. For CEC 2017, the convergence speed is not taken into consideration, while for CEC 2022 the number of function evaluations to reach the desired accuracy is considered.

Table 13. L-NTADE (SR) vs. alternative approaches, Mann–Whitney tests, CEC 2017.

Algorithm	10D	30D	50D	100D
L-NTADE (SR, $c = 4$) vs. LSHADE-SPACMA [35]	9/18/3 (25.76)	17/6/7 (75.97)	14/3/13 (17.98)	14/1/15 (1.50)
L-NTADE (SR, $c = 4$) vs. jSO [25]	6/21/3 (7.27)	19/10/1 (139.25)	22/6/2 (153.40)	24/1/5 (147.83)
L-NTADE (SR, $c = 4$) vs. EBOwithCMAR [40]	3/19/8 (−37.99)	16/9/5 (79.00)	18/7/5 (110.30)	22/2/6 (131.61)
L-NTADE (SR, $c = 4$) vs. L-SHADE-RSP [9]	3/24/3 (2.69)	18/11/1 (127.15)	20/8/2 (131.90)	20/6/4 (121.52)
L-NTADE (SR, $c = 4$) vs. NL-SHADE-RSP [10]	12/9/9 (17.82)	23/4/3 (173.49)	29/1/0 (250.21)	29/0/1 (241.25)
L-NTADE (SR, $c = 4$) vs. NL-SHADE-LBC [11]	6/20/4 (6.58)	21/8/1 (176.68)	28/2/0 (227.05)	27/1/2 (210.08)
L-NTADE (SR, $c = 4$) vs. L-NTADE [12]	11/14/5 (37.30)	14/15/1 (75.39)	15/11/4 (69.26)	15/11/4 (70.63)

Table 14. L-NTADE (SR) vs. alternative approaches; Friedman ranking. Smaller ranks are better, CEC 2017.

Algorithm	10D	30D	50D	100D	Total
LSHADE-SPACMA [35]	139.42	127.42	100.55	85.06	452.45
jSO [25]	134.97	131.33	132.14	139.09	537.53
EBOwithCMAR [40]	**116.15**	123.19	124.94	131.67	495.94
L-SHADE-RSP [9]	133.93	126.37	121.91	122.36	504.58
NL-SHADE-RSP [10]	146.25	196.87	226.37	226.63	796.13
NL-SHADE-LBC [11]	133.80	177.80	193.38	191.89	696.88
L-NTADE [12]	143.55	110.41	100.85	101.93	456.75
L-NTADE (SR, $c = 4$)	131.92	86.60	79.85	81.37	379.75

As can be seen from Table 13, the L-NTADE algorithm with success rate-based adaptation is able to outperform most of the alternative algorithms in both benchmarks. EBOwithCMAR performed better in the 10D case, and LSHADE-SPACMA showed comparable in the 100D case. Table 14 supports these results, ranking L-NTADE with $c = 4$ the best

algorithm overall, and even the standard L-NTADE is very competitive. In Table 15, the comparison yields similar results, and L-NTADE with $c = 4$ ranks first, followed by LSHADE-SPACMA and L-NTADE.

Table 15. L-NTADE (SR) vs. alternative approaches, U-scores. Larger ranks are better [39], CEC 2017.

Algorithm	**10*D***	**30*D***	**-**
LSHADE-SPACMA [35]	263,193	293,839.5	-
jSO [25]	272,469	282,122.5	-
EBOwithCMAR [40]	322,146.5	303,458.5	-
L-SHADE-RSP [9]	276,570.5	296,012	-
NL-SHADE-RSP [10]	244,170	112,109	-
NL-SHADE-LBC [11]	276,146	161,327	-
L-NTADE [12]	248,186	336,712.5	-
L-NTADE (SR, $c = 4$)	281,959	399,259	-
Algorithm	50*D*	100*D*	Total
LSHADE-SPACMA [35]	364,005.5	404,653	1,325,691
jSO [25]	281,506	262,631	1,098,728.5
EBOwithCMAR [40]	298,859	282,227	1,206,691
L-SHADE-RSP [9]	305,628.5	305,817	1,184,028
NL-SHADE-RSP [10]	34,664.5	34,687	425,630.5
NL-SHADE-LBC [11]	121,800.5	124,111	683,384.5
L-NTADE [12]	362,005	359,221	1,306,124.5
L-NTADE (SR, $c = 4$)	416,371	411,493	1,509,082

Table 16. L-NTADE (SR) vs. alternative approaches, CEC 2022.

Algorithm	**10*D***	**20*D***
L-NTADE (SR, $c = 4$) vs. APGSK-IMODE [36]	8/2/2 (36.86)	10/0/2 (52.60)
L-NTADE (SR, $c = 4$) vs. MLS-LSHADE [37]	4/2/6 (−9.67)	4/1/7 (−19.78)
L-NTADE (SR, $c = 4$) vs. MadDE [38]	8/2/2 (39.05)	8/2/2 (43.60)
L-NTADE (SR, $c = 4$) vs. EA4eigN100 [41]	2/6/4 (−15.49)	5/3/4 (5.51)
L-NTADE (SR, $c = 4$) vs. NL-SHADE-RSP-MID [42]	3/4/5 (−5.98)	6/2/4 (17.59)
L-NTADE (SR, $c = 4$) vs. L-SHADE-RSP [9]	5/3/4 (14.23)	4/6/2 (12.24)
L-NTADE (SR, $c = 4$) vs. NL-SHADE-RSP [10]	6/3/3 (24.51)	7/4/1 (34.78)
L-NTADE (SR, $c = 4$) vs. NL-SHADE-LBC [11]	2/3/7 (−30.20)	4/4/4 (−3.75)
L-NTADE (SR, $c = 4$) vs. L-NTADE [12]	5/5/2 (26.17)	2/6/4 (−1.82)

The comparison on the CEC 2022 benchmark in Table 16 shows that L-NTADE with $c = 4$ is outperformed by NL-SHADE-LBC and EA4eigN100 in the 10D case, but in 20D the difference between them decreases. However, the MLS-LSHADE is able to deliver better performance in the 20D case. The Friedman ranking in Table 17 sets EA4eigN100 in first place, followed by NL-SHADE-LBC, and MLS-LSHADE and L-NTADE with $c = 4$. However, when using U-scores in Table 18, L-NTADE with $c = 4$ takes second place after NL-SHADE-LBC, and MLS-LSHADE has an almost identical total rank. We note that L-NTADE used the same parameter settings in both benchmarks, and still was able to show highly competitive results in both of them.

Table 17. L-NTADE (SR) vs. alternative approaches, Friedman ranking. Smaller ranks are better, CEC 2022.

Algorithm	10*D*	20*D*	Total
APGSK-IMODE [36]	81.57	87.92	169.48
MLS-LSHADE [37]	64.05	50.13	114.18
MadDE [38]	87.15	87.02	174.17
EA4eigN100 [41]	39.17	53.02	92.18
NL-SHADE-RSP-MID [42]	58.92	71.25	130.17
L-SHADE-RSP [9]	66.65	59.07	125.72
NL-SHADE-RSP [10]	88.23	83.58	171.82
NL-SHADE-LBC [11]	49.77	56.73	106.50
L-NTADE [12]	64.58	55.92	120.50
L-NTADE (SR, $c = 4$)	59.92	55.37	115.28

Table 18. L-NTADE (SR) vs. alternative approaches, U-scores. Larger ranks are better [39], CEC 2022.

Algorithm	10*D*	20*D*	Total
APGSK-IMODE [36]	38,228	28,381.5	66,609.5
MLS-LSHADE [37]	52,255	62,548.5	114,803.5
MadDE [38]	35,727	30,402.5	66,129.5
EA4eigN100 [41]	50358	60,202	110,560
NL-SHADE-RSP-MID [42]	56,740.5	44,291.5	101,032
L-SHADE-RSP [9]	44,754.5	54,543.5	99,298
NL-SHADE-RSP [10]	35,363.5	32,772.5	68,136
NL-SHADE-LBC [11]	64,330	56,998.5	121,328.5
L-NTADE [12]	51,346	57,894	109,240
L-NTADE (SR, $c = 4$)	56,897.5	57,965.5	114,863

In order to compare the computational complexity of the proposed approach, the experiment on the CEC 2022 benchmark was performed according to the rules of the CEC 2022 competition rules [14]. The T0, T1 and T2 values are the estimations of time required for the processor to perform mathematical evaluations, target function evaluation time and the algorithm runtime, respectively. Table 19 compares L-NTADE with the SHA and the SR adaptation methods.

Table 19. Computational complexity of L-NTADE with SHA and the SR adaptation methods.

L-NTADE				
D	$T0$	$T1$	$T2$	$(T2-T1)/T0$
$D=10$	2.5×10^{-2}	2.1×10^{-2}	1.358×10^{-1}	4.592
$D=20$	2.5×10^{-2}	5.0×10^{-2}	1.704×10^{-1}	4.816
L-NTADE (SR, $c=4$)				
D	$T0$	$T1$	$T2$	$(T2-T1)/T0$
$D=10$	2.5×10^{-2}	2.1×10^{-2}	1.192×10^{-1}	3.928
$D=20$	2.5×10^{-2}	5.0×10^{-2}	1.658×10^{-1}	4.632

As can be seen from Table 19, applying SR reduces the amount of time, as the calculation of the weighted Lehmer mean for F in success history-based adaptation requires significant computational effort.

5. Discussion

The main advantage of the proposed success rate-based adaptation of the scaling factor is its simplicity. As simple as the original DE, it relies on a naturally present value in the algorithm, namely the number of successful individuals, i.e., the number of solutions that were improved according to the selection step. This number is, in fact, calculated by the majority of existing DE algorithms. Success rate-based adaptation does not require complex calculations of means, averaging over time and other techniques, and still performs very well. Moreover, the experiments have shown that the sensitivity to the c parameter is very low: the same value, $c=4$, works on different benchmarks, different functions and different computational resource. This property makes for a universal approach, which seems to work well enough in most scenarios. Although there are cases when the success history adaptation may deliver better performance, we believe that this can be mitigated by further development of SR-based adaptation schemes.

As for the reasons for high performance, and why $c=4$ is a good choice in many cases, the following explanation seems reasonable. The SR in most tested algorithms worked together with the current-to-pbest mutation strategy, which has a specific structure. If $F>0.5$, then the new solutions are attracted closer to some of the best ones, although this also means that larger steps in other directions will be made. If $F<0.5$, this means that the attention towards better solutions is smaller, and the algorithm searches more around the current positions of each individual. What the success rate adaptation does is it switches between these two behaviors. If the search is going well, it makes DE generate solutions closer to better ones, and the F values are relatively stable. However, if the algorithm is stagnating, a wider search is beneficial. In this case, success rate adaptation makes MF oscillate in the $[0,0.5]$ range, sampling smaller F values and trying to escape local optima. In this manner, applying a simple curve like $SR^{1/4}$ produces the desired behavior of DE without complex adaptation mechanisms.

The experiments with other mutation strategies revealed that SR-based adaptation works not only with the current-to-pbest strategy, but also with other ones, and in all cases there was a setting of c which lead to improved performance. Moreover, in most cases, any c value resulted in better performance than SHA. In the case of the rand/1 and rand/2 strategies, this improvement was very significant, and even allowed for current-to-pbest to be outperformed in some cases. This might mean that the SR-based adaptation can be used as a universal mechanism, independent of the mutation strategy, although the c value should be set accordingly.

Replacing the original success history adaptation in other DE-based algorithms, such as APGSK-IMODE, MLS-LSHADE, MadDE, jSO and LSHADE-SPACMA, did not always improve performance; nevertheless, the results were mostly comparable. Some of the

modified algorithms represented hybrids with other methods, such as local search or covariance matrix adaptation, and probably better results could be achieved with a more careful tuning of the hybridization.

Of course, the presented experiments cannot cover all the possibilities of success rate-based adaptation. However, it is worth mentioning some of the possible directions of further studies, which include:

1. Developing a crossover rate adaptation scheme based on the success rate,
2. Connecting the success rate to the *pbest* parameter,
3. Combining the advantages of the success history and success rate adaptation.

6. Conclusions

In this study, a new adaptation technique for the scaling factor parameter in differential evolution was proposed. The new method relies on the success rate, and the performed experimental analysis showed that the new method is insensitive to computational resource, works with different mutation strategies, and can be included in various algorithms.

Author Contributions: Conceptualization, V.S. and E.S.; methodology, V.S. and E.S.; software, V.S.; validation, V.S. and E.S.; formal analysis, E.S.; investigation, V.S.; resources, E.S. and V.S.; data curation, E.S.; writing—original draft preparation, V.S.; writing—review and editing, V.S.; visualization, V.S.; supervision, E.S.; project administration, E.S.; funding acquisition, V.S. and E.S. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Ministry of Science and Higher Education of the Russian Federation within limits of state contract No. FEFE-2023-0004.

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

GA	Genetic Algorithms
GP	Genetic Programming
EC	Evolutionary Computation
DE	Differential Evolution
CEC	Congress on Evolutionary Computation
SHADE	Success History Adaptive Differential Evolution
LPSR	Linear Population Size Reduction
LBC	Linear Bias Change
RSP	Rank-based Selective Pressure
L-NTADE	Linear population size reduction Newest and Top Adaptive Differential Evolution

References

1. Price, K.; Storn, R.; Lampinen, J. *Differential Evolution: A Practical Approach to Global Optimization*; Springer: Berlin/Heidelberg, Germany, 2005.
2. Das, S.; Suganthan, P. Differential evolution: a survey of the state-of-the-art. *IEEE Trans. Evol. Comput.* **2011**, *15*, 4–31. [CrossRef]
3. Brest, J.; Greiner, S.; Boškovic, B.; Mernik, M.; Žumer, V. Self-adapting control parameters in differential evolution: a comparative study on numerical benchmark problems. *IEEE Trans. Evol. Comput.* **2006**, *10*, 646–657. [CrossRef]
4. Zhang, J.; Sanderson, A.C. JADE: Adaptive Differential Evolution With Optional External Archive. *IEEE Trans. Evol. Comput.* **2009**, *13*, 945–958. [CrossRef]
5. Tanabe, R.; Fukunaga, A. Success-history based parameter adaptation for differential evolution. In Proceedings of the 2013 IEEE Congress on Evolutionary Computation, Cancun, Mexico, 20–23 June 2013; pp. 71–78. [CrossRef]
6. Piotrowski, A.P.; Napiorkowski, J.J. Step-by-step improvement of JADE and SHADE-based algorithms: Success or failure? *Swarm Evol. Comput.* **2018**, *43*, 88–108. [CrossRef]
7. Stanovov, V.; Akhmedova, S.; Semenkin, E. Biased Parameter Adaptation in Differential Evolution. *Inf. Sci.* **2021**, *566*, 215–238. [CrossRef]

8. Stanovov, V.; Akhmedova, S.; Semenkin, E. The automatic design of parameter adaptation techniques for differential evolution with genetic programming. *Knowl. Based Syst.* **2022**, *239*, 108070. [CrossRef]
9. Stanovov, V.; Akhmedova, S.; Semenkin, E. LSHADE Algorithm with Rank-Based Selective Pressure Strategy for Solving CEC 2017 Benchmark Problems. In Proceedings of the 2018 IEEE Congress on Evolutionary Computation (CEC), Rio de Janeiro, Brazil, 8–13 July 2018; pp. 1–8.
10. Stanovov, V.; Akhmedova, S.; Semenkin, E. NL-SHADE-RSP Algorithm with Adaptive Archive and Selective Pressure for CEC 2021 Numerical Optimization. In Proceedings of the 2021 IEEE Congress on Evolutionary Computation (CEC), Kraków, Poland, 28 June–1 July 2021; pp. 809–816. [CrossRef]
11. Stanovov, V.; Akhmedova, S.; Semenkin, E. NL-SHADE-LBC algorithm with linear parameter adaptation bias change for CEC 2022 Numerical Optimization. In Proceedings of the 2022 IEEE Congress on Evolutionary Computation (CEC), Padua, Italy, 18–23 July 2022.
12. Stanovov, V.; Akhmedova, S.; Semenkin, E. Dual-Population Adaptive Differential Evolution Algorithm L-NTADE. *Mathematics* **2022**, *10*, 4666. [CrossRef]
13. Awad, N.; Ali, M.; Liang, J.; Qu, B.; Suganthan, P. *Problem Definitions and Evaluation Criteria for the CEC 2017 special Session and Competition on Single Objective Bound Constrained Real-Parameter Numerical Optimization*; Technical Report; Nanyang Technological University: Singapoure, 2016.
14. Kumar, A.; Price, K.; Mohamed, A.K.; Suganthan, P.N. *Problem Definitions and Evaluation Criteria for the CEC 2022 Special Session and Competition on Single Objective Bound Constrained Numerical Optimization*; Technical Report; Nanyang Technological University: Singapoure, 2021.
15. Storn, R.; Price, K. Differential evolution—A simple and efficient heuristic for global optimization over continuous spaces. *J. Glob. Optim.* **1997**, *11*, 341–359. [CrossRef]
16. Biedrzycki, R.; Arabas, J.; Jagodziński, D. Bound constraints handling in Differential Evolution: An experimental study. *Swarm Evol. Comput.* **2019**, *50*, 100453. [CrossRef]
17. Kumar, A.; Biswas, P.P.; Suganthan, P.N. Differential evolution with orthogonal array-based initialization and a novel selection strategy. *Swarm Evol. Comput.* **2022**, *68*, 101010. [CrossRef]
18. Das, S.; Mullick, S.; Suganthan, P. Recent advances in differential evolution—An updated survey. *Swarm Evol. Comput.* **2016**, *27*, 1–30. [CrossRef]
19. Al-Dabbagh, R.D.; Neri, F.; Idris, N.; Baba, M.S.B. Algorithmic design issues in adaptive differential evolution schemes: Review and taxonomy. *Swarm Evol. Comput.* **2018**, *43*, 284–311. [CrossRef]
20. Brest, J.; Maucec, M.; Bovsković, B. The 100-Digit Challenge: Algorithm jDE100. In Proceedings of the 2019 IEEE Congress on Evolutionary Computation (CEC), Wellington, New Zealand, 10–13 June 2019; pp. 19–26.
21. Brest, J.; Maucec, M.; Bosković, B. Differential Evolution Algorithm for Single Objective Bound-Constrained Optimization: Algorithm j2020. In Proceedings of the 2020 IEEE Congress on Evolutionary Computation (CEC), Glasgow, UK, 19–24 July 2020; pp. 1–8.
22. Qin, A.K.; Suganthan, P.N. Self-adaptive differential evolution algorithm for numerical optimization. In Proceedings of the 2005 IEEE Congress on Evolutionary Computation, Edinburgh, UK, 2–5 September 2005; Volume 2; pp. 1785–1791 .
23. Bullen, P. *Handbook of Means and Their Inequalities*; Springer: Amsterdam, The Netherlands, 2003. [CrossRef]
24. Tanabe, R.; Fukunaga, A. Improving the search performance of SHADE using linear population size reduction. In Proceedings of the IEEE Congress on Evolutionary Computation, CEC, Beijing, China, 6–11 July 2014; pp. 1658–1665. [CrossRef]
25. Brest, J.; Maučec, M.; Boškovic, B. Single objective real-parameter optimization algorithm jSO. In Proceedings of the IEEE Congress on Evolutionary Computation, Donostia, Spain, 5–8 June 2017; pp. 1311–1318. [CrossRef]
26. Gong, W.; Cai, Z. Differential Evolution With Ranking-Based Mutation Operators. *IEEE Trans. Cybern.* **2013**, *43*, 2066–2081. [CrossRef] [PubMed]
27. Viktorin, A.; Senkerik, R.; Pluhacek, M.; Kadavy, T.; Zamuda, A. Distance based parameter adaptation for Success-History based Differential Evolution. *Swarm Evol. Comput.* **2019**, *50*, 100462. [CrossRef]
28. Bujok, P.; Kolenovsky, P. Differential Evolution with Distance-based Mutation-selection Applied to CEC 2021 Single Objective Numerical Optimisation. In Proceedings of the 2021 IEEE Congress on Evolutionary Computation (CEC), Kraków, Poland, 28 June–1 July 2021; pp. 849–856.
29. Yang, M.; Cai, Z.; Li, C.; Guan, J. An improved adaptive differential evolution algorithm with population adaptation. In Proceedings of the Annual Conference on Genetic and Evolutionary Computation, Amsterdam, The Netherlands, 6–10 July 2013.
30. Santucci, V.; Baioletti, M.; Bari, G.D. An improved memetic algebraic differential evolution for solving the multidimensional two-way number partitioning problem. *Expert Syst. Appl.* **2021**, *178*, 114938. [CrossRef]
31. Chen, X.; Shen, A. Self-adaptive differential evolution with Gaussian–Cauchy mutation for large-scale CHP economic dispatch problem. *Neural Comput. Appl.* **2022**, *34*, 11769–11787. [CrossRef]
32. Yi, W.; Chen, Y.; Pei, Z.; Lu, J. Adaptive differential evolution with ensembling operators for continuous optimization problems. *Swarm Evol. Comput.* **2021**, *69*, 100994. [CrossRef]
33. Yang, Q.; Qiao, Z.Y.; Xu, P.; Lin, X.; Gao, X.D.; Wang, Z.J.; Lu, Z.Y.; Jeon, S.W.; Zhang, J. Triple competitive differential evolution for global numerical optimization. *Swarm Evol. Comput.* **2024**, *84*, 101450. [CrossRef]

34. Kitamura, T.; Fukunaga, A. Differential Evolution with an Unbounded Population. In Proceedings of the 2022 IEEE Congress on Evolutionary Computation (CEC), Padua, Italy, 18–23 July 2022.
35. Mohamed, A.; Hadi, A.A.; Fattouh, A.; Jambi, K. LSHADE with semi-parameter adaptation hybrid with CMA-ES for solving CEC 2017 benchmark problems. In Proceedings of the 2017 IEEE Congress on Evolutionary Computation (CEC), Donostia, Spain, 5–8 June 2017; pp. 145–152.
36. Mohamed, A.W.; Hadi, A.A.; Agrawal, P.; Sallam, K.M.; Mohamed, A.K. Gaining-Sharing Knowledge Based Algorithm with Adaptive Parameters Hybrid with IMODE Algorithm for Solving CEC 2021 Benchmark Problems. In Proceedings of the 2021 IEEE Congress on Evolutionary Computation (CEC), Kraków, Poland, 28 June–1 July 2021; pp. 841–848.
37. Cuong, L.V.; Bao, N.N.; Binh, H.T.T. T*echnical Report: A Multi-Start Local Search Algorithm with L-SHADE for Single Objective Bound Constrained Optimization*; Technical Report; SoICT, Hanoi University of Science and Technology: Hanoi, Vietnam, 2021.
38. Biswas, S.; Saha, D.; De, S.; Cobb, A.D.; Das, S.; Jalaian, B. Improving Differential Evolution through Bayesian Hyperparameter Optimization. In Proceedings of the 2021 IEEE Congress on Evolutionary Computation (CEC), Kraków, Poland, 28 June–1 July 2021; pp. 832–840.
39. Price, K.V.; Kumar, A.; Suganthan, P. Trial-based dominance for comparing both the speed and accuracy of stochastic optimizers with standard non-parametric tests. *Swarm Evol. Comput.* **2023**, *78*, 101287. [CrossRef]
40. Kumar, A.; Misra, R.K.; Singh, D. Improving the local search capability of Effective Butterfly Optimizer using Covariance Matrix Adapted Retreat Phase. In Proceedings of the 2017 IEEE Congress on Evolutionary Computation (CEC), Donostia, Spain, 5–8 June 2017; pp. 1835–1842.
41. Bujok, P.; Kolenovsky, P. Eigen Crossover in Cooperative Model of Evolutionary Algorithms Applied to CEC 2022 Single Objective Numerical Optimisation. In Proceedings of the 2022 IEEE Congress on Evolutionary Computation (CEC), Padua, Italy, 18–23 July 2022.
42. Biedrzycki, R.; Arabas, J.; Warchulski, E. A Version of NL-SHADE-RSP Algorithm with Midpoint for CEC 2022 Single Objective Bound Constrained Problems. In Proceedings of the 2022 IEEE Congress on Evolutionary Computation (CEC), Padua, Italy, 18–23 July 2022.