A Version of NL-SHADE-RSP Algorithm with Midpoint for CEC 2022 Single Objective Bound Constrained Problems

Rafał Biedrzycki
Warsaw University of Technology
Institute of Computer Science
Warsaw, Poland
rafal.biedrzycki@pw.edu.pl

Jarosław Arabas
Warsaw University of Technology
Institute of Computer Science
Warsaw, Poland
jaroslaw.arabas@pw.edu.pl

Eryk Warchulski
Warsaw University of Technology
Institute of Computer Science
Warsaw, Poland
eryk.warchulski@pw.edu.pl

***Abstract*—This paper presents an enhanced version of NL-SHADE-RSP, which won CEC'2021 competition on single objective bound-constrained numerical optimization for shifted and rotated shifted functions. The proposed version uses the midpoint of the population to estimate the optimum. The midpoint fitness is also used to introduce a restart trigger. For large populations, the midpoint is calculated after splitting the population into two parts by the k-means algorithm. Other introduced modifications include changing the bound constrain handling method and reducing population size. The performance of the proposed approach is evaluated on the CEC 2022 benchmark for single objective bound-constrained numerical optimization. The results confirm that each proposed modification gradually improves the algorithm's ranking on the benchmark.**

***Index Terms*—differential evolution, CEC 2022, midpoint, resampling**

I. Introduction

Differential Evolution (DE) [1], [2] is a simple yet powerful concept that created the whole algorithms family. Step-by-step developed improvement of the family is a success story that led to very competitive algorithms [3]. One example is NL-SHADE-RSP [4] which won the CEC'2021 competition on single objective bound-constrained numerical optimization [5] for affine transformed, i.e., shifted or rotated, functions.

The aim of this paper is to enhance the winning algorithm from the CEC'2021 competition. To enable it, several components have been added to NL-SHADE-RSP to improve its ranking on the CEC'2022 benchmark suite [6]. The first component calculates and evaluates the midpoint of the population, which better estimates the optimum than the best-so-far solution [7]. The second introduces restart triggers, whilst the third changes the bound constraint handling method. Finally, the fourth component uses the machine learning method to split the population into two more strongly related halves to increase the chances of finding a better midpoint.

The empirical experiments validate the influence of each component with three methods of comparison being used: the ranking defined for CEC'2022, Wilcoxon signed-rank test, and empirical cumulative distribution functions (ECDFs) [8]. According to the results, proposed modifications improved the base method, especially for 20-dimensional problems.

This paper is organized as follows. In Section II a short description of DE and NL-SHADE-RSP is provided together with a short survey of restart methods and k-means clustering applications in the DE family. Section III presents all modifications that were introduced to the NL-SHADE-RSP. The results of experiments that investigate the impact of the introduced changes are given in Section IV. Section V concludes the paper.

II. Related work

This section presents a short survey of related works. After a brief description of the NL-SHADE-RSP, which is a method extended in this paper, there are subsections related to each component used in the proposed method.

A. From DE to NL-SHADE-RSP

Since the first publications [1], [2] Differential Evolution (DE) has been paid a growing attention thanks to the efficiency of the method and simplicity of the basic idea. DE maintains population of individuals. In the simplest version, called DE/rand/1/bin, the i-th mutant is a combination of three randomly selected solutions, i.e.: $v_i = x_{r3} + F \cdot (x_{r1} - x_{r2})$, where the scale factor F (usually $F \in (0, 1]$) is a parameter of the algorithm. After mutation, a trial vector is generated by the binomial crossover: for each dimension, a value is copied from the mutant if a random number sampled from the standard uniform distribution is less than the crossover ratio (CR) value. Otherwise, it is copied from the i-th individual. The value of CR is the algorithm parameter.

Numerous modifications of this scheme have been proposed — see [9] for an overview — with the L-SHADE [10] being one example. The L-SHADE adapts all parameters necessary for DE, i.e., population size (μ), CR, and F. The value of μ is linearly reduced from the maximal to the minimal value during the search. Values of CR and F are generated randomly before every mutation and crossover operation, taking into account the most successful values that have been observed in the search history. L-SHADE also uses an archive of inferior solutions to guide mutation. The idea of L-SHADE was extended by L-SHADE-RSP [11], which added a concept of rank-based selective pressure (RSP), where probabilities of selecting x_{r1} and x_{r2} are dependent on fitness-based ranks.

The NL-SHADE-RSP [4] is an extension of L-SHADE-RSP. One of the differences was exposed in the algorithm's name, i.e., it uses non-linear population size reduction (NL). In this idea, a new population size is calculated as follows: $\mu_{t+1} = \lfloor(\mu_{min} - \mu_{max})\, s^{1-s} + \mu_{max}\rfloor$, where μ_{min} and μ_{max} are minimal and maximal population size, and s is a ratio of the current number of objective function evaluations to the maximal budget. The rank-based selection was also changed to apply only to x_{r2}. There are also some minor changes, like the introduction of the automatic tuning of archive usage probability.

Unfortunately, due to the page limits, a more detailed description of the NL-SHADE-RSP and its predecessors cannot be provided here.

B. Restarts in DE

The idea of restarts is frequently used within the EC community. It is used to improve a CMA-ES [12], [13], but is also known in the DE family. In [14] a restart strategy was introduced into DE, SHADE, and L-SHADE. In modified DE and SHADE, a restart is performed when: 1) solution vector converged, i.e., changes in coordinates across the population are below threshold; 2) fitness values in the population converged; 3) there was no update of the best-so-far solution during $500D$ fitness evaluations, where D is the problem dimensionality. After the restart is triggered, the algorithm is started from scratch, i.e., all data structures are cleaned. In the L-SHADE, only the third restart trigger was used.

In [15] a restart differential evolution algorithm (RDEL) is proposed. What is important here is that the whole algorithm is not restarted; only random coordinates of individuals that stagnated are either set to a random value or undergo an additional mutation. The individual is considered stagnated when its fitness has not changed by more than 10^{-6} every 25 iterations.

In [16] a variant of DE is proposed which uses random reinitialization of 20% of the individuals, which is called restart. The restart is triggered every 300 iterations. The best individual is excluded from reinitialization.

In [17] the JADE algorithm is extended, with one of the extensions being a restart mechanism. In that mechanism, the stagnation is detected by analysis of the so-called "variable coefficient", defined as the quotient of fitness standard deviation and fitness mean. If this coefficient is below the specified threshold, a restart is triggered. As a result, trial vectors are generated by a special mutation. For each dimension, a value from a normal distribution whose standard deviation depends on the distance from bounds to the best solution x_b is added to x_b. Additionally, the crossover ratio (CR) is reduced to 0.1.

C. K-means clustering in DE

K-means clustering method [18] plays an important role in data mining and knowledge discovery. It separates given observations into k clusters in a way to minimize within-cluster variances.

In [19] an island model was used with DE. A population was split into subsets called islands by the k-means clustering, where k is a parameter of the algorithm.

In [20] a clustering-based differential evolution algorithm was proposed. In the algorithm, every ten iterations, a population is split into clusters by one step of the k-means algorithm. It allows for the application of different mutation strategies for each cluster. The number of clusters is randomly drawn from 2 to $\sqrt{\mu}$, where μ is the population size.

In [21] the parent selection operator uses k-means to select parents from different groups. This procedure is only applied to 5% of the mutations.

D. Midpoint

In [7] it was shown theoretically and experimentally for DE, JADE, SADE, and other methods, that a midpoint of the population better estimates the optimum than a commonly used best-so-far solution. This knowledge was used to introduce the RB-IPOP [13] which is a version of IPOP-CMA-ES [12]. In the RB-IPOP, the midpoint updates the best-so-far solution and is used to define an additional restart trigger. The trigger fires when the fitness of the midpoint does not change for a predefined number of iterations.

III. Modifications of NL-SHADE-RSP

In order to further improve NL-SHADE-RSP results, four new components have been added to the original algorithm. The positive influence of each of them was verified using ranking defined by organizers of the CEC'2022 competition on single objective bound-constrained numerical optimization [6].

The first added component calculates the midpoint of the population at each iteration. The objective function of the midpoint is used to update the best-so-far result. In addition, the objective function of each midpoint is compared with the objective of the nearest to midpoint individual in the population, with the better of these two being retained.

The second new component is responsible for stagnation detection and triggering restart. One such trigger checks the distance between a new midpoint and a midpoint nine iterations older. When the distance is below 10^{-9}, a restart is triggered. Another trigger stems from knowing that the optimum does not lie on the bounds of the search space. Each position $i = 1 \ldots \mu$ in the population has an assigned counter c_i. For each iteration, it is verified whether the individual x_i lies on bound in at least one dimension. If so, the counter c_i is incremented. Otherwise, it is set to 0. When any of the counters reaches ten, a restart is triggered.

The restarted algorithm loses all gained knowledge. It starts from a randomly initialized population. It was observed that NL-SHADE-RSP requires large populations for complex problems. Therefore, when it is restarted, the initial population size is set to 400. As in the original algorithm, the population size is reduced in the function of the utilized budget. The function responsible for calculating new population size is modified to decrease faster. For this task, the homographic function was used:

$$\mu_{t+1} = \lfloor a_1 + 1/(b * 10^{-6} + 2.57 * a_2) \rfloor \quad (1)$$

where b is a number of evaluations used in the current restart, and a_1, a_2 are the parameters set in a way to achieve 400 at the beginning and 20 at the end of the current b range. The comparison of curves used for population size reduction is presented in Fig. 1, assuming budget left for restart is 10^5.

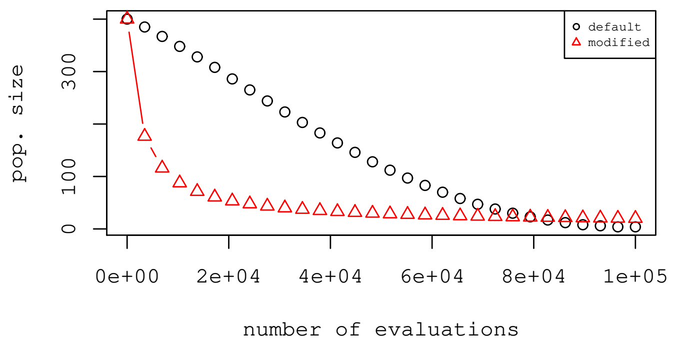

Fig. 1. The comparison of curves used for population size reduction

The third modification of the algorithm changes the bound constraint handling method from reinitialization to resampling. In resampling, the mutation is repeated when a mutant is out of bounds. According to [22] resampling was the best technique for DE-based methods, while reinitialization achieved a middle raking of 17 considered methods. In the version used here, when an out-of-bounds solution is generated, the differential mutation is repeated up to 10 unsuccessful trails without changing F and CR parameters. After that, the parameters are recalculated according to the rules defined by the NL-SHADE-RSP. When a total number of mutation repeats for a mutant achieves 100, the algorithm falls back to the default repair mechanism. It never happened during the experiments.

The fourth modification stems from the observation that the population in the algorithm is well spread, i.e., it does not occupy a single optimum for a long time. For the sake of making more use of the concept of the midpoint, the population is grouped into two groups by k-means algorithm [18]. The individuals inside each group are closer to one another than individuals without grouping. Therefore, it is more likely that they occupy the same attraction basin, which increases the chances of finding a good midpoint. Grouping into more clusters does not improve results, but it slows down the algorithm. In order to save evaluations of the objective functions, only midpoints resulting from grouping with quality above the threshold are evaluated. The assessment is done by the silhouette score [23] which values range from -1 (the worst) to 1 (the best). The k-means is not used when population size drops below a specified threshold.

IV. Results of the experiments

The implementation of NL-SHADE-RSP was downloaded from [24]. All parameters were left untouched except for the population size, which was set to $5D$, where D is the problem dimensionality. All introduced components were sequentially added on top of accepted components. The modified version described in the paper is available from [25].

The experimental setup strictly follows the CEC'2022 competition rules [6]. The budget was set to $2 \cdot 10^5$ for 10 D, and 10^6 for 20 D. For each function in each dimensionality, 30 independent runs were performed. All objective function error values smaller than 10^{-8} were set to 10^{-8}. The seeds of the Mersenne Twister pseudo-random generator were set to those provided by the competition organizers. The initial population is initialized with randomly generated individuals distributed uniformly in the admissible area (± 100).

The ranking defined by the competition organizers is used here as one of the methods of algorithms comparison. The ranking not only rewards the objective function value achieved at the end of the search, but it also rewards faster algorithms when the optimum is found. Using the comparison rule mentioned above, all trials are ranked from the worst to the best for each function. The score of each algorithm is a sum of its ranks (the more, the better). When only two methods are compared, the score reduces to the Mann-Whitney U-statistic.

A. *Parameters setup*

The implementations of the majority of modern optimization methods are full of constants. Some are initial values of parameters that undergo adaptation, and some are thresholds used to make specific decisions. Usually, these constants are not disclosed in the articles, making it impossible to reimplement the method based only on the article.

The parameters of the original method undergo auto adaptation. All introduced modifications do not change the core of the algorithm. Therefore, fine-tuning of the parameters is not necessary.

This section describes how parameters were set up, whereas their meaning was described in Section III. Generally, after the initial guess, the parameter value was changed in the vicinity to detect the promising direction of changes. If the changes in the algorithm performance were small or (roughly) optimal value of the parameter was found, parameter setup was finished.

To save computational time, newly introduced parameters were set up using a 20-dimensional version of the benchmark. The ranking defined for the CEC'2022 was used to decide which parameter value is the best.

The midpoint component does not introduce any additional parameters. The restart component required setting up restart triggers, and the shape of the function used to decrease population size. The maximal number of iterations on bounds for an individual was set to 10 after examining values 8, 9, 10, and 11. The trigger based on the midpoint location requires two parameters, i.e., the difference threshold and the maximal number of iterations. These parameters are dependent, but the maximal number of iterations is more sensitive. Therefore, the threshold was set to 10^{-9}, and values 7, 8, 9, 10 were used as candidates for the maximal number of iterations. Nine was the best. After that, the default method of changing population size was compared with the proposed one. Three alternative multipliers of b from (1), i.e., $0.5 * 10^{-6}$, 10^{-6}, $2 * 10^{-6}$ were tested. The proposed method with a multiplier 10^{-6} was the best.

The resampling component has two parameters: the maximal number of mutation repeats, which was initially set to 100, and the number of repair trials that use old F and CR values, which was set to 10. There is no need to tune these parameters. The maximal number of mutation repeats is only for protection from an infinite loop. The limit was never hit during the experiments. The maximum number of repair trials using old F and CR values should be above 1 and below 100. When it is 100, the repair mechanism sometimes fails when randomly drawn F is 1. When it is 0, the results are slightly worse.

The k-means component introduces the threshold that separates good groupings from weak ones. Values $1/\left(4\sqrt{D}\right)$, $1/\left(3\sqrt{D}\right)$, $1/\left(5\sqrt{D}\right)$ were examined; $1/\left(4\sqrt{D}\right)$ was the best. The k-means is not used when population size drops below a threshold. Three threshold values were examined: 4, 10, and 20. Based on the results of the experiments, the value 20 will be used as the minimum population size that enables clustering.

B. Validating influence of added components

The influence of each added component was verified by CEC'2022 ranking for all functions in 10D in Table I, for 20D in Table II and the sum of the scores for 10D and 20D is provided in Table III.

TABLE I
CEC'2022 RANKING FOR EACH SUBSEQUENT MODIFICATION OF NL-SHADE-RSP (D = 10). THE BEST RESULTS FOR EACH ROW WERE TYPESET IN BOLD

	nl-shade-rsp	midpoint	restart	resampling	k-means
F1	1055	1640	1933	1990	**2384**
F2	1534	**1963**	1989	1819	1890
F3	11	1981	1850	1691	**3329**
F4	1105	1627	1966	1862	**2441**
F5	**1864**	1752	1748	1792	1845
F6	1524	**1964**	1895	1893	1724
F7	**1998**	1729	1463	1934	1877
F8	1893	**2050**	1694	1627	1736
F9	**1846**	1770	1845	1770	1770
F10	**2238**	1666	1728	1816	1553
F11	1617	1979	1662	1678	**2064**
F12	1782	1862	**1878**	1821	1658
Sum	18465	21982	21594	21692	**24269**

All components were added sequentially upon previously added ones. Therefore, each column shows the result of adding a component whose name is in the column's title on the top of all components listed on the left in the table. For instance, column "restart" shows results of NL-SHADE-RSP with midpoint and restart components, whereas column "k-means" shows the results of all introduced components.

TABLE II
CEC'2022 RANKING FOR EACH SUBSEQUENT MODIFICATION OF NL-SHADE-RSP (D = 20). THE BEST RESULTS FOR EACH ROW WERE TYPESET IN BOLD

	nl-shade-rsp	midpoint	restart	resampling	k-means
F1	1115	1678	1840	1827	**2540**
F2	682	562	2696	2262	**2799**
F3	2	1790	1850	1758	**3600**
F4	285	693	2449	2766	**2808**
F5	704	1141	**2478**	2399	2278
F6	**2153**	1876	1626	1450	1895
F7	1650	1604	1932	1837	**1977**
F8	1781	1593	1364	1731	**2531**
F9	1800	1800	1800	1800	1800
F10	1254	1490	1595	2108	**2554**
F11	1302	1209	2075	2163	**2251**
F12	**1946**	1656	1600	1901	1898
Sum	14673	17091	23304	24539	**28394**

TABLE III
CEC'2022 RANKING FOR EACH SUBSEQUENT MODIFICATION OF NL-SHADE-RSP (D=10 + D=20). THE BEST RESULT IS TYPESET IN BOLD

	nl-shade-rsp	midpoint	restart	resampling	k-means
Sum	33138	39073	44897	46231	**52663**

As could be seen for the sum of the scores for 10D (Table I), for 20D (Table II) and overall for the whole benchmark (Table III) each sequentially added component improves upon a previous step.

When analyzing results for each function in 10D, it can be observed that the original algorithm is the best for functions 5, 7, 9, and 10. Function 9 is challenging; most considered methods could only find the same local optimum, which lies on the bounds. Theoretically, restarts should help here, but they only were able not to degrade the results of the original method. The midpoint component made the algorithm the best version for functions 2, 6, and 8. Enabling restarts made the algorithm the best for function 12. Resampling was never the best; k-means were best for functions: 1, 3, 4, 11. These functions, except for F4, have in common that they were solved to the optimum in more than half of the runs. All these functions have in common that they do not have plateaus, and they resemble noisy functions with visible global shapes.

Even though k-means was not consistently advantageous for all functions, its usage gave an advantage over the plain version of the algorithm of about 6000 ranking points.

The differences between examined versions are easier to notice in 20D. The original version was the best only on functions 6 and 12 (it did not achieve similar performance for these functions in 10D). All versions failed on F9. The restart enormously helped for function 5. K-means was the best for the rest of the functions. In 20D, k-means gathered two times more points than the base algorithm.

C. Examining the population size

According to the result shown in the previous section, enabling all additional components, including k-means, results in a better method according to ranks. This section examines the influence of population size on the performance of the k-means version. As differences between variants of the algorithm were bigger in 20D, this dimensionality was used here. The results in a form of ranking for initial population size set to 40, 100, 160, 200, 400, 600 are presented in Table IV. It can

TABLE IV
THE INFLUENCE OF THE INITIAL POPULATION SIZE ON THE RESULTS OF THE PROPOSED METHOD (D = 20). THE BEST RESULTS FOR EACH ROW WERE TYPESET IN BOLD

	40	100	160	200	400	600
F1	**4500**	3600	2700	1800	900	0
F2	**2780**	2362	1999	2493	2071	1796
F3	**4500**	3600	2700	1800	900	0
F4	**2807**	2383	2252	2282	2284	1494
F5	2644	2135	1279	1507	2351	**3584**
F6	306	2406	2471	2412	**3419**	2486
F7	1148	2545	2230	2207	**2820**	2551
F8	1328	2284	2244	2460	2576	**2609**
F9	2250	2250	2250	2250	2250	2250
F10	1068	**4032**	3420	2580	1650	750
F11	**3208**	2627	2042	2436	1722	1467
F12	1259	1528	1950	2531	3113	**3119**
Sum	27797	**31751**	27535	26757	26056	22105

be observed that the sum of the scores was the best for the initial population size equal to 100, the second was 40, and the third was 160. Finally, the population with 600 individuals was the worst.

A somewhat different picture emerges when winners for each function are analyzed. The population of 600 individuals was the best for functions 5, 8, and 12, but it completely failed for functions 1 and 3. Size 400 was the best for functions 6 and 7; size 100 was the best only for function 10, and the smallest considered population size was the best for the rest of the functions. All population sizes failed for function number 9.

To understand better what is happening, Fig. 2 presents the results of the algorithm using the smallest, the largest, and optimal population size. To visualize the results, the plots of Empirical Cumulative Distribution Functions (ECDF) are used. The ECDF curve describes the fraction of target fitness values that are achieved in consecutive iterations by the best-so-far solution observed in that run. The ECDF curves can be averaged over many independent runs of the algorithm and many different optimization problems. Thus, they are a convenient tool for results aggregation and presentation.

It can be observed that when the initial population contained 100 individuals, it allowed for solving the largest number of problems at the end of the optimization. The largest initial population gave the lowest number of solved problems for most of the search. On the contrary, the smallest population gave the largest number of solved problems for most of the search, but it was outperformed at the finish of the search. Further analysis showed that good results of the small population are caused by faster solving of the functions that are solved to the optimum.

The version of NL-SHADE with all components, which uses an initial population of 100 individuals, will be named NL-SHADE-RSP-MID in the rest of the paper.

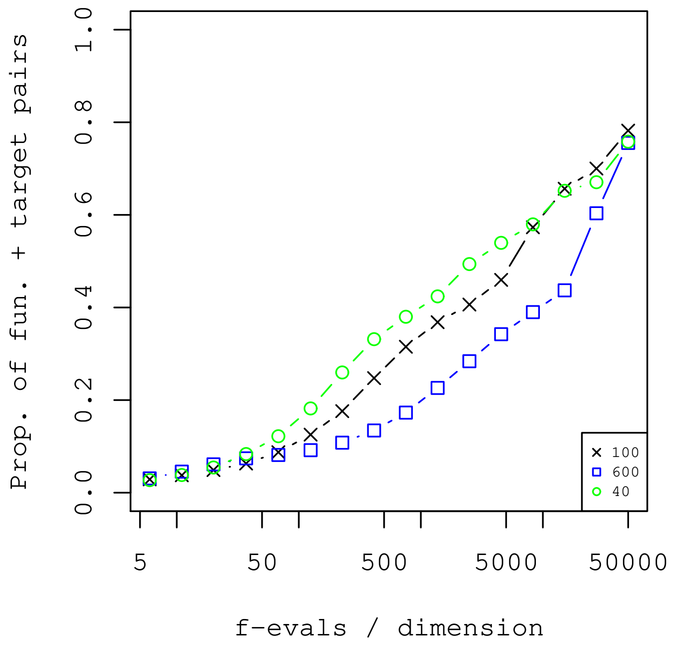

Fig. 2. The influence of population size on the results of the proposed method (D = 20)

D. *Verification by other metrics*

According to the outcome of the official ranking, NL-SHADE-RSP-MID is better than plain NL-SHADE-RSP. In this section, other methods are used to verify that. Table V contains means and standard deviations, and the outcome of the Wilcoxon signed-rank test calculated for each of the functions in 10D.

According to the results, the proposed version is better on function 4 and worse on function 10. No other statistically significant differences were detected. It can be observed that ignoring information about when the optimum is found makes it harder to distinguish competitors. It makes it impossible to differentiate methods that nearly always find the optimum (like algorithms under comparison on functions 1, 2, 3, 7, and 11).

In 20D (Table VI), functions 1 and 3 are solved by both methods, so algorithms are not distinguishable on these functions. Both algorithms failed on F9. The proposed method was better on 5 functions, and it was not distinguishable on the rest of them. It looks like the advantage of the proposed method grows with dimensionality.

The last type of comparison used ECDF curves. The curves for 10D are presented on Fig. 3 and for 20D are presented on Fig. 4.

The plots additionally include the result of the CMA-ES [26], [27], and DES [28] as a reference. DES (Differential Evolution Strategy) is an evolutionary algorithm that combines DE with the ES's internal dynamics. As shown in [28], its evolution of expectation vectors and the covariance matrices resembles the one observed in the CMA-ES without involving any matrix operations. At the CEC'2017 competition,

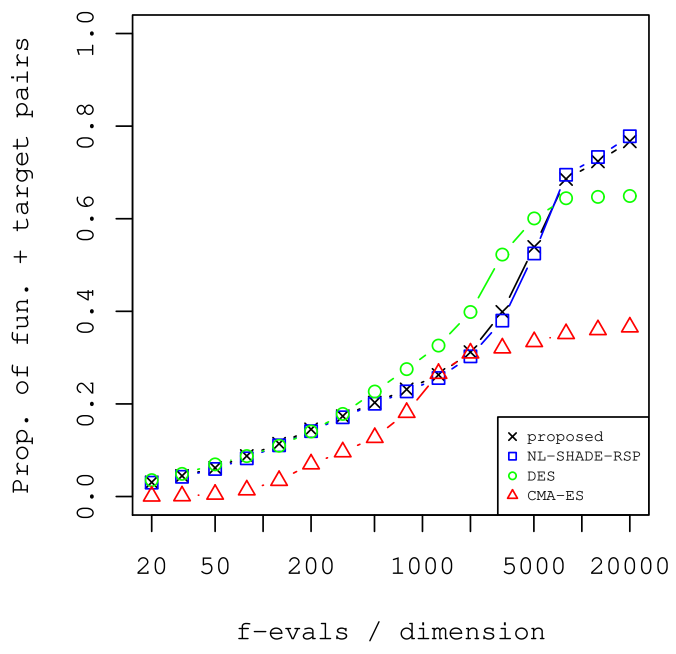

Fig. 3. Comparison of the proposed method with NL-SHADE-RSP, CMA-ES, DES in 10D.

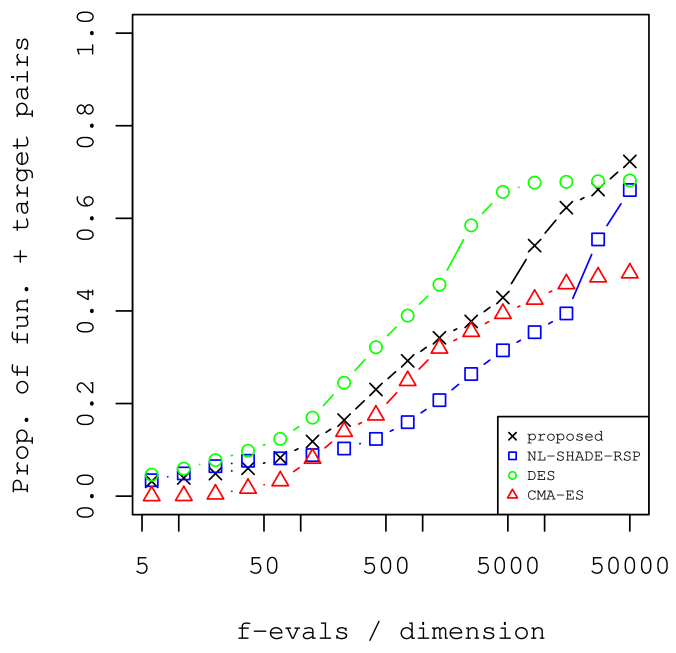

Fig. 4. Comparison of the proposed method with NL-SHADE-RSP, CMA-ES, DES in 20D

TABLE V
AVERAGE FITNESS, STANDARD DEVIATIONS AND WILCOXON SIGNED-RANK TEST OUTCOMES IN 10D (REFERENCE=NL-SHADE-RSP)

	NL-SHADE-RSP		Proposed version		
F.	Mean	Std	Mean	Std	T
1	1e−08	0e+00	1e−08	0e+00	=
2	1e−08	0e+00	1e−08	0e+00	=
3	1e−08	0e+00	1e−08	0e+00	=
4	1.6e+01	5.6e+00	1e+01	4.5e+00	+
5	1.7e+00	2.9e+00	1.7e+00	3.9e+00	=
6	1.8e−01	2.5e−01	1.7e−01	2.4e−01	=
7	3.7e−06	2e−05	1e−08	0e+00	=
8	2.2e−01	3e−01	2.4e−01	2.8e−01	=
9	2.2e+02	4.2e+01	2.3e+02	0e+00	=
10	1.3e−01	6.1e−01	4.5e+00	1.8e+01	−
11	1e−08	0e+00	1e−08	6.5e−10	=
12	1.6e+02	8.6e−01	1.6e+02	9.7e−01	=

TABLE VI
AVERAGE FITNESS, STANDARD DEVIATIONS AND WILCOXON SIGNED-RANK TEST OUTCOMES IN 20D (REFERENCE=NL-SHADE-RSP)

	NL-SHADE-RSP		Proposed version		
F.	Mean	Std	Mean	Std	T
1	1e−08	0e+00	1e−08	0e+00	=
2	4.4e+01	1.2e+01	8.9e+00	1.7e+01	+
3	1e−08	0e+00	1e−08	0e+00	=
4	8.9e+01	1.9e+01	2.8e+01	7.8e+00	+
5	4e+02	3.2e+02	1.5e+02	7.6e+01	+
6	5.6e+00	5.6e+00	6.3e+00	5.4e+00	=
7	1.4e+01	8.2e+00	1.2e+01	9e+00	=
8	2e+01	9.3e−01	2e+01	1.2e+00	+
9	1.8e+02	0e+00	1.8e+02	0e+00	=
10	1e−03	5.7e−03	2.1e−03	7.9e−03	=
11	2.8e+02	7.6e+01	1.5e+02	1.5e+02	+
12	2.4e+02	4.4e+00	2.4e+02	4e+00	=

DES achieved the best results for multidimensional problems amongst all participants.

The differences between NL-SHADE-RSP and its modification are hardly visible in 10D. Both methods are better than CMA-ES and DES at the finish of the search. DES is the best for budgets visible in the middle of the plot.

For 20D proposed method is better than its predecessor for most of the search. It is the best at the end of the search. For small budgets, DES is the best.

E. CEC tables

According to the rules of CEC'2022, the results at the end of the search should also be presented in the tables containing best, worst, median, mean and standard deviation for each function. The results for 10D are available in Table VII. The results for 20D are presented in Table VIII.

If we assume that function is solved when the median is at the optimum, the proposed method solves functions 1, 2, 3, 7, 10, 11 in 10D and functions 1, 3, 10 in 20D. For function 9, the standard deviation is 0, which means that consistently the same local optimum is found in all runs.

TABLE VII
BEST, WORST, MEDIAN, AVERAGE, AND STANDARD DEVIATION FOR 30 INDEPENDENT RUNS ON CEC'2022 PROBLEMS FOR 10D. THE BEST POSSIBLE VALUES WERE TYPESET IN BOLD

F.	Best	Worst	Median	Mean	Std
1	**1.00e−08**	**1.00e−08**	**1.00e−08**	**1.00e−08**	0.00e+00
2	**1.00e−08**	**1.00e−08**	**1.00e−08**	**1.00e−08**	0.00e+00
3	**1.00e−08**	**1.00e−08**	**1.00e−08**	**1.00e−08**	0.00e+00
4	2.98e+00	2.09e+01	9.45e+00	1.00e+01	4.55e+00
5	**1.00e−08**	1.66e+01	3.17e−01	1.69e+00	3.88e+00
6	1.78e−02	1.04e+00	9.65e−02	1.67e−01	2.45e−01
7	**1.00e−08**	**1.00e−08**	**1.00e−08**	**1.00e−08**	0.00e+00
8	9.39e−05	6.58e−01	8.09e−02	2.38e−01	2.78e−01
9	2.29e+02	2.29e+02	2.29e+02	2.29e+02	0.00e+00
10	**1.00e−08**	1.00e+02	**1.00e−08**	4.53e+00	1.83e+01
11	**1.00e−08**	1.36e−08	**1.00e−08**	1.01e−08	6.50e−10
12	1.63e+02	1.66e+02	1.65e+02	1.65e+02	9.72e−01

TABLE VIII
BEST, WORST, MEDIAN, AVERAGE, AND STANDARD DEVIATION FOR 30 INDEPENDENT RUNS ON CEC 2022 PROBLEMS FOR 20D. THE BEST POSSIBLE VALUES WERE TYPESET IN BOLD

F.	Best	Worst	Median	Mean	Std
1	**1.00e−08**	**1.00e−08**	**1.00e−08**	**1.00e−08**	0.00e+00
2	3.65e−05	4.91e+01	3.50e−01	8.93e+00	1.67e+01
3	**1.00e−08**	**1.00e−08**	**1.00e−08**	**1.00e−08**	0.00e+00
4	1.59e+01	5.77e+01	2.69e+01	2.79e+01	7.83e+00
5	1.48e+01	3.40e+02	1.39e+02	1.47e+02	7.61e+01
6	3.81e−01	2.18e+01	4.42e+00	6.35e+00	5.41e+00
7	9.95e−01	2.23e+01	7.16e+00	1.16e+01	8.99e+00
8	1.39e+01	2.05e+01	2.03e+01	2.00e+01	1.18e+00
9	1.81e+02	1.81e+02	1.81e+02	1.81e+02	0.00e+00
10	**1.00e−08**	3.12e−02	**1.00e−08**	2.08e−03	7.92e−03
11	**1.00e−08**	3.00e+02	1.50e+02	1.50e+02	1.53e+02
12	2.34e+02	2.52e+02	2.43e+02	2.43e+02	3.99e+00

F. Algorithm complexity

The experiments were performed on a laptop equipped with an Intel Core i7-6500U @ 2.5GHz processor that was working under the control of Ubuntu 20.04. The single threaded program was written in the C++ language and compiled by gcc with O2 flag. The rules of CEC 2022 [6] require measurements of the runtime of several functions to allow for the comparison of the complexity of the algorithms. More specifically, $T0$ is the execution time of $2 \cdot 10^5$ evaluations of some basic mathematical expressions, $T1$ is the time of $2 \cdot 10^5$ evaluations of function 1, and $\widehat{T2}$ is the mean taken from 5 independent runs of the presented algorithm. Each algorithm run was performed on a budget of $2 \cdot 10^5$ evaluations of function 1. Target function value was set lower than a global optimum to prevent the algorithm from stopping too early. In 10D several restarts were triggered by stagnation detection mechanisms.

The results of the experiments are presented in Table IX.

It can be observed that the algorithm's time complexity grows linearly with problem dimensionality. A two-fold increase in dimensionality increases more than twice the time required by objective function evaluations, but it increases less than twofold the time required by the whole algorithm. It is due to the computational time required to prepare data structures.

It can also be observed that the k-means component strongly

TABLE IX
COMPUTATIONAL COMPLEXITY OF THE ALGORITHM AND ITS VERSION WITHOUT K-MEANS ON INTEL CORE I7-6500U @ 2.5GHZ PROCESSOR. THE PROGRAM WAS WRITTEN IN C++ LANGUAGE AS A SINGLE THREADED APPLICATION

Dim. & version	$T0$	$T1$	$\widehat{T2}$	$(\widehat{T2} - T1)/T0$
10, default	0.0042	0.027	0.66s	150
20, default	0.0042	0.078	0.91s	197
10, no k-means	0.0042	0.027	0.18s	36
20, no k-means	0.0042	0.078	0.26s	44

increases the algorithm's complexity. The speed can be easily gained by using alternative versions of k-means and alternative initialization methods. The code that converts data structures between NL-SHADE-RSP and k-means can also be corrected. We also have to bear in mind that thanks to using C++, $\widehat{T2}$ is still below 1s, which cannot be achieved in some languages the researchers frequently use. For example, $\widehat{T2}$ achieved by the NL-SHADE-RSP-MID is about 83 times smaller than T1 computed in R language with the 'cecs' package in 10D.

V. CONCLUSIONS

The enhanced version of NL-SHADE-RSP, called NL-SHADE-RSP-MID, was proposed in this paper. The NL-SHADE-RSP won CEC'2021 on single objective bound-constrained numerical optimization for two categories of functions: shifted and rotated shifted. The NL-SHADE-RSP-MID extends its predecessor by adding four components: 1) the concept of midpoint; 2) restarts; 3) resampling as bound constraint handling; 4) k-means grouping. The midpoint of the population is used to estimate the optimum. Midpoint's fitness is also used to introduce a restart trigger. For large enough populations, the midpoint is calculated after splitting the population into two parts by the k-means algorithm. The resampling repeats mutation when an unfeasible point is generated.

The performance of the proposed approach was evaluated on the CEC'2022 benchmark for single objective bound-constrained numerical optimization. According to the ranking defined by the benchmark, each introduced component improved the base algorithm in 10 and 20 dimensions. The resulting algorithm was also compared to its predecessor using ECDF curves and statistical tests performed on the results obtained at the end of the budget. For these comparison methods, significant changes were not detected in 10D but were visible in 20D. The results of the NL-SHADE-RSP-MID were also compared to the results of the CMA-ES and DES. DES was the best for a low budget, but NL-SHADE-RSP-MID won for a large budget on both dimensionalities.

REFERENCES

[1] R. Storn and K. Price, "Differential evolution – a simple and efficient adaptive scheme for global optimization over continuous spaces," TR-95-012, ICSI, Tech. Rep., 1995.
[2] ——, "Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, no. 4, pp. 341–359, Dec 1997.

[3] A. P. Piotrowski and J. J. Napiorkowski, “Step-by-step improvement of JADE and SHADE-based algorithms: Success or failure?” *Swarm and Evolutionary Computation*, vol. 43, pp. 88–108, 2018.

[4] V. Stanovov, S. Akhmedova, and E. Semenkin, “NL-SHADE-RSP algorithm with adaptive archive and selective pressure for CEC 2021 numerical optimization,” in *2021 IEEE Congress on Evolutionary Computation (CEC)*, 2021, pp. 809–816.

[5] A. W. Mohamed, A. A. Hadi, A. K. Mohamed, P. Agrawal, A. Kumar, and P. Suganthan, “Problem definitions and evaluation criteria for the CEC 2021 special session and competition on single objective bound constrained numerical optimization,” Nanyang Technol. Univ., Singapore, Tech. Rep., 2020.

[6] A. Kumar, K. V. Price, A. W. Mohamed, A. A. Hadi, and P. Suganthan, “Problem definitions and evaluation criteria for the CEC 2022 special session and competition on single objective bound constrained numerical optimization,” Nanyang Technol. Univ., Singapore, Tech. Rep., 2022.

[7] J. Arabas and R. Biedrzycki, “Improving evolutionary algorithms in a continuous domain by monitoring the population midpoint,” *IEEE Transactions on Evolutionary Computation*, vol. 21, no. 5, pp. 807–812, Oct 2017.

[8] N. Hansen, D. Brockhoff, O. Mersmann, T. Tusar, D. Tusar, O. A. ElHara, P. R. Sampaio, A. Atamna, K. Varelas, U. Batu, D. M. Nguyen, F. Matzner, and A. Auger. (2019) COmparing Continuous Optimizers: numbbo/COCO on Github. [Online]. Available: https://github.com/numbbo/coco

[9] S. Das, S. S. Mullick, and P. Suganthan, “Recent advances in differential evolution – an updated survey,” *Swarm and Evol. Comput.*, vol. 27, pp. 1 – 30, 2016.

[10] R. Tanabe and A. S. Fukunaga, “Improving the search performance of SHADE using linear population size reduction,” in *2014 IEEE Congress on Evolutionary Computation (CEC)*, 2014, pp. 1658–1665.

[11] V. Stanovov, S. Akhmedova, and E. Semenkin, “LSHADE algorithm with rank-based selective pressure strategy for solving CEC 2017 benchmark problems,” in *2018 IEEE Congress on Evolutionary Computation (CEC)*, 2018, pp. 1–8.

[12] A. Auger and N. Hansen, “A restart CMA evolution strategy with increasing population size,” in *2005 IEEE Congress on Evolutionary Computation*, vol. 2, Sept 2005, pp. 1769–1776 Vol. 2.

[13] R. Biedrzycki, “A version of IPOP-CMA-ES algorithm with midpoint for CEC 2017 single objective bound constrained problems,” in *2017 IEEE Congress on Evolutionary Computation (CEC)*, June 2017, pp. 1489–1494.

[14] R. Tanabe and A. Fukunaga, “Tuning differential evolution for cheap, medium, and expensive computational budgets,” in *2015 IEEE Congress on Evolutionary Computation (CEC)*, 2015, pp. 2018–2025.

[15] A. W. Mohamed, “RDEL: Restart differential evolution algorithm with local search mutation for global numerical optimization,” *Egyptian Informatics Journal*, vol. 15, no. 3, pp. 175–188, 2014.

[16] J. Wetweerapong and P. Puphasuk, “An adaptive differential evolution algorithm with restart for solving continuous optimization problems,” *WSEAS Transactions on Systems and Control*, vol. 15, pp. 254–269, 06 2020.

[17] Y.-X. Zhang and J. Gou, “Adaptive differential evolution algorithm based on restart mechanism and direction information,” *IEEE Access*, vol. 7, pp. 166 803–166 814, 2019.

[18] J. MacQueen, “Some methods for classification and analysis of multivariate observations,” in *Proc. 5th Berkeley Symp. Math. Stat. Probab., Univ. Calif*, 1967, pp. 281–297.

[19] X. Tan and S.-Y. Shin, “Differential evolution algorithm of soft island model based on k-means clustering,” *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 19, p. 1548, 09 2020.

[20] M. Z. Ali, N. Awad, R. Duwairi, J. Albadarneh, R. G. Reynolds, and P. N. Suganthan, “Cluster-based differential evolution with heterogeneous influence for numerical optimization,” in *2015 IEEE Congress on Evolutionary Computation (CEC)*, 2015, pp. 393–400.

[21] C. Cobos, L. Sierra Martínez, and J. Corrales, “Continuous optimization based on a hybridization of differential evolution with k-means,” *Lecture Notes in Computer Science*, vol. 8864, 11 2014.

[22] R. Biedrzycki, J. Arabas, and D. Jagodziński, “Bound constraints handling in differential evolution: An experimental study,” *Swarm and Evolutionary Computation*, vol. 50, no. 100453, 2019.

[23] P. J. Rousseeuw, “Silhouettes: A graphical aid to the interpretation and validation of cluster analysis,” *Journal of Computational and Applied Mathematics*, vol. 20, pp. 53–65, 1987.

[24] P. Suganthan. (2021, Nov.) P.N. Suganthan Github repository. [Online]. Available: https://github.com/P-N-Suganthan

[25] (2022, Apr.) Implementation of the NL-SHADE-RSP-MID. [Online]. Available: https://staff.elka.pw.edu.pl/%7Erbiedrzy/publ/nl-shade-rsp-mid.zip

[26] N. Hansen and A. Ostermeier, “Completely derandomized self-adaptation in evolution strategies,” *Evol. Comput.*, vol. 9, no. 2, pp. 159–195, 2001.

[27] J. Bossek. (2021, Nov.) cmaesr: Covariance matrix adaptation - evolution strategy in R. [Online]. Available: https://github.com/jakobbossek/cmaesr

[28] D. Jagodziński and J. Arabas, “A differential evolution strategy,” in *IEEE Congr. Evol. Comput.*, 2017, pp. 1872–1876.