Differential Evolution with Success Rate-based adaptation CL-SRDE for Constrained Optimization

1st Vladimir Stanovov
Institute of Informatics and Telecommunication
Reshetnev Siberian State University
of Science and Technology
Krasnoyarsk, Russian Federation
vladimirstanovov@yandex.ru

2rd Eugene Semenkin
Institute of Informatics and Telecommunication
Reshetnev Siberian State University
of Science and Technology
Krasnoyarsk, Russian Federation
eugenesemenkin@yandex.ru

***Abstract*—Constrained numerical optimization problems introduce significant challenges for optimization methods. One of the popular heuristic optimization techniques for such problems is the Differential Evolution algorithm. This study focuses on applying a variant of differential evolution with two populations to the set of benchmark problems from the CEC 2024 Constrained Single Objective Numerical Optimization competition. In the proposed CL-SRDE algorithm the adaptation of the scaling factor is performed based on the success rate, which is the number of replaced individuals during selection divided by population size. The constraints are handled by a modified epsilon-constraint method. The analysis of the experimental results show that the used adaptation scheme achieves better feasibility rates and overall performance, compared to the alternative approaches.**

***Index Terms*—differential evolution, constrained optimization, numerical optimization, parameter adaptation**

I. Introduction

The development of heuristic numerical optimization methods is mostly focused on bound-constrained problems, however, a large number of real-world problems from different areas such as engineering, economics, and machine learning include several constraints, that should be satisfied. One of the ways of proposing better constrained optimization methods is to utilize the latest methods, proposed for bound constrained (unconstrained) case. The differential evolution (DE) algorithm has recently received a lot of attention from the scientific community due to its high efficiency and simplicity [1], and today is often used for solving constrained optimization problems (COPs). A typical COP can be formulated as follows:

$$\min_x f(x)$$
$$\text{subject to}$$
$$\begin{cases} g_j(x) \leq 0, j = 1, ..., q \\ h_j(x) = 0, j = q+1, ..., m \end{cases}$$

where f is the the real-valued function $f : X \to R$, $X \subseteq R^n$ and a set of m constraints $g_j(x)$ are q inequality constraints, and $h_j(x)$ are $m-q$ equality constraints, defining the feasible region $\mathcal{F}$. So the goal of solving COP is to find the best x^*, while safisfying the constraints. Obviously, the equality constraints $h_j(x)$ are hard to satisfy completely, so a tolerance level δ is introduced, and set to a small value, e.g. 0.001.

This work was supported by the Ministry of Science and Higher Education of the Russian Federation (Grant No.075-15-2022-1121).

There are many known constraints handling techniques proposed, such as penalty factors [2] and superiority of feasible solutions [3]. The later was further developed into the ε-constraint handling technique [4], which was successfully applied in DE [5].

In this study the CL-SRDE (Constrained Linear population size reduction Success Rate-based Differential Evolution) algorithm is proposed, which is a modification of the L-NTADE algorithm, presented in [6]. The main features of L-NTADE are two populations of newest and best solutions, and specific selection and mutation strategies. In CL-SRDE the success history-based adaptation for the scaling factor F is replaced by the success rate-based adaptation, and the crossover rate repair is used. The constraint handling technique is taken from the CL-SHADE-RSP algorithm, presented in [7], where the ε value is based on the current violation at each generation. The proposed algorithm is tested on the Congress on Evolutionary Computation (CEC) 2024 competition on constrained optimization [8], which is based on the CEC 2017 competition [9] and uses 30-dimensional test functions.

The rest of the paper is organized as follows. In the next section, a brief description of DE and constraint handling techniques is given. In the third section the CL-SRDE is described in detail. The fourth section contains the experimental setup and results, and the fifth section concludes the paper.

II. Related work

A. Differential evolution

The main feature of the differential evolution algorithm is the mutation strategy, which uses difference between vectors in the population to generate new solutions. This allows using the distribution of individuals in the search space to implicitly adapt to the landscape. DE starts by initializing N points $x_{i,j}$ in D-dimensional search space randomly, $i = 1, 2, ...N$, $j = 1, 2, ...D$. After initialization the mutation is applied. One of the popular mutation strategies is called current-to-pbest, and was used in the SHADE algorithm [10]:

$$v_{i,j} = x_{i,j} + F(x_{pbest,j} - x_{i,j}) + F(x_{r1,j} - x_{r2,j}). \quad (1)$$

where x_i is the $i-th$ vector in the population of N individuals, each vector consists of D variables, v_i is the donor vector, $r1$ and $r2$ are random indices from the population, $pbest$ is one of the $p\%$ best individuals, and F is the scaling factor parameter. After mutation the crossover is applied with probability Cr. The most popular method is called binomial crossover:

$$u_{i,j} = \begin{cases} v_{i,j}, & \text{if } rand(0,1) \leq Cr \text{ or } j = jrand \\ x_{i,j}, & \text{otherwise} \end{cases}. \quad (2)$$

Here u_i is the trial vector, and $jrand$ is required to make sure that at least one component is taken from the donor vector. The trial vectors are also processed by the bound constraint handling methods, such as midpoint-target or resampling [11].

After evaluating the new solution, the selection step is applied. DE selection is a simple comparison of the taget and trial vectors:

$$x_{i,j} = \begin{cases} u_{i,j}, & \text{if } f(u_i) \leq f(x_i) \\ x_{i,j}, & \text{if } f(u_i) > f(x_i) \end{cases}. \quad (3)$$

The DE algorithm is known to be highly sensitive to its three main parameters, F, Cr and N. Significant number of studies has been addressing this issue, starting with [12] and jDE [13] algorithms. One of the most efficient adaptation methods is the success history-based adaptation (SHA), described in [10]. SHA adapts both F and Cr in a similar manner by randomly generating parameter values around promising regions and saving the values, that were successful, i.e. delivered improvement in terms of fitness during selection. SHA also relies on the improvement value, i.e. the new means for F and Cr are calculated using weighted Lehmer mean, with weights determined by fitness difference. SHA sets H memory cells containing $M_{F,k}$ and $M_{Cr,k}$ parameters to be further used for sampling with Cauchy and normal distribution respectively: $F = randc(M_{F,h}, 0.1)$, $Cr = randn(M_{Cr,h}, 0.1)$, h is randomly chosen in $[1, H]$. During selection successful values of F and Cr are saved to S_F and S_{Cr} along with fitness improvement $S_{\Delta f} = f(x_i) - f(u_i)$. At the end of each generation two weighted Lehmer means are calculated:

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

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

$$\begin{cases} M_{F,k}^{g+1} = 0.5 \cdot (M_{F,k}^{g} + mean_{(wL,F)}) \\ M_{Cr,k}^{g+1} = 0.5 \cdot (M_{Cr,k}^{g} + mean_{(wL,Cr)}) \end{cases}, \quad (4)$$

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

As for the population size, in L-SHADE algorithm [14] the Linear Population Size Reduction (LPSR) was proposed, in which the population size is reduced at every generation to a minimum value of 4. This allows covering a large portion of the search at the beginning and concentrating search in the most promising region at the end. The success history adaptation and LPSR are applied in many algorithms, which originate from L-SHADE [15], although it has been shown that the variant of weighted Lehmer mean may be suboptimal [16].

Other studies on DE address other issues, for example in [17] the rank-based selection for choosing vectors for mutation was proposed, and it was further studied in [18]. In [19] the general schme of DE was changed, and instead of replacing old individuals, all solutions were saved, and specific selection techniques were applied to direct the search. Based on these findings in [6] the L-NTADE algorithm was proposed. The main idea of L-NTADE is to use two populations, one for the latest generated solutions, and another for the best so far solutions x^{top}. The latest solutions are stored in x^{new}, and this population is updated with a specific selection method:

$$x_{nc} = \begin{cases} u_i, & \text{if } f(u_i) \leq f(x_{r1}^{new}) \\ x_{nc}, & \text{if } f(u_i) > f(x_{r1}^{new}) \end{cases}. \quad (5)$$

where nc is an index, iterated after every successful replacement, and if $nc > N$ it is reset to 1. The trial vector u_i is compared to the basic vector x_{r1}^{new}, and the trial is better, than it replaces individual with index nc, not with index i. Hence, the x^{new} is continuously updated, replacing older solutions. All solutions that are saved to x^{new} are also copied to a temporary pool x^{temp}, and at the end of the generation x^{top} and x^{temp} are joined, sorted by fitness and the best N solutions are returned to x^{top}. The L-NTADE algorithm uses LPSR for both populations, i.e. x^{new} and x^{top} always have the same size.

To use the information from both populations, the r-new-to-ptop/n/t mutation strategy is proposed, which works as follows:

$$v_{i,j} = x_{r1,j}^{new} + F \cdot (x_{pbest,j}^{top} - x_{i,j}^{new}) + F \cdot (x_{r2,j}^{new} - x_{r3,j}^{top}) \quad (6)$$

where $r1$ and $r3$ are chosen randomly, and $r2$ is chosen with rank-based selective pressure with $kp = 3$, $pbest$ is chosen from $pb\%$ best solutions in the top population. Note that unlike current-to-pbest, the basic solution is not x_i, but x_{r1}, i.e. it is chosen randomly.

In [6] is was shown that the performance of L-NTADE can be significantly better than that of the several alternative approaches. However, the algorithm used the success history-based adaptation from L-SHADE, which could be suboptimal. In order to develop a new adaptation technique, in [20] the Hyper-Heuristic [21] approach was used, with Genetic Programming generating equations to adapt F values. It was found that there is a strong connection between the success rate, i.e. the number of replaced solutions, and the mean to generate F around. This allowed us to propose success rate-based adaptation, described in the next section.

B. Constraint handling

In order to deal with several constraintes in a COP, the overall constraint violation $\phi(x)$ is calculated as follows:

$$\phi(x) = \sum_{j=1}^{q} max(0, g_j(x)) + \sum_{j=q+1}^{m} max(0, |h_j(x)| - \delta) \quad (7)$$

The total violation is then used in a constraint handling method. One of the most efficient techniques is the ε-constraint method, proposed in [4] and [22]. The idea is to introduce an order, in which the individuals would be preferred with $\phi(x)$ preceding $f(x)$. Given two function values $f(x_1)$, $f(x_2)$ and two constraint violations $\phi(x_1)$, $\phi(x_2)$ for two points x_1 and x_2 the ordering is defined as follows:

$$x_1 \prec_\varepsilon x_2 \Leftrightarrow \begin{cases} f(x_1) < f(x_2), & \text{if } \phi(x_1), \phi(x_2) \leq \varepsilon \\ f(x_1) < f(x_2), & \text{if } \phi(x_1) = \phi(x_2) \\ \phi(x_1) < \phi(x_2), & \text{otherwise} \end{cases} \quad (8)$$

The solution with smaller violation will be considered better, if both individuals have violations larger than ε (ε-infeasible), and if both are smaller (ε-feasible), then individuals are compared by fitness. Also, if one of them is ε-feasible and the other is not, then the first will be considered better.

The case of $\varepsilon = 0$ corresponds to superiority of feasible solutions (SFS), which could be quite efficient by itself. However, the disadvantage of such approach is that it almost completely prohibits exploration of the infeasible region, and it can be helpful to promote the search [23]. That is why in [22] the ε is initially set to a large value, and gradually reduced every iteration G as follows:

$$\varepsilon(0) = \phi(x_\theta)$$
$$\varepsilon(G) = \begin{cases} \varepsilon(0)(1 - \frac{G}{T_c})^{cp}, & \text{if } 0 < G < T_c \\ 0, & \text{otherwise} \end{cases} \quad (9)$$

where x_θ is the top θ-th in an array sorted by total constraint violation, $\theta = \gamma N$, N is the current population size, $\gamma \in [0.2, 0.8]$, and $cp \in [2, 10]$, $T_c \in [0.1T_{max}, 0.8T_{max}]$, T_{max} is the maximum number of iterations. Note that $\varepsilon = 0$ after T_c, this is required to push the individuals towards feasible region at the end of the search.

In [7] several modifications to the ε-constraint method have been proposed, in which the ε level was tuned not based on the value in the initial population, but during the search process. In particular, the EC_P approach was proposed:

$$\theta = \theta_p N(1 - \frac{NFE}{NFE_{max}})^{cp} \quad (10)$$

$$\varepsilon(G) = \begin{cases} \phi(x_\theta), & NFE < NFE_c \\ 0, & otherwise \end{cases} \quad (11)$$

where NFE is the current number of function evaluations, NFE_{max} is the total available computational resource, NFE_c is the cutoff level set to $0.8NFE_{max}$, θ_p is the control parameter $\in [0, 1]$, and θ is the index of an individual in an array sorted by constraint violation. In other words, ε is set to θ-th largest violation at every generation, with θ decreasing to zero. Such approach is needed to make sure that the ε level corresponds to the current violations $\phi(x)$, allowing a part of the population satisft the constriants, while the other optimizes the target function. As the search progesses, more individuals that violate the constraints are considered infeasible, so the violation of the constraints becomes unacceptable for them.

The EC_P together with other approaches was used in the CL-SHADE-RSP algorithm, and it was shown that setting ε values based on current violation could result in better feasibility rates and overall performance.

III. Proposed approach: CL-SRDE

The CL-SRDE algorithm uses the general sceme of L-NTADE, constraint handling from CL-SHADE-RSP, and adds several new features. First of all, the mF value, used to generate scaling factors F is adapted based on the success rate as follows:

$$mF = SR^{\frac{1}{3}} \quad (12)$$

where $SR = \frac{NS}{N}$, NS is the number of successful solutions, i.e. number of times the selection worked. In other words, mF is set as a cubic root of the success rate. The F values are sampled using normal distribution $F = randn(mF, 0.05)$. This dependence was originally discovered in [20] and further considered in [24]. The adaptation of Cr is performed by SHA, same as in L-SHADE algorithm. Figure 1 shows the curve describing dependence between SR and mF.

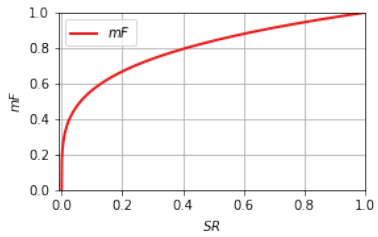

Fig. 1. Dependence of mF on success rate.

The second feature of CL-SRDE is the usage of repaired crossover rate [25]. During each crossover operation, the actual value Cr_a is calculated, as the number of components that were replaced by those from the donor vector. That is, even if $Cr = 0.5$, it is possible that, say, in $10D$ case only 2 values were actually replaced, resulting in $Cr_a = 2/10$. So, it is better to use Cr_a instead of Cr for further adaptation. This makes the tuning of Cr in SHA more accurate, as it follows the used values more precisely.

The full pseudocode of CL-SRDE is given in Algorithm 1. Note that $g(x)$ and $h(x)$ are vectors.

The next section contrains the experimental setup and numerical results.

IV. Experiments and results

The experiments in this study were performed using the set of test functions from the Congress on Evolutionary

Algorithm 1 CL-SRDE

Input: D, NFE_{max}, N_{max}, $f(x)$, $g(x)$, $h(x)$
Output: x_{best}^{top}, $f(x_{best}^{top})$, $g(x_{best}^{top})$, $h(x_{best}^{top})$
Set $N = N_{max}$, $N_{min} = 4$, $H = 5$, $M_{Cr,r} = 1$
Set $SR = 0.5$, $k = 1$, $kp = 7$, $g = 0$, $nc = 1$, $pb = 0.3$
Set $\delta = 0.001$, $NFE_c = 0.8$, $cp = 2$, $\theta_p = 0.8$
Initialize population $(x_{1,j}^{new}, ..., x_{N,j}^{new})$ randomly
Calculate $f(x^{new})$, $g(x^{new})$, $h(x^{new})$ and $\phi(x^{new})$
Copy x^{new} to x^{top}
while $NFE < NFE_{max}$ **do**
  $S_{Cr} = \emptyset$, $S_{\Delta f} = \emptyset$
  $\theta = \theta_p N(1 - \frac{NFE}{NFE_{max}})^{cp}$
  Sort x^{new} by $\phi(x^{new})$
  Calculate $\varepsilon = \phi(x^{new})_\theta$
  Sort x^{top} using ε
  Assign ranks for x^{new}
  for $i = 1$ to N^g **do**
    Sort x^{new} using ε
    repeat
      $F_i = randn(mF, 0.05)$
    until $F_i \in (0, 1)$
    Current memory index $r = randi[1, H]$
    Crossover rate $Cr_i = randn(M_{Cr,r}, 0.1)$
    $Cr_i = min(1, max(0, Cr))$
    $r1 = randi(N^g)$
    repeat
      $pbest = randi(1, N^g \cdot pb)$
      Generate $r2$ with rank-based selection
      $r3 = randi(1, N^g)$
    until indexes $r1$, $r2$, $r3$ and $pbest$ are different
    Apply mutation to produce v_i with F_i
    Apply binomial crossover to produce u_i with Cr_i
    Calculate actual rate Cr_a
    Apply bound constraint handling method
    Calculate $f(u_i)$, $g(u_i)$, $h(u_i)$ and $\phi(u_i)$
    if u_i is better than x_{r1}^{new} (eq. 7) **then**
      $Cr_a \rightarrow S_{Cr}$
      if Replaced by fitness **then**
        $(f(x_{r1}^{new}) - f(u_i)) \rightarrow S_{\Delta f}$
      else
        $(\phi(x_{r1}^{new}) - \phi(u_i)) \rightarrow S_{\Delta f}$
      end if
      $u_i \rightarrow x^{temp}$
      $x_{nc}^{new} = u_i$
      $nc = mod(nc + 1, N^g)$, $m = m + 1$
    end if
  end for
  Get N^{g+1} with LPSR
  Join together x^{top} and x^{temp} and sort using ε
  Copy N^{g+1} best vectors back to x^{top}
  if $N^g > N^{g+1}$ **then**
    Remove worst individuals from x^{new}
  end if
  Update $M_{Cr,k}$
  $k = mod(k + 1, H)$, $g = g + 1$, $m = 1$
end while
Return x_{best}^{top}, $f(x_{best}^{top})$, $g(x_{best}^{top})$, $h(x_{best}^{top})$

Computation 2024 competition on single objective constrained numerical optimization [8]. The 2024 competition is based on the CEC 2017 benchmark [9], with 28 constrained test functions defined for dimensions 10, 30, 50 and 100. However, CEC 2024 uses only $30D$ test functions, and differs in evaluation criteria: the ranking for the CEC 2024 also considers the number of function evaluations required to get to the optimum, if it was found during the run. The number of runs was set to 25, and the computational resource $20000D$ evaluations. In addition to $30D$ functions, we have performed experiments for $10D$ and $50D$ cases. The test functions contained both inequality and equality constraints, and the tolerance level was set to $\delta = 0.001$.

The CL-SRDE algorithm had the following parameters: population size $N_{max} = 600$, number of memory cells for Cr adaptation $H = 5$, selective pressure coefficient $kp = 7$, mutation greediness parameter was fixed at $pb = 0.3$, and the ε-constraint method was switched to SFS after cutoff level $NFE_c = 0.8$, i.e. after 80% of the computational resource were spent. The adaptation of F values used cubic root of the success rate to set the mean parameter mF. The CL-SRDE algorithm was implemented in C++, compiled with GCC and ran under Ubuntu 20.04 machine with AMD Ryzen 5800X processor. The results post-processing, ranking and statistical tests were performed with Python 3.8.

A. CEC 2017 comparison

The comparison was performed with several alternative approaches, for which the results were available, including CAL-LSHADE [26], LSHADE44 [5], LSHADE44-IDE [27], UDE [28], LSHADE_IEpsilon [29], MA-ES [30] IUDE [29] and CLSHADE-RSP-IFL (with the combined fitness-epsilon vector length approach from [7]). In addition, the CL-SRDE with success history based adaptation for F was tested, shown as CL-SRDE (SHA). As for these methods the number of function evaluations to get to the optimum was not measured, the comparison was performed according to the CEC 2017 rules. For this the best, mean and median of the target function values were determined, as well as the median and mean constraint violation and feasibility rates. Two rankings were applied, in the first the algorithms were ranked based on feasibility rate, mean violation and mean objective function value (in this particular order). In the second ranking the algorithms were compared by median solutions based on feasibility, target function value and constraint violation. The final ranking was the sum or ranks for all 28 test problems. If two algorithms had the same feasibility rate, violation and target function value, then they received fractional ranks in order to perform tie-breaking properly. Table I contains the ranking of the mentioned algorithms, lower ranks are better.

As Table I shows, the CL-SRDE was able to get the best rank in all cases cases. Moreover, using the success history based adaptation for scaling factor F only decreased the performance of the algorithm.

Table II contains the Mann-Whitney statistical tests comparing CL-SRDE with CLSHADE-RSP algorithm. The ranking in

TABLE I
RANKING ACCORDING TO CEC17 RULES AND COMPARISON WITH OTHER METHODS.

Algorithm	$10D$	$30D$	$50D$	Total
CAL-LSHADE	438.00	413.84	444.50	1296.34
LSHADE44	324.50	366.84	380.17	1071.51
LSHADE44-IDE	336.14	332.51	313.67	982.32
UDE	327.50	331.51	317.17	976.17
LSHADE_IEpsilon	284.64	297.84	311.67	894.15
MA-ES	327.64	313.84	302.50	943.98
IUDE	246.14	277.72	250.17	774.03
CLSHADE-RSP	248.14	243.72	234.06	725.92
CL-SRDE (SHA)	323.14	319.67	336.61	979.42
CL-SRDE	**224.14**	**182.51**	**189.50**	**596.15**

the statistical test was performed using superiority of feasible solutions approach, and the significance level was set to $p = 0.01$. As the statistics were approximated by normal distribution, it allowed calculating the standard Z score as a sum of scores over all functions. For every function the Z score within the range $(-2.58, 2.58)$ meant that the difference between algorithms is insignificant.

TABLE II
MANN-WHITNEY TESTS OF CL-SRDE AGAINS CLSHADE-RSP, CEC 2017 BENCHMARK

Dimension	Wins	Ties	Losses	Total Z score
10	8	16	4	17.02
30	5	17	6	3.27
50	7	15	6	9.33

As Table II shows, the CL-SRDE algorithm performs better than CLSHADE-RSP in $10D$ case on 8 functions, but worse on 5 functions. In particular, worse performance of CL-SRDE was observed on functions 3, 4, 16, 22, 25, and better performance on 7, 12, 18, 20, 21, 23, 26, 27. In $30D$ case there were 6 wins and 6 losses, in particular on functions 3, 4, 16, 17, 25, 26 and 12, 18, 21, 23, 27, 28. In $50D$ case the CL-SRDE was worse on 8 functions and better only on 6. The observed losses of CL-SRDE were mostly due to larget target function value, as both algorithms were able to find feasible solutions. The feasibility rates for CLSHADE-RSP are 83.57, 84.43, 80.0 for $10D$, $30D$ and $50D$ respectively, and 85.57, 85.43, 83.14 for CL-SRDE. The CL-SRDE was not able to get feasible solutions only for four problems with numbers 17, 19, 26, 28. Problems 17 and 26 constraints included the sign function, i.e. were characterised by plateaus.

B. CEC 2024 results

The presentation of the results for the CEC 2024 competition requires the algorithm complexity calculation. Table III contains the time complexity results of the CL-SRDE algorithm. Here T1 is the average time required to evaluate all 28 functions 10000 times, and T2 is the average computing time of the algorithm on all functions with 10000 evaluations computational resource.

The results of CL-SRDE on the CEC 2024 benchmark for $10D$, $30D$ and $50D$ are presented in Tables IV-X. In particular,

TABLE III
ALGORITHM COMPLEXITY OF CL-SRDE, CEC2024

Dimension	T1	T2	(T2-T1)/T1
$10D$	0.0062	0.075	11.005
$30D$	0.019	0.089	3.596
$50D$	0.033	0.105	2.170

TABLE IV
CEC 2024 RESULTS OF CL-SRDE, FUNCTIONS 1-12, $30D$

Funcion	1	2	3	4
Best	0	0	334.79	31.838
Median	0	0	471.1	47.758
$\overline{v}$	0	0	0	0
Mean	6.2567E-31	4.3165E-30	502.12	46.245
Worst	6.2567E-31	4.3165E-30	502.12	46.245
Std	1.9717E-30	7.391E-30	105.23	8.6971
FR	100	100	100	100
$\overline{vio}$	0	0	0	0
Funcion	5	6	7	8
Best	0	0	-1118.3	-0.00028398
Median	0	0	-816.95	-0.00028398
$\overline{v}$	0	0	0	0
Mean	0	0	-811.12	-0.00028398
Worst	0	0	-811.12	-0.00028398
Std	0	0	200.4	2.713E-16
FR	100	100	100	100
$\overline{vio}$	0	0	0	0
Funcion	9	10	11	12
Best	-0.0026655	-0.00010284	-0.92493	3.9825
Median	-0.0026655	-0.00010284	-0.92493	9.7752
$\overline{v}$	0	0	0	0
Mean	-0.0026655	-0.00010284	-0.83029	8.8483
Worst	-0.0026655	-0.00010284	-0.83029	8.8483
Std	4.3368E-19	1.9596E-16	0.15733	2.1236
FR	100	100	100	100
$\overline{vio}$	0	0	0	0

the best, median, mean, standard deviation and worst target function value are shown, as well as the mean value of the violations of all constraints at the median solution $\overline{v}$ and the mean constraint violation value of all the solutions of 25 runs $\overline{vio}$. Finally, the mean number of function evaluations to reach the optimum $\overline{NFE}$ is given.

V. CONCLUSION

This paper proposed the CL-SRDE algorithm, a modification of the L-NTADE with success rate-based parameter adaptation for constrained optimization problems. Other features of the CL-SRDE are the crossover rate repair and a modified ε calculation method, taken from CLSHADE-RSP algorithm. The experiments have shown that compared to alternative approaches the CL-SRDE is able to show competitive performance not only in $30D$ case, but also in other dimensions.

REFERENCES

[1] S. Das and P. Suganthan, "Differential evolution: a survey of the state-of-the-art," *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 1, pp. 4–31, 2011.

TABLE V
CEC 2024 RESULTS OF CL-SRDE, FUNCTIONS 13-28, $30D$

Funcion	13	14	15	16
Best	0	1.4085	2.3563	18.849
Median	0	1.4954	5.4977	25.133
$\overline{v}$	0	0	0	0
Mean	2.5685E-28	1.4572	6.503	22.557
Worst	2.5685E-28	1.4572	6.503	22.557
Std	2.8977E-28	0.043147	1.7138	2.7345
FR	100	100	100	100
$\overline{vio}$	0	0	0	0
Funcion	17	18	19	20
Best	0.43969	36.52	0	1.2237
Median	0.87258	36.52	0	2.7067
$\overline{v}$	15.5	0	21375	0
Mean	0.83724	36.52	0	2.7876
Worst	0.83724	36.52	0	2.7876
Std	0.12153	6.1969E-06	0	0.6838
FR	0	100	0	100
$\overline{vio}$	15.5	0	21375	0
Funcion	21	22	23	24
Best	9.7752	2.1741E-26	1.4085	5.4977
Median	28.326	3.0654E-26	1.4085	5.4977
$\overline{v}$	0	0	0	0
Mean	25.982	3.3104E-26	1.4085	5.4977
Worst	25.982	3.3104E-26	1.4085	5.4977
Std	10.05	1.0782E-26	2.2204E-16	8.8818E-16
FR	100	100	100	100
$\overline{vio}$	0	0	0	0
Funcion	25	26	27	28
Best	18.849	0.53253	36.52	0
Median	25.133	0.895	36.52	0
$\overline{v}$	0	15.5	0	21375
Mean	24.504	0.87023	36.52	0
Worst	24.504	0.87023	36.52	0
Std	3.4126	0.097755	2.5654E-05	0
FR	100	0	100	0
$\overline{vio}$	0	15.5	0	21375

[2] C. A. C. Coello, "Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: A survey of the state of the art," *Computer Methods in Applied Mechanics and Engineering*, vol. 191, pp. 1245–1287, 2002. [Online]. Available: https://api.semanticscholar.org/CorpusID:62303805

[3] K. Deb, "An efficient constraint handling method for genetic algorithms," *Computer Methods in Applied Mechanics and Engineering*, vol. 186, pp. 311–338, 2000. [Online]. Available: https://api.semanticscholar.org/CorpusID:12321441

[4] T. Takahama and S. Sakai, "Constrained optimization by epsilon constrained particle swarm optimizer with epsilon-level control," in *WSTST*, 2005. [Online]. Available: https://api.semanticscholar.org/CorpusID:35365399

[5] R. Poláková, "L-shade with competing strategies applied to constrained optimization," *2017 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1683–1689, 2017. [Online]. Available: https://api.semanticscholar.org/CorpusID:1128936

[6] V. Stanovov, S. Akhmedova, and E. Semenkin, "Dual-population adaptive differential evolution algorithm l-ntade," *Mathematics*, 2022.

[7] ——, "Combined fitness–violation epsilon constraint handling for differential evolution," *Soft Computing*, vol. 24, pp. 7063–7079, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:215857659

[8] K. Qiao, X. Wen, X. Ban, P. Chen, K. V. Price, P. N. Suganthan, J. Liang, G. Wu, and C. Yue, "Evaluation criteria for cec 2024 competition and special session on numerical optimization considering accuracy and speed," Zhengzhou University, Central South University, Henan Institute of Technology, Qatar University, Tech. Rep., 2023.

[9] G. Wu, R. Mallipeddi, and P. N. Suganthan, "Problem definitions and evaluation criteria for the cec 2017 competition on constrained realparameter optimization," 2016.

[10] R. Tanabe and A. Fukunaga, "Success-history based parameter adaptation for differential evolution," in *Proceedings of the IEEE Congress on Evolutionary Computation*. IEEE Press, 2013, pp. 71–78.

[11] R. Biedrzycki, J. Arabas, and D. Jagodziński, "Bound constraints handling in differential evolution: An experimental study," *Swarm Evol. Comput.*, vol. 50, 2019.

[12] A. Qin and P. Suganthan, "Self-adaptive differential evolution algorithm for numerical optimization," in *Proceedings of the IEEE Congress on Evolutionary Computation*, 2005, pp. 1785–1791.

[13] J. Brest, S. Greiner, B. Boškovic, M. Mernik, and V. Žumer, "Self-adapting control parameters in differential evolution: a comparative study on numerical benchmark problems," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 6, pp. 646–657, 2006.

[14] R. Tanabe and A. Fukunaga, "Improving the search performance of SHADE using linear population size reduction," in *Proceedings of the IEEE Congress on Evolutionary Computation, CEC*, Beijing, China, July 2014, pp. 1658–1665.

[15] 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.

[16] V. Stanovov, S. Akhmedova, and E. Semenkin, "Biased parameter adaptation in differential evolution," *Information Sciences*, vol. 566, pp. 215–238, 2021.

[17] W. Gong and Z. Cai, "Differential evolution with ranking-based mutation operators," *IEEE Transactions on Cybernetics*, vol. 43, pp. 2066–2081, 2013. [Online]. Available: https://api.semanticscholar.org/CorpusID:1755811

[18] V. Stanovov, S. Akhmedova, and E. Semenkin, "Selective pressure strategy in differential evolution: Exploitation improvement in solving global optimization problems," *Swarm Evol. Comput.*, vol. 50, 2019.

[19] T. Kitamura and A. Fukunaga, "Differential evolution with an unbounded population," *2022 IEEE Congress on Evolutionary Computation (CEC)*, 2022.

[20] V. Stanovov and E. Semenkin, "Genetic programming for automatic design of parameter adaptation in dual-population differential evolution," *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:260119279

[21] E. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. Woodward, *A Classification of Hyper-Heuristic Approaches: Revisited*. Springer International Publishing, 2019, pp. 453–477.

[22] T. Takahama and S. Sakai, "Constrained optimization by the ε constrained differential evolution with gradient-based mutation and feasible elites," *2006 IEEE International Conference on Evolutionary Computation*, pp. 1–8, 2006. [Online]. Available: https://api.semanticscholar.org/CorpusID:983972

[23] H. K. Singh, K. Alam, and T. Ray, "Use of infeasible solutions during constrained evolutionary search: A short survey," in *Australasian Conference on Artificial Life and Computational Intelligence*, 2016. [Online]. Available: https://api.semanticscholar.org/CorpusID:39691733

[24] V. Stanovov and E. Semenkin, "Surrogate-assisted automatic parameter adaptation design for differential evolution," *Mathematics*, vol. 11, no. 13, 2023. [Online]. Available: https://www.mdpi.com/2227-7390/11/13/2937

[25] W. Gong, Z. Cai, and Y. Wang, "Repairing the crossover rate in adaptive differential evolution," *Appl. Soft Comput.*, vol. 15, pp. 149–168, 2014. [Online]. Available: https://api.semanticscholar.org/CorpusID:41345493

[26] A. Zamuda, "Adaptive constraint handling and success history differential evolution for cec 2017 constrained real-parameter optimization," *2017 IEEE Congress on Evolutionary Computation (CEC)*, pp. 2443–2450, 2017. [Online]. Available: https://api.semanticscholar.org/CorpusID:21947974

[27] J. Tvrdík and R. Poláková, "A simple framework for constrained problems with application of l-shade44 and ide," *2017 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1436–1443, 2017. [Online]. Available: https://api.semanticscholar.org/CorpusID:25514207

[28] A. Trivedi, K. Sanyal, P. P. Verma, and D. Srinivasan, "A unified differential evolution algorithm for constrained optimization problems," *2017 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1231–1238, 2017. [Online]. Available: https://api.semanticscholar.org/CorpusID:2642562

[29] A. Trivedi, D. Srinivasan, and N. Biswas, "An improved unified differential evolution algorithm for constrained optimization problems." [Online]. Available: https://api.semanticscholar.org/CorpusID:160003726

[30] M. Hellwig and H.-G. Beyer, "A matrix adaptation evolution strategy for constrained real-parameter optimization," *2018 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1–8, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:52931376