RDEx: An effectiveness-driven hybrid Constrained Multi-Objective optimizer that adaptively selects and combines multiple operators and strategies

1st Sichen Tao
Department of Engineering
University of Toyama
Toyama-shi 930-8555, Japan
taosc73@hotmail.com

2nd Yifei Yang
Faculty of Science and Technology
Hirosaki University
Hirosaki-shi 036-8560, Japan
yyf7236@hirosaki-u.ac.jp

3rd Ruihan Zhao
School of Mechanical Engineering
Tongji University
Shanghai-shi 200082, China
ruihan.z@outlook.com, 2110415@tongji.edu.cn

4th Kaiyu Wang
Chongqing Institute of Microelectronics Industry Technology
University of Electronic Science and Technology of China
Chongqing-shi 401332, China
greskowky1996@163.com

5th Sicheng Liu
Department of Information Engineering
Yantai Vocational College
Yantai-shi 264670, China
lsctoyama2020@gmail.com

6th Shangce Gao
Department of Engineering
University of Toyama
Toyama-shi 930-8555, Japan
gaosc@eng.u-toyama.ac.jp

***Abstract*—**

I. Introduction

Constrained multi-objective optimization problems (CMOPs) pose significant challenges in evolutionary computation. These problems require simultaneously optimizing multiple conflicting objectives while satisfying complex constraints, yielding a set of trade-off solutions (a Pareto-optimal front) instead of a single optimum. Designing algorithms for CMOPs is difficult because the search must balance exploration of diverse solutions, exploitation of high-quality regions, and strict constraint handling. Evolutionary algorithms (EAs) are well-suited to CMOPs as they maintain a population of solutions and can approximate the Pareto-optimal front in a single run [2]. Among these, *Differential Evolution* (DE) has emerged as a powerful yet simple population-based optimizer, known for fast convergence and requiring minimal parameters [1], [3]. Over the years, DE has been extended to multi-objective contexts with specialized variation operators and selection mechanisms to handle multiple objectives effectively [4].

In recent years, the development of tailored DE variants for constrained multi-objective problems has been spurred by international competitions such as the IEEE CEC. Notably, the *CEC 2024 Competition on Constrained Multi-Objective Problems* (CEC'24 CMOP) showcased state-of-the-art algorithms, with the winning entry being a DE-based approach called *DESDE* [7]. DESDE distinguished itself by demonstrating superior performance on the CEC'24 CMOP benchmark suite, effectively handling constraints and multiple objectives to produce high-quality solutions. It incorporated advanced strategies such as a polynomial mutation operator (as commonly used in multi-objective EAs [2]) and other customized genetic operators that contributed to its success. However, there remained room for improvement in both solution quality and computational efficiency. For instance, the polynomial mutation update operator, while useful for local exploration, added extra computational overhead. Additionally, maintaining population diversity and guiding the search towards the feasible Pareto-optimal region can become increasingly difficult as problem complexity grows (especially when feasible regions are small or fragmented [6]).

Building upon DESDE's foundation, we propose **RDEx**, an enhanced Differential Evolution variant with targeted modifications to improve performance on the latest CMOP benchmarks (e.g., the CEC'25 CMOP test suite). The goal of RDEx is to further *increase solution accuracy* (i.e., find a Pareto front closer to the true optimum with better convergence on objectives) and *reduce computational cost* (i.e., achieve results with fewer function evaluations or less runtime) when tackling complex constrained multi-objective problems. RDEx retains the basic evolutionary framework of DESDE but introduces several key changes to its variation and constraint-handling procedures. Specifically, RDEx includes the following improvements over its predecessor:

- **Removal of the Polynomial Mutation Update:** The dedicated polynomial mutation operator from DESDE is eliminated to streamline the algorithm. This simplification reduces computational overhead and algorithmic complexity, while reliance on other diversity mechanisms compensates for its removal.
- **Cauchy Distribution-Based Random Jumps in Crossover:** A dimension-wise random perturbation based on the *Cauchy distribution* is introduced during the crossover phase. This heavy-tailed perturbation occasionally produces large jumps for decision variables, helping the population escape local optima and maintain diversity. Most of the time the perturbation is small, but infrequent large Δ jumps (drawn from Cauchy) inject strong exploration capability [5].
- **Fitness-Guided Mutation Vector Selection:** In the differential mutation step, the two random individuals (often denoted 'rand1' and 'rand2') used to generate the mutant vector are sorted by fitness (objective performance). Ensuring that the difference vector is oriented from a poorer individual toward a better one means that the mutation is consistently guided by a superior solution. This pushes offspring search directions toward regions of higher fitness, accelerating convergence.
- **Partial Reinitialization for Boundary Violations:** To handle boundary constraints (decision variables exceeding allowed bounds), RDEx employs a partial reinitialization repair strategy. Instead of discarding an out-of-bounds offspring or simply clamping its values, RDEx reinitializes *half* of the decision variables of an infeasible offspring by copying the corresponding values from its parent. This partial reset preserves some of the parent's feasible genetic material, effectively nudging the offspring back into the valid search space without entirely randomizing it.

II. Methodology

In this section, we describe the RDEx algorithm in detail, emphasizing how it differs from the base DESDE approach. RDEx follows the general framework of a multi-objective differential evolution algorithm: it maintains a population of candidate solutions and iteratively applies variation operators (mutation and crossover) to generate offspring, followed by a selection mechanism that prefers non-dominated and feasible solutions to progress towards the Pareto front. RDEx inherits from DESDE the overall structure for handling multiple objectives and constraints (for example, the use of Pareto dominance-based selection and a constraint-violation comparison rule), but introduces modifications in key components of the evolutionary cycle. We outline the main components of RDEx and highlight the improvements made in each:

A. Elimination of Polynomial Mutation Operator

One key change in RDEx is the *omission of the polynomial mutation update* that was part of DESDE's variation toolkit. In many multi-objective evolutionary algorithms (e.g., NSGA-II [2]), a polynomial mutation operator is used to introduce small random perturbations in solutions, following a probability distribution that heavily favors small moves (with occasional larger moves). DESDE included a similar operator to enhance local exploration. However, such an operator can increase the computational load and adds extra parameters (such as a distribution index) to tune. RDEx opts to simplify the algorithm by removing this step entirely. By doing so, RDEx reduces the number of variation operations per generation, which **lowers the overall computational cost**. Any potential loss of exploratory capability from removing polynomial mutation is compensated by the newly introduced Cauchy-based jump in the crossover (described below) and the inherent stochasticity of DE's mutation and crossover. In practice, we found that eliminating the polynomial update streamlines the algorithm without sacrificing performance, as the other mechanisms in RDEx sufficiently maintain population diversity.

B. Cauchy Distribution-Based Random Jumps in Crossover

In RDEx's crossover operator, after the usual DE mutation has produced a donor vector (mutant solution), crossover is performed between the target parent and the donor to create a trial offspring. *Cauchy perturbations* are applied as follows: for each decision variable that is selected to inherit from the donor (according to the crossover probability CR), the value is not copied verbatim. Instead, RDEx perturbs this value by adding a random offset Δ drawn from a Cauchy distribution. Specifically, if v_i is the i-th component of the donor vector and u_i is the resulting offspring value, then

$$u_i = v_i + \Delta_i, \qquad \Delta_i \sim \text{Cauchy}(0, \gamma),$$

where γ is a scale parameter controlling the spread of the Cauchy distribution. This heavy-tailed random jump mechanism injects additional randomness into the offspring. Unlike a Gaussian or uniform perturbation, the Cauchy's heavy tail ensures that, on rare occasions, Δ_i is large, causing a significant jump in that coordinate. By doing this independently for each dimension, RDEx can occasionally generate offspring that are quite far from their parents, thereby exploring previously unvisited regions. At the same time, because most Cauchy samples are near zero, the crossover usually makes only mild changes (preserving exploitation of good donor information in most generations). This strategy maintains diversity and helps escape local optima. It effectively replaces the exploratory role of the removed polynomial mutation by providing another source of variation, integrated directly into the crossover step with minimal overhead. Previous studies have shown that heavy-tailed mutations can significantly improve exploration in DE [5], and our results confirm that this mechanism leads to better coverage of the Pareto front and avoidance of stagnation.

C. Fitness-Guided Differential Mutation

Differential evolution's core operation is the creation of a mutant vector via scaled differences of individuals. In a classical DE/rand/1 mutation, for each target individual x_t,

one selects three distinct individuals from the population (say x_a, x_b, x_c) and creates a mutant $v = x_a + F \cdot (x_b - x_c)$, where F is a scaling factor. RDEx modifies this process by introducing a **fitness-guided selection** for the difference vector. Specifically, when choosing the two individuals that form the difference (x_b and x_c in the example), RDEx first ensures they are distinct and different from x_a (the base), then sorts these two by their fitness or quality. Let p_{best} denote the better (more fit) individual and q_{worse} the poorer one among the two. The mutant is then generated as:

$$v = x_{\text{base}} + F \cdot (p_{\text{best}} - q_{\text{worse}}).$$

By guiding the differential mutation with the fitter individual, the resultant difference vector $(p_{\text{best}} - q_{\text{worse}})$ points *towards* a region of higher objective performance in the search space. This is a form of greedy bias: the mutation step is biased in a direction more likely to improve the target solution. Consequently, offspring generated by this guided mutation tend to inherit search directions that lead to performance improvements, thus potentially accelerating convergence. It is important to note that in a multi-objective sense, "fitness" here might be determined by non-dominated sorting rank and crowding distance (for example, an individual on a better Pareto front or with higher diversity is considered fitter), or by a scalarized fitness if using an aggregation approach. RDEx consistently uses the better of the two selected individuals as the positive reference in the difference, ensuring a search pressure toward promising regions. This strategy improves the algorithm's ability to refine solutions (exploitation) without completely sacrificing diversity (since the base individual and the random selection still introduce randomness). Overall, the fitness-guided mutation helps RDEx achieve higher solution quality more quickly than an unguided approach.

D. Partial Reinitialization for Boundary Constraints

RDEx introduces a nuanced repair mechanism for boundary violations: **partial reinitialization using the parent's genes**. When an offspring solution (trial vector) is found to have one or more decision variables outside the permitted range, RDEx does not throw the solution away. Instead, it performs the following repair: it randomly selects *half* of the dimensions of this offspring and replaces those coordinates with the corresponding values from its parent (the original target individual from which this offspring was generated). The remaining half of the dimensions of the offspring are left as they were (unless they still violate bounds, in which case another minimal adjustment like clipping is done for those specific dimensions). For example, if a parent solution is $x = (x_1, x_2, \ldots, x_D)$ and an offspring was generated $u = (u_1, u_2, \ldots, u_D)$, and some u_j lies outside the allowed range for index j, then we choose a random subset $S \subset \{1, \ldots, D\}$ of size $D/2$. For each $i \in S$, we set $u_i := x_i$ (copying the parent's value, which is guaranteed in-bounds), and for each $k \notin S$, we keep u_k as originally generated (or clipped if still out of bounds). This results in a *blended offspring*: half of its genes exactly inherited from the feasible parent, and half from the mutated child.

The rationale is that the parent is a known feasible solution, so inheriting some of its variables helps pull the offspring back into a feasible region. Yet, by not simply copying the parent entirely (which would erase the offspring's exploratory changes), the offspring still retains some new genetic material from mutation/crossover, preserving exploration potential. This balance prevents the algorithm from undoing all progress whenever an out-of-bounds situation occurs, while still correcting the violation. Partial reinitialization is thus an effective compromise between *full reinitialization* (which would randomize the solution completely, losing valuable information) and *no repair* (which would leave an invalid solution). It repairs the violation in a guided way by leveraging the parent's feasibility.

Overall, the above methodological enhancements allow RDEx to outperform baseline algorithms like DESDE. The removal of the polynomial mutation operator and use of simpler yet effective variation operators make the algorithm **leaner and faster**. The Cauchy-based jumps and fitness-sorted mutation vectors work in tandem to **improve convergence**: the former injects diversity to explore new regions (preventing stagnation), while the latter intensifies the search in fruitful directions (promoting faster attainment of the Pareto front). The partial reinitialization ensures robust **constraint handling** for boundary constraints, preserving feasibility without sacrificing novel solutions. Empirical comparisons (which will be presented in the next section) confirm that these methodological improvements translate into state-of-the-art performance on the CEC'25 CMOP benchmark suite: RDEx achieves higher solution accuracy than DESDE and does so with fewer evaluations, making it a highly effective algorithm for solving complex constrained multi-objective optimization problems.

III. Experiments and Results

A. Metrics and How to Read Table III

For each CEC-style CMOP (F1–F15), we run **25** independent trials under the same evaluation budget and summarize the primary performance indicator with **best/mean/std/worst** (smaller is better). The indicator is a minimization-style quality measure on the obtained (feasible) nondominated set (e.g., IGD/IGD$^+$ or any monotone loss of HV).[1]

Constraint reporting. For each function we also report **LCV** (least total constraint violation) achieved among the 25 runs. By definition, $LCV = 0$ iff at least one run produced a feasible nondominated set. When all runs are infeasible, $LCV > 0$.

NaN convention. If a run yields no evaluable indicator (e.g., no feasible nondominated solutions), the indicator for that run is recorded as `NaN`. We then apply a *strict NaN-propagation* rule to emphasize robustness: if any of the 25 runs is `NaN`, we set `mean/std/worst` to `NaN`, while `best` is computed

[1]If a maximization indicator such as HV is used, we convert it to a minimization loss, or equivalently reverse the inequality in ranking.

over the valid (non-NaN) runs. This explains, e.g., F14 where best is finite but mean/std/worst are NaN: some runs failed to produce an evaluable front even though at least one run succeeded (consistent with $LCV = 0$).

All numbers are computed from the *final* population (plus archive, if maintained) at the evaluation budget. Smaller values denote better convergence/diversity trade-offs (closer approximation to the reference Pareto front).

How to read: (i) rows with LCV=0 indicate that at least one run reached feasibility; (ii) cells with NaN in mean/std/worst (e.g., F14) reveal instability—some runs produced no evaluable front; (iii) small gaps among best/mean/worst with tiny std imply stable performance across seeds (e.g., F12); (iv) large std and a wide best→worst span (e.g., F5–F6, F9–F11) suggest sensitivity to landscape/constraints; (v) monotone increases over F5→F6 and F9→F10 are typical when moving from simpler to more composite constrained functions.

B. Overall Ranking (U-score)

Given algorithms $\mathcal{A}$ and functions $\mathcal{F}$, we compute per-function ranks using the *mean* value of the primary indicator (lower is better). Ties are broken by, in order: (1) LCV (lower is better), (2) *best* (lower), (3) *std* (lower), (4) *worst* (lower). The aggregate score of $a \in \mathcal{A}$ is

$$U(a) = \sum_{m \in \mathcal{F}} \mathrm{rank}_m(a),$$

where a smaller $U(a)$ indicates better overall performance.

For significance, we use the Wilcoxon signed-rank test on per-function means (paired by function), with Holm–Bonferroni correction at $\alpha = 0.05$. For multi-algorithm comparisons, we additionally report a Friedman test followed by Nemenyi post-hoc analysis. When some means are NaN (due to strict NaN-propagation), we conduct the tests on the subset of functions where all compared algorithms have valid means and separately list the count of excluded cases.

Table III presents the rankings computed by the U-score (smaller is better). DESDE is the CEC'24 champion. On the same CMOP test suite, RDEx achieves a substantial improvement over prior CMOP algorithms, further confirming the effectiveness of RDEx's hybrid mechanism.

REFERENCES

[1] R. Storn and K. Price, "Differential Evolution – a simple and efficient heuristic for global optimization over continuous spaces," *J. Global Optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[2] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

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

[4] S. Kukkonen and J. Lampinen, "GDE3: The third evolution step of generalized differential evolution," in *Proc. IEEE Congress on Evolutionary Computation (CEC)*, vol. 1. Edinburgh, UK: IEEE, 2005, pp. 443–450.

[5] M. Ali and M. Pant, "Improving the performance of differential evolution algorithm using Cauchy mutation," *Soft Computing*, vol. 15, no. 5, pp. 991–1007, 2011.

[6] Y. G. Woldesenbet, G. G. Yen, and B. G. Tessema, "Constraint handling in multi-objective evolutionary optimization," *IEEE Trans. Evolutionary Computation*, vol. 13, no. 3, pp. 514–525, 2009.

[7] IEEE WCCI 2024 Competition on Constrained Multiobjective Optimization (CMOP), Yokohama, Japan, 2024.

Algorithm 1: RDEx for Constrained Multi-Objective Optimization (Part 1/2)

Input: N, D, bounds $\{[l_j, u_j]\}_{j=1}^{D}$, $G_{\max}$, F, CR, Cauchy scale γ, perturb prob. p, initial elite rate $\beta_{\text{init}} \in (0, 1)$
Output: population $\{\mathbf{x}_{i,g}\}$ after variation/repair in gen. g, current archive A, elite-rate β_g
Init: **for** $i = 1..N$ **do**
  for $j = 1..D$ **do**
    sample $x_{i,j,0} \sim \mathcal{U}(l_j, u_j)$
  evaluate $\mathbf{f}(\mathbf{x}_{i,0})$ and violation $v(\mathbf{x}_{i,0})$
$A \leftarrow$ nondominated feasible solutions in $\{\mathbf{x}_{i,0}\}$; $\beta_0 \leftarrow \beta_{\text{init}}$
for $g = 0, 1, \ldots, G_{\max} - 1$ **do**
  compute **quality labels** (constrained rank: feasible $\prec$ infeasible; among feasible, by Pareto rank and crowding; among infeasible, by $v(\cdot)$)
  for $i = 1..N$ **do**
    draw distinct $b, r_1, r_2 \neq i$; `// b will be base`
    order $\{r_1, r_2\}$ by quality $\Rightarrow$ p_{best} (better), q_{worse} (worse)
    if *with prob.* β_g *(elite-guided)* **then**
      $b \leftarrow i$; sample elite $\mathbf{x}^\star$ (rank-weighted)
      $\mathbf{v}_{i,g} \leftarrow \mathbf{x}_{b,g} + F(\mathbf{x}^\star - \mathbf{x}_{b,g}) + F(\mathbf{x}_{p_{\text{best}},g} - \mathbf{x}_{q_{\text{worse}},g})$
    else
      $b \leftarrow r_1$;
      $\mathbf{v}_{i,g} \leftarrow \mathbf{x}_{b,g} + F(\mathbf{x}_{p_{\text{best}},g} - \mathbf{x}_{q_{\text{worse}},g})$
    `// Binomial crossover`
    pick $j_{\text{rand}} \in \{1, \ldots, D\}$; **for** $j = 1..D$ **do**
      $u'_{i,j,g} \leftarrow \begin{cases} v_{i,j,g}, & \text{if } \texttt{rand}() < CR \text{ or } j = j_{\text{rand}} \\ x_{i,j,g}, & \text{otherwise} \end{cases}$
    `// Cauchy perturbation + bound-aware repair (partial reinit)`
    for $j = 1..D$ **do**
      if *with prob.* p **then**
        draw $\delta_j \sim \text{Cauchy}(0, \gamma)$; $u_{i,j,g} \leftarrow u'_{i,j,g} + \delta_j$
      else
        $u_{i,j,g} \leftarrow u'_{i,j,g}$
      if $u_{i,j,g} \notin [l_j, u_j]$ **then**
        mark dim j as violated
    if *any bound violated* **then**
      pick $S \subset \{1, \ldots, D\}$ with $|S| = \lfloor D/2 \rfloor$ (uniform)
      for $j \in S$ **do**
        $u_{i,j,g} \leftarrow x_{i,j,g}$
        `// inherit from feasible parent`
      for $j \notin S$ **do**
        $u_{i,j,g} \leftarrow \min\{u_j, \max\{l_j, u_{i,j,g}\}\}$
        `// clip residual`
    `// Only mutation/crossover/perturbation/repair is performed here; evaluation and selection are covered in Part 2/2.`

Algorithm 2: RDEx for Constrained Multi-Objective Optimization (Part 2/2)

Input: trial vectors $\{\mathbf{u}_{i,g}\}$ from Part 1/2 and parents $\{\mathbf{x}_{i,g}\}$, current archive A, elite-rate β_g
Output: next population $\{\mathbf{x}_{i,g+1}\}$, updated archive A, elite-rate β_{g+1}
for $i = 1..N$ **do**
  evaluate $\mathbf{f}(\mathbf{u}_{i,g})$ and $v(\mathbf{u}_{i,g})$
  `// Constrained parent-vs-child selection`
  if *both infeasible* **then**
    keep individual with smaller $v(\cdot)$
  else
    if *only one feasible* **then**
      keep the feasible
    else
      if $\mathbf{u}_{i,g} \prec \mathbf{x}_{i,g}$ **then**
        keep $\mathbf{u}_{i,g}$
      else if $\mathbf{x}_{i,g} \prec \mathbf{u}_{i,g}$ **then**
        keep $\mathbf{x}_{i,g}$
      else
        keep one with larger crowding distance
`// Archive and environmental selection`
$A \leftarrow A \cup \{$feasible nondominated in current pop.$\}$; truncate by crowding if needed
perform constrained nondominated sorting on population; refill next generation by feasible layers first, then by least violation
`// Effectiveness-driven update of` β
collect improved cases (dominance gain or violation drop $\Delta v > 0$), and those improved via elite-guided operator
if *any improvement* **then**
  $\beta_{g+1} \leftarrow \dfrac{\sum_{\text{elite-used \& improved}} \text{gain}}{\sum_{\text{all improved}} \text{gain} + \epsilon}$; clip $\beta_{g+1} \in [0, 1]$
else
  $\beta_{g+1} \leftarrow \beta_g$
return next population, archive A, and β_{g+1}

TABLE I
RESULT.

	F1	F2	F3	F4	F5	F6	F7	F8
best	2.65E-02	7.73E-03	5.34E-01	6.15E-01	8.43E+00	1.06E+01	4.74E-02	4.77E-01
mean	4.03E-01	2.15E-02	9.48E-01	1.01E+00	4.62E+01	1.17E+02	1.34E-01	7.33E-01
std	1.89E-01	1.05E-02	1.19E-01	1.46E-01	2.33E+01	1.56E+02	6.79E-02	4.84E-02
worst	4.99E-01	4.05E-02	1.05E+00	1.12E+00	8.69E+01	4.97E+02	2.72E-01	7.42E-01
LCV	0	0	0	0	0	0	0	0
	F9	F10	F11	F12	F13	F14	F15	
best	1.65E+00	1.82E+00	8.65E-01	8.05E-01	3.19E+00	5.43E+00	6.48E-02	
mean	1.39E+01	5.45E+00	4.60E+00	8.05E-01	6.79E+00	NaN	7.04E-02	
std	8.85E+00	2.13E+00	2.65E+00	5.04E-06	1.82E+00	NaN	3.13E-03	
worst	3.10E+01	8.16E+00	8.47E+00	8.05E-01	9.50E+00	NaN	7.61E-02	
LCV	0	0	0	0	0	0	0	