AIMS Mathematics, 11(1): 734–766.
DOI:10.3934/math.2026032
Received: 06 October 2025
Revised: 22 December 2025
Accepted: 25 December 2025
Published: 09 January 2026

https://www.aimspress.com/journal/Math

Research article

Adaptive dimension-wise Cauchy perturbation for enhanced differential evolution optimization

Tae Jong Choi[1] **and Yeji An**[2,*]

[1] Graduate School of Data Science, Chonnam National University, Gwangju 61186, Republic of Korea

[2] Department of Lifelong Education, Kyungil University, Gyeongsangbuk-do 38428, Republic of Korea

* **Correspondence:** Email: yejijoyan@kiu.kr; Tel: +82-53-600-5709; Fax: +82-53-600-5709.

Abstract: We propose an adaptive dimension-wise Cauchy perturbation mechanism to enhance the performance of differential evolution (DE). While traditional Cauchy perturbation improves solution diversity, it uses a fixed jumping rate and fails to address dimension-specific premature convergence. To overcome these limitations, the proposed method dynamically estimates the convergence level of each dimension in every generation and adaptively adjusts the jumping rate accordingly. This dimension-specific adaptive perturbation, applied during the crossover phase, mitigates premature convergence and strengthens the algorithm's ability to locate high-quality solutions. The proposed method was embedded into the Linear population size reduction Success RaTe-based Differential Evolution (L-SRTDE) algorithm, winner of the Institute of Electrical and Electronics Engineers Congress on Evolutionary Computation (IEEE CEC) 2024 competition. Extensive experiments on challenging benchmark optimization problems from the IEEE CEC 2017 test suite demonstrate that our method significantly outperforms the original L-SRTDE and several state-of-the-art DE variants in both convergence speed and solution accuracy.

Keywords: artificial intelligence; evolutionary algorithms; evolutionary computation; differential evolution; mathematical optimization
Mathematics Subject Classification: 68T01, 68W50

1. Introduction

Differential evolution (DE) [1] is a widely adopted evolutionary algorithm (EA) known for its effectiveness in solving continuous optimization problems. DE maintains a population of candidate solutions and iteratively improves them by generating new candidates through simple combinational operations. The best-performing candidates are selected and retained for subsequent generations.

An essential factor influencing the performance of EAs is the balance between exploration and exploitation. Matej et al. [2] defined exploration as "the process of visiting entirely new regions of a search space" and exploitation as "the process of visiting regions within the neighborhood of previously visited points". An imbalance in favor of exploration could cause the algorithm to insufficiently leverage promising solutions [3]. Conversely, excessive exploitation may reduce the probability of discovering the global optimum by restricting diversity [3].

A significant limitation of DE and other EAs is premature convergence [4], which occurs when excessive exploitation traps the algorithm in suboptimal solutions, significantly reducing exploration capabilities and thus restricting the potential to locate global optima, limiting algorithm performance.

A notable strategy to address premature convergence is the use of Cauchy perturbation [5]. This approach perturbs the target vector using a Cauchy distribution with a fixed jumping rate. The distribution's long-tailed property promotes diversity among trial vectors, enhancing the ability of DE to escape local optima. Cauchy perturbation has been effectively integrated into numerous DE variants, including representative approaches such as Best-Worst individuals driven multiple-layered Differential Evolution (BWDE) [6], Dynamic Population Structures-based Differential Evolution (DPSDE) [7], and Fractional Order Differential Evolution (FODE) [8]. However, the conventional Cauchy perturbation method faces two critical limitations:

- The method does not consider dimension-specific levels of premature convergence.
- The method employs a fixed jumping rate, reducing adaptability and potentially compromising performance.

To overcome these issues, this study introduces an adaptive, dimension-wise Cauchy perturbation mechanism. Our proposed method calculates convergence levels for each dimension at every generation, dynamically adjusting the jumping rates accordingly. These adaptive rates then guide the perturbation during crossover, enabling targeted and efficient exploration of the search space. We integrated this mechanism into the Linear population size reduction Success RaTe-based Differential Evolution (L-SRTDE) algorithm [9], the winner of the Institute of Electrical and Electronics Engineers Congress on Evolutionary Computation (IEEE CEC) 2024 competition. Extensive experiments on 29 benchmark problems [10] demonstrate that our approach significantly outperforms both the original L-SRTDE and several state-of-the-art DE variants in terms of convergence speed and solution accuracy.

In summary, the main contributions of this study are as follows:

- We analyze the limitations of traditional Cauchy perturbation in DE, showing that the existing approach typically (i) uses a single global jumping rate and (ii) ignores the heterogeneous convergence behavior of individual dimensions, which can lead to a waste of function evaluations.
- We propose an adaptive dimension-wise Cauchy perturbation (ADCP) mechanism that computes a convergence level for each dimension in every generation and maps it to a dimension-specific jumping rate via an exponential scaling function.
- We integrate ADCP into the L-SRTDE algorithm and conduct an extensive experimental study on the 29 functions of the IEEE CEC 2017 benchmark suite in 30, 50, and 100 dimensions, comparing the proposed algorithm with 11 recent and competitive DE variants.

The rest of this paper is organized as follows: Section 2 introduces preliminary concepts. In Section 3, we review relevant literature, with a particular focus on Linear population size reduction Success-History Adaptation Differential Evolution (L-SHADE) variants. Section 4 describes the proposed

algorithm in detail. Section 5 presents the experimental setup and discusses the results. Finally, Section 6 concludes the paper.

2. Preliminaries

2.1. Differential evolution

DE [1] is a population-based evolutionary algorithm widely recognized for its simplicity, robustness, and efficiency in solving continuous optimization problems. It maintains a population of candidate solutions, each represented as a real-valued vector in a multidimensional search space, and iteratively evolves these solutions through its mutation, crossover, and selection operators.

During mutation, DE generates a mutant vector by perturbing existing solutions through vector differences. The classic mutation strategy DE/rand/1 constructs mutant vectors by combining randomly selected individuals from the current population

$$\vec{v}_i^{\,g} = \vec{x}_{r1}^{\,g} + F \cdot (\vec{x}_{r2}^{\,g} - \vec{x}_{r3}^{\,g}), \tag{2.1}$$

where $\vec{x}_{r1}^{\,g}$, $\vec{x}_{r2}^{\,g}$, $\vec{x}_{r3}^{\,g}$ are distinct randomly chosen vectors, and F denotes the scaling factor controlling the perturbation amplitude.

Following mutation, the crossover operator blends each mutant vector with its corresponding candidate solution to produce a trial vector. The most common method is the binomial crossover, defined as

$$u_{i,j}^{g} = \begin{cases} v_{i,j}^{g}, & \text{if } rand_{i,j} \le CR \text{ or } j = j_{rand}, \\ x_{i,j}^{g}, & \text{otherwise}, \end{cases} \tag{2.2}$$

where CR is the crossover probability, and j_{rand} ensures that at least one parameter from the mutant vector is inherited. Here, $v_{i,j}^{g}$ and $x_{i,j}^{g}$ denote the jth components of $\vec{v}i^{g}$ and $\vec{x}i^{g}$, respectively, and $rand_{i,j} \sim U(0, 1)$.

Finally, the selection operator compares the fitness values of each trial vector and its corresponding candidate solution, retaining the solution with equal or better fitness for the next generation

$$\vec{x}_i^{\,g+1} = \begin{cases} \vec{u}_i^{\,g}, & \text{if } f(\vec{u}_i^{\,g}) \le f(\vec{x}_i^{\,g}), \\ \vec{x}_i^{\,g}, & \text{otherwise}, \end{cases} \tag{2.3}$$

where $f(\vec{\cdot})$ represents the objective function being minimized.

This evolutionary process continues until a predefined termination criterion is met, converging toward optimal or near-optimal solutions for complex optimization tasks. The DE algorithm is highly flexible for optimizing continuous problems and can integrate various techniques to enhance performance, such as adaptive parameter control [11], population size control [12], advanced mutation strategies [13], opposition-based learning [14], eigenvector-based crossover [15], and hybridization with chaotic neural networks and quality-diversity frameworks [16, 17]. Moreover, DE has been successfully applied to training neural networks [18] and multi-objective feature selection [19]. For a more comprehensive description of recent advancements in DE, please refer to survey papers [20–22].

2.2. *Analysis of Cauchy distribution*

The Cauchy distribution is a continuous probability distribution with a distinctively sharp peak and notably heavy tails compared to the Gaussian and other common distributions. It is defined by two parameters: the location parameter x_0, which specifies the distribution's center, and the scale parameter γ, which determines the peak's height and spread. A larger γ produces a lower peak and broader spread, whereas a smaller γ yields a taller, narrower peak.

Mathematically, the probability density function (PDF) is

$$f(x; x_0, \gamma) = \frac{1}{\pi\gamma\left[1 + \left(\frac{x-x_0}{\gamma}\right)^2\right]} = \frac{1}{\pi}\left[\frac{\gamma}{(x-x_0)^2 + \gamma^2}\right]. \tag{2.4}$$

The cumulative distribution function (CDF) is

$$F(x; x_0, \gamma) = \frac{1}{\pi}\arctan\left(\frac{x-x_0}{\gamma}\right) + \frac{1}{2}. \tag{2.5}$$

The most distinctive feature of the Cauchy distribution is its heavy-tailed nature, which increases the probability of generating extreme values. This makes it particularly effective in stochastic optimization processes such as perturbation mechanisms in evolutionary algorithms by enabling broader exploration of the search space and improving the chances of escaping local optima [23].

Figure 1 illustrates the PDF of the Cauchy distribution for several γ values, highlighting its effect on the distribution's shape and spread.

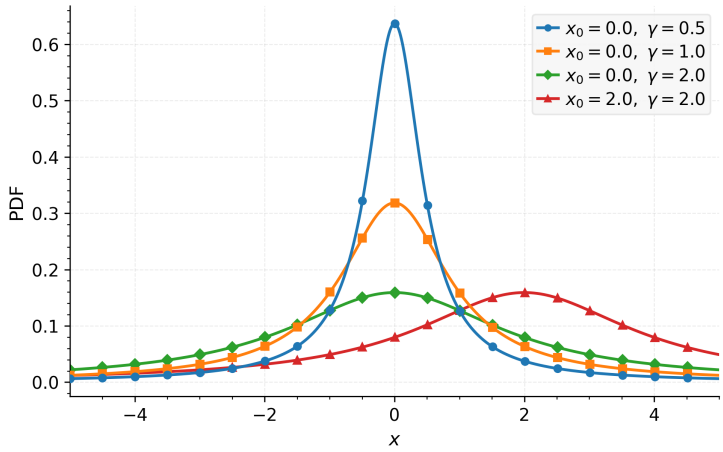


Figure 1. Probability density function of the Cauchy distribution for several γ values.

3. Literature review

DE has been extensively refined over the last decade, mainly by introducing adaptive mechanisms that balance exploration and exploitation more effectively. Among these contributions, the L-SHADE framework has become a central baseline: many recent state-of-the-art algorithms can be interpreted as L-SHADE variants that enhance parameter control, population management, or hybrid search strategies. In this section, we organize the literature around four main directions: success-history-based

parameter adaptation, population-size and selective-pressure mechanisms, dual- or multi-population and ensemble schemes, and success-rate-based control. We then position the proposed method within this landscape.

3.1. Success-history–based adaptation in L-SHADE

JADE [24] is a seminal adaptive DE variant that introduced the widely used `DE/current-to-`p`best/1` mutation. Each target vector is perturbed toward one of the top p-percent best individuals while also adding a scaled difference between two randomly selected individuals. This mutation simultaneously exploits good solutions and maintains diversity. JADE also keeps an external archive of recently replaced parents and occasionally uses differences involving these archived vectors to further diversify the search. Control parameters F and CR are generated from Cauchy/normal distributions whose means are updated from successful offspring, so the algorithm gradually steers parameter values toward problem-dependent effective regions.

SHADE [25] generalizes JADE's adaptation scheme by storing a memory of successful parameter values rather than a single running mean. It keeps H pairs of historical (F, CR) values and, for each trial, samples from a distribution centered at one randomly chosen memory entry. After each generation, the memory is updated with weighted averages of the F and CR values that produced successful offspring. This success-history mechanism improves robustness and reduces the risk that one bad update permanently misguides the parameter means.

L-SHADE [26] combines SHADE's success-history adaptation with linear population size reduction (LPSR). The algorithm starts with a relatively large population to support broad exploration and then gradually shrinks the population in a linear fashion as the run progresses, pruning the worst individuals. This simple deterministic schedule typically accelerates convergence while still preserving early diversity, and L-SHADE has therefore become a standard reference in DE competitions and subsequent research.

3.2. Population control and selective pressure in L-SHADE variants

Several L-SHADE variants refine how the population is resized and how selection pressure is imposed. The jSO algorithm [27] builds directly on L-SHADE and improved L-SHADE (iL-SHADE). It keeps the `current-to-`p`best/1` mutation but modifies the way F and CR are sampled over time. Large CR values are deliberately propagated in early generations, while restrictions on F are used to avoid overly aggressive steps in the initial phase. As the number of function evaluations increases, jSO gradually shifts to more exploitative settings. Combined with L-SHADE's population reduction, these stage-dependent rules yield a very competitive DE on the CEC 2017 benchmark.

L-SHADE with Rank-based Selective Pressure (LSHADE-RSP) [28] augments L-SHADE with rank-based selective pressure. Individuals are sorted by fitness, and the probability that a candidate is selected as a partner in mutation depends on its rank. The modified `current-to-`p`best` strategy therefore uses fitter individuals more often when forming difference vectors, biasing search directions toward promising regions while still allowing weaker individuals to contribute. To further control the search dynamics, LSHADE-RSP also couples L-SHADE-style parameter adaptation with jSO-inspired rules on F and CR, including a weighted scaling factor and time-varying p for the $p$best set.

Non-Linear SHADE with Rank-based Selective Pressure (NL-SHADE-RSP) [29] extends LSHADE-RSP in two ways. First, it replaces linear population size reduction with a non-linear

schedule, so the rate of population shrinkage can be slower or faster at different stages of the run. Second, it refines archive usage and crossover control: the algorithm adaptively adjusts the probability of drawing partners from the archive, and it sorts individuals by crossover rates when updating the success-history memory. These modifications aim to maintain an effective balance between exploration and exploitation throughout the run.

Non-Linear SHADE with Linear Bias Change (NL-SHADE-LBC) [30] focuses more directly on parameter adaptation. It uses a generalized Lehmer mean with a linearly changing bias when updating the historical (F, CR) memories. At early stages it biases F toward relatively large values to encourage exploration, then gradually shifts toward more moderate values as the search progresses. A similar bias is applied to CR so that recombination becomes more conservative or more aggressive depending on the phase of the search. Together with non-linear population reduction and improved archive management, this linear-bias change further stabilizes and strengthens the self-adaptation process.

3.3. Dual-population and ensemble L-SHADE extensions

Another line of work restructures the population or enriches the set of search operators. Linear population size reduction Newest and Top Adaptive Differential Evolution (L-NTADE) [31] introduces a dual-population scheme consisting of a "newest" population and a "top" population. The newest population stores the most recently generated individuals, whereas the top population maintains the best solutions found so far. A variant of `current-to-`p`best` mutation combines information from both populations, and successful offspring are inserted into the newest population immediately so that improvements can influence subsequent offspring within the same generation. L-NTADE also uses SHADE-style parameter adaptation and linear population reduction. This design emulates an effectively unbounded population while keeping the actual population size finite, improving diversity and reducing stagnation on difficult multimodal problems.

Multi-operator ensemble LSHADE with Restart and Local search (mLSHADE-RL) [32] represents a more hybrid, ensemble-style extension of LSHADE with ensemble sinusoidal differential covariance matrix adaptation with Euclidean neighborhood (LSHADE-cnEpSin). It employs several mutation strategies (including weighted `current-to-`p`best` and ordered-$p$best variants) and assigns them adaptively according to their recent success. The algorithm also includes a restart mechanism: when stagnation is detected, part of the population is reinitialized or diversified to escape local optima. In the later stages, mLSHADE-RL invokes a local search procedure around high-quality solutions to intensify exploitation. This combination of multi-operator search, restart, local search, and success-history adaptation makes mLSHADE-RL a representative example of highly hybridized L-SHADE descendants.

3.4. Success-rate–based parameter control

While success-history–based schemes infer good parameter values from the distribution of successful offspring, another family of methods uses the *success rate* itself as a feedback signal. L-SRTDE [9] is a recent representative of this class. L-SRTDE starts from the dual-population design of L-NTADE (newest and top populations) but replaces the success-history adaptation of F with a success-rate–driven update. In each generation, the algorithm computes the fraction SR of offspring that successfully replace their parents and then sets the mean scaling factor m_F via a smooth function of SR (for example, a tanh-shaped curve). Individual F values are sampled from a narrow normal

distribution around m_F. In addition, the greediness parameter p_b that determines the size of the top-p_b elite set is also expressed as a function of SR, shrinking toward very small values when the algorithm is already making frequent improvements. In this way, L-SRTDE automatically shifts from exploratory behavior (moderate F, large p_b) to strongly exploitative behavior (larger F, very small p_b) as the search becomes more successful, while still using L-SHADE's linear population size reduction. Empirical evaluations on several CEC benchmark suites show that this success-rate feedback yields performance comparable to or better than many strong L-SHADE variants.

3.5. Summary

To summarize, modern DE research has largely evolved within the L-SHADE framework. Success-history–based methods such as SHADE and L-SHADE provide robust automatic tuning of F and CR. Subsequent variants refine population size reduction and selective pressure mechanisms (jSO, LSHADE-RSP, NL-SHADE-RSP, NL-SHADE-LBC) introduce dual-population designs and ensemble search operators (L-NTADE, mLSHADE-RL) and exploit global performance signals such as success rate (L-SRTDE).

4. Proposed algorithm

This section introduces the adaptive dimension-wise Cauchy perturbation (ADCP) method and its integration into the L-SRTDE algorithm [9]. We first review the traditional Cauchy perturbation strategy [5] and highlight its limitations. We then present the ADCP method, which dynamically adjusts perturbation strengths according to dimension-specific convergence levels. Finally, we describe how ADCP is integrated into the L-SRTDE algorithm [9], detailing the initialization, mutation, crossover, and selection phases.

4.1. Review of Cauchy perturbation

Cauchy perturbation [5] is a strategy for enhancing the exploration capability of DE algorithms by exploiting the long-tailed property of the Cauchy distribution. Compared with other probability distributions, such as the Gaussian, the Cauchy distribution's heavier tail facilitates occasional large jumps in the search space, enabling algorithms to escape local optima and conduct a more diverse search [33]. Unlike Cauchy mutation [34], which perturbs a mutant vector, Cauchy perturbation modifies a target vector during the crossover operation. This technique has been successfully applied to DE algorithms, including representative approaches such as BWDE [6], DPSDE [7], and FODE [8], significantly improving their performance.

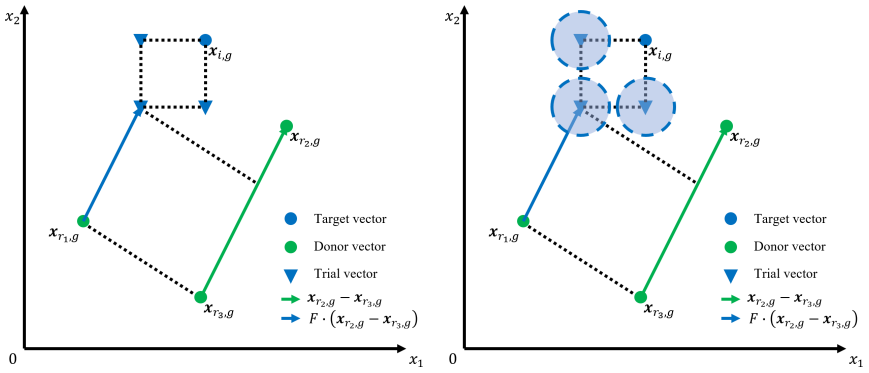
The Cauchy perturbation approach alternates between two recombination operators according to a jumping rate (JR). When generating a trial vector, the algorithm uses the original operator if a randomly generated number exceeds JR; otherwise, it employs the modified operator

$$u_{i,j}^{g} = \begin{cases} v_{i,j}^{g}, & \text{if } \mathrm{rand}_{i,j}^{(1)} \le CR \text{ or } j = j_{\mathrm{rand}}, \\ \mathrm{rndc}\left(x_{i,j}^{g}, 0.1\right), & \text{if } \mathrm{rand}_{i,j}^{(1)} > CR \text{ and } \mathrm{rand}_{i}^{(2)} \le JR, \\ x_{i,j}^{g}, & \text{otherwise.} \end{cases} \tag{4.1}$$

While traditional Cauchy perturbation improves exploration and robustness, its reliance on a fixed

jumping rate limits adaptability during premature convergence. This can lead to either excessive randomness or insufficient exploration. Overcoming this limitation is the motivation for the adaptive dimension-wise Cauchy perturbation (ADCP) method proposed in this paper.

Figure 2 illustrates the influence of the Cauchy perturbation operator on the offspring distribution.



(a) DE/rand/1/bin without Cauchy perturbation (b) DE/rand/1/bin with Cauchy perturbation

Figure 2. Cauchy perturbation can enlarge the exploration capability by perturbing the target vector, thus significantly expanding the feasible region accessible to the trial vectors.

4.2. Adaptive dimension-wise Cauchy perturbation (ADCP)

Traditional Cauchy perturbation applies a uniform jumping rate across all dimensions regardless of their diversity levels. However, premature convergence can vary significantly between dimensions; some may already have sufficient exploration, and others require stronger perturbation to escape local optima. Applying the same jumping rate to all dimensions may therefore degrade performance by introducing unnecessary exploration in dimensions that already maintain high diversity.

To overcome this limitation, we propose the adaptive dimension-wise Cauchy perturbation (ADCP) method. ADCP dynamically adjusts the jumping rate for each dimension based on its convergence level (CL_j^g), defined as

$$CL_j^g = 1 - \frac{\sigma_j^g}{\sigma_j^0}, \tag{4.2}$$

where σ_j^g and σ_j^0 are the current and initial standard deviations of the population in dimension j, respectively. A higher CL_j^g indicates stronger convergence (lower diversity), signaling stagnation and the need for increased exploration.

The dimension-specific jumping rate JR_j is updated using an exponential scaling function,

$$JR_j = JR_{min} + (JR_{max} - JR_{min}) \cdot \frac{e^{k \cdot CL_j^g} - 1}{e^k - 1}, \tag{4.3}$$

where JR_{min} and JR_{max} are the predefined minimum and maximum jumping rates, and $k = 10$ controls the steepness of the exponential growth. This formulation increases JR_j for highly converged dimensions, encouraging exploration while keeping JR_j lower for diverse dimensions to preserve existing variability.

The modified recombination operation applies the adaptive jumping rate per dimension:

$$u_{i,j}^{g} = \begin{cases} v_{i,j}^{g}, & \text{if } \text{rand}_{i,j}^{(1)} \leq CR \text{or } j = j_{\text{rand}}, \\ \text{rndc}\left(x_{i,j}^{g}, 0.1\right), & \text{if } \text{rand}_{i,j}^{(1)} > CR \text{ and } \text{rand}_{i,j}^{(2)} \leq JR_j, \\ x_{i,j}^{g}, & \text{otherwise.} \end{cases} \tag{4.4}$$

By tailoring perturbation strength dimension-wise, ADCP adapts to the evolving population distribution, improving the algorithm's ability to overcome premature convergence without disrupting dimensions that already exhibit sufficient diversity.

At generation g, σ_j^g, CL_j^g, and JR_j are computed once using the top population $\vec{x}^{top}$, and the resulting JR_j values are reused for all $i = 1, 2, ..., N_g$ within that generation. In both the traditional Cauchy perturbation and ADCP, we fix the Cauchy scale parameter to 0.1. This value is widely used in DE variants employing Cauchy perturbation: larger values tend to induce overly disruptive jumps, whereas smaller values concentrate the perturbation too strongly and reduce the practical benefit of heavy-tailed exploration.

4.3. Enhanced L-SRTDE with ADCP

4.3.1. Initialization

L-SRTDE begins by initializing a population of candidate solutions uniformly at random within the search domain. Let N_{max} denote the initial population size. Each individual $\vec{x}_i^{new}$ for $i = 1, 2, ..., N_{max}$ is a D-dimensional vector of decision variables sampled between the lower and upper bounds of the problem. An identical copy of this population is stored as the top population $\vec{x}^{top}$; initially $\vec{x}^{top} = \vec{x}^{new}$ for all i and thus contains the same N_{max} solutions. All individuals are evaluated on the objective function $f(\cdot)$ to obtain their fitness values. Along with the main population, L-SRTDE maintains a small memory of size H for certain strategy parameters (inherited from SHADE). For instance, the memory entries $M_{Cr,k}$ are used to adapt the crossover rate as described later. Additionally, the enhanced L-SRTDE algorithm initializes the standard deviations of the population (σ_j^0) for each dimension j.

As the evolutionary process progresses, L-SRTDE employs linear population size reduction (LPSR) to gradually shrink the population from N_{max} to a predetermined minimum size N_{min} by the end of the run. This is achieved by removing the worst individuals at certain intervals so that the population decays roughly linearly over generations. For example, if FEs_{max} is the maximum number of function evaluations, the allowed population size at a given point can be recalculated as

$$N_{allow} = \left\lceil N_{min} + (N_{max} - N_{min}) \cdot \frac{FEs_{max} - FEs_{current}}{FEs_{max}} \right\rceil, \tag{4.5}$$

and any excess individuals are pruned from the population (based on worst fitness) to match this size. In this way, the algorithm allocates more search agents in early stages for exploration and gradually focuses resources as it nears termination. Throughout the run, L-SRTDE also tracks the success rate (SR) of the algorithm on a per-generation basis. The success rate SR is defined as the fraction of population members that were improved by offspring in a generation, that is, $SR = \frac{Ns}{N}$, where Ns is the number of successful replacements out of the current population of size N. This success rate is a key driver for the adaptive strategies described below.

4.3.2. Mutation

The mutation step remains unchanged in the enhanced L-SRTDE algorithm. Each generation of L-SRTDE produces offspring via a tailored differential mutation strategy called r-new-to-ptop/n/t. This strategy operates on two synchronized populations: the current population $\vec{x}^{new}$ of size N and the elite population $\vec{x}^{top}$ (also of size N) that stores the best solutions found so far. For each target individual $\vec{x}_i^{new}$ (where $i = 1, 2, ..., N$), a mutant vector $\vec{v}_i$ is generated by mixing information from randomly chosen individuals in both populations. In particular, L-SRTDE randomly selects one index r_1 from $\{1, ..., N\}$ to serve as a base from the new population and another index $r_3 \in \{1, ..., N\}$ to pick a vector from the top population. It also selects a third index r_2 from the new population using a rank-based selection (biased toward better-ranked individuals, with a pressure parameter k_p). Additionally, a "greediness" parameter $p_b \in (0, 1]$ controls the choice of an elite target: an index *pbest* is uniformly sampled from the top $100p_b\%$ fraction of the $\vec{x}^{top}$ population. Ensuring all chosen indices are distinct, the mutant vector is then computed as

$$\vec{v}_i = \vec{x}_{r_1}^{new} + F \cdot (\vec{x}_{pbest}^{top} - \vec{x}_i^{new}) + F \cdot (\vec{x}_{r_2}^{new} - \vec{x}_{r_3}^{top}). \tag{4.6}$$

This mutation formula combines elements from the new population and the top population. The first difference term $\vec{x}_{pbest}^{top} - \vec{x}_i^{new}$ drives the mutant toward one of the current elite solutions (introducing an exploitative bias), while the second term $\vec{x}_{r_2}^{new} - \vec{x}_{r_3}^{top}$ injects diversity by adding a scaled difference between a random individual from the new population and a random elite individual.

Adaptive scaling factor (F) In L-SRTDE, the differential weight F for each trial is dynamically adapted each generation based on the observed success rate. At the beginning of generation g, a target mean m_F is calculated as a sigmoid-shaped function of the previous generation's success rate:

$$m_F = 0.4 + 0.25 \cdot tanh(5 \cdot SR). \tag{4.7}$$

Here, $SR \in [0, 1]$ is the ratio of successful offspring in the last generation. This formula was obtained through a hyper-heuristic search and reflects that when the algorithm is performing well (high SR), a slightly higher average F is beneficial, whereas when progress is slow (low SR), a more moderate F is preferred. For each mutant vector, an actual scaling factor F is then sampled from a narrow normal distribution around this mean: $F \sim \mathcal{N}(m_F, 0.02^2)$. This self-adaptation mechanism causes F to increase or decrease in response to the algorithm's recent success: a higher success rate yields a larger F on average (potentially allowing bigger exploratory jumps), whereas a low success rate keeps F closer to a conservative baseline to maintain stability.

Adaptive elite fraction (p_b) The parameter p_b determining the fraction of top individuals eligible for selection as *pbest* is likewise adjusted each generation according to the success rate. L-SRTDE uses an exponential decay relationship,

$$p_b = 0.7 \cdot exp(-7 \cdot SR). \tag{4.8}$$

When the SR is very low, p_b approaches 0.7, meaning the *pbest* individual is chosen from roughly the top 70% of $\vec{x}^{top}$, essentially a wide selection, close to random among elites. This promotes exploration

when improvements are scarce. However, as the algorithm starts succeeding more frequently, SR grows and p_b shrinks exponentially. For example, if $SR = 0.5$, then $p_b \approx 0.7e^{-3.5} \approx 0.03$, focusing on only the top 3% of solutions, effectively forcing selection of the absolute best individuals. By tightening the elite group as success improves, the mutation strategy seamlessly transitions from exploration to intense exploitation of the best solutions discovered.

4.4. Crossover

After mutation, each mutant vector $\vec{v}_i$ is recombined with the current target $\vec{x}_i^{new}$ through a crossover operator to produce a trial vector $\vec{u}_i$. The enhanced L-SRTDE algorithm calculates the convergence level (CL_j^g) for each dimension j and subsequently updates the jumping rate (JR_j) according to Eqs (4.2) and (4.3). Subsequently, the algorithm employs the ADCP method according to Eq (4.4). The crossover rate Cr is adapted between generations using a success-history based mechanism but with a critical modification to use the effective crossover rather than the nominal rate. Specifically, once $\vec{u}_i$ is formed, the algorithm computes the realized crossover rate Cr_a as the fraction of components where $\vec{u}_i$ inherited genes from the mutant $\vec{v}_i$. The algorithm uses this Cr_a value for learning and adjustment: if the trial $\vec{u}_i$ turns out to be successful (i.e., it yields an improvement in selection), then Cr_a is recorded into a set of successful crossover rates for that generation.

At the end of each generation, if there were any successful trials, the algorithm updates the stored crossover-rate memory. Typically, this involves computing a weighted average of all recorded Cr_a values and then updating one of the memory entries M_{Cr} (cycling through the H entries) with this average. In implementation, new trial crossover rates are sampled from a normal distribution around a randomly chosen memory mean: $Cr \sim \mathcal{N}(M_{Cr,h}, 0.05^2)$ for some index $h \in \{1, ..., H\}$. By using a relatively small standard deviation (0.05) for this sampling, the algorithm limits drastic fluctuations in crossover rate, enabling a more stable search. This crossover scheme, together with the memory mechanism, allows the algorithm to self-tune the balance between exploration (lower Cr tends to retain more parent genes) and recombination (higher Cr produces more mixed offspring) based on what has historically led to improvements.

4.5. Selection

The selection step remains unchanged in the enhanced L-SRTDE algorithm. L-SRTDE's selection compares the trial to the mutation base vector and uses a rotating index for replacement. For each trial vector $\vec{u}_i$ generated (with associated base index r_1), the algorithm identifies the fitness of the base individual $\vec{x}_{r_1}^{new}$. If the trial is better or equal in fitness, then $\vec{u}_i$ is considered a successful offspring. Rather than replacing r_1 or the original target i, the algorithm inserts this successful offspring into the population at a position indicated by a running index nc. The value of nc starts at 1 and is incremented by 1 each time an insertion occurs; if $nc > N$, it wraps around to 1, cycling through population positions. Formally, the replacement can be described as

$$\vec{x}_{nc}^{new} = \begin{cases} \vec{u}_i, & \text{if } f(\vec{u}_i) \le f(\vec{x}_{r_1}^{new}), \\ \vec{x}_{nc}^{new}, & \text{otherwise,} \end{cases} \tag{4.9}$$

after which nc is advanced by 1 (modulo N). If the trial is not better than the base, no replacement is made for that trial.

Algorithm 1: Enhanced L-SRTDE with ADCP

Require: D, FEs_{max}, N_{max}, $f(\cdot)$, JR_{min}, JR_{max}, $k = 10$
Ensure: $\vec{x}^{top}_{best}$, $f(\vec{x}^{top}_{best})$

Initialize parameters: $N = N_{max}, N_{min} = 4, H = 5, M_{Cr,r} = 1$
Set initial $SR = 0.5$, memory index $k = 1$, generation $g = 0$, counters $nc = 1$, $kp = 3$
Initialize populations $(\vec{x}^{new}_1, ..., \vec{x}^{new}_N)$ randomly
Evaluate $f(\vec{x}^{new})$
Copy $\vec{x}^{new}$ to $\vec{x}^{top}$, $f(\vec{x}^{new})$ to $f(\vec{x}^{top})$
Compute initial standard deviations σ^0_j, $j = 1, ..., D$
while $FEs_{current} < FEs_{max}$ **do**
  $S_{Cr} = \emptyset$, $S_{\Delta f} = \emptyset$ {Initialize memory sets}
  Sort and assign ranks for $\vec{x}^{new}$ and $f(\vec{x}^{new})$
  Compute standard deviations σ^g_j from top solutions
  Compute convergence levels CL^g_j (Eq 4.2)
  Compute jumping rates JR_j (Eq 4.3)
  Set successful offspring counter: $Ns = 0$
  Set temporary storage index: $m = 1$
  for $i = 1$ to N^g **do**
    Generate $\vec{u}_i$ and actual crossover rate Cr_a by calling **ADCP_Recombination**$(\vec{x}^{new}, \vec{x}^{top}, CL^g_j, JR_j, SR)$
    Apply bound constraints and evaluate $f(\vec{u}_i)$
    if $f(\vec{u}_i) < f(\vec{x}^{new}_{r_1})$ **then**
      Store Cr_a into S_{Cr}
      Store $(f(\vec{x}^{new}_{r_1}) - f(\vec{u}_i))$ into $S_{\Delta f}$
      Store successful offspring: $\vec{x}^{new}_{nc} = \vec{u}_i$, $f(\vec{x}^{new}_{nc}) = f(\vec{u}_i)$
      Also store into temporary population: $\vec{x}^{temp}_m = \vec{u}_i$, increment m
      Increment successful offspring counter: $Ns = Ns + 1$
      Increment replacement index: $nc = mod(nc + 1, N^g)$
    end if
  end for
  Compute success rate: $SR = Ns/N^g$
  Update population size: $N^{g+1} = \lceil N_{min} + (N_{max} - N_{min}) \frac{FEs_{max} - FEs_{current}}{FEs_{max}} \rceil$
  Combine $\vec{x}^{top}$ and successful offspring from $\vec{x}^{temp}$, sort by fitness
  Retain best N^{g+1} individuals as updated $\vec{x}^{top}$
  if $N^g > N^{g+1}$ **then**
    Remove worst solutions from current population $\vec{x}^{new}$ accordingly
  end if
  Update crossover memory $M_{Cr,k}$ using S_{Cr} and $S_{\Delta f}$
  Increment indices: $k = mod(k + 1, H)$, generation $g = g + 1$
end while
return Best solution found $\vec{x}^{top}_{best}$ and its fitness $f(\vec{x}^{top}_{best})$

Algorithm 2: ADCP_Recombination (Mutation + Adaptive Crossover)

Require: Populations: $\vec{x}^{new}, \vec{x}^{top}$, convergence levels CL_j^g, jumping rates JR_j, current SR
Ensure: Trial vector $\vec{u}_i$, actual crossover rate Cr_a

Compute adaptive scaling factor mean: $mF = 0.4 + 0.25 \cdot tanh(5 \cdot SR)$
repeat
  $F_i = randn(mF, 0.02)$
until $F_i \in (0, 1)$
Choose memory index $r = randi[1, H]$
Sample and limit crossover rate: $Cr_i = randn(M_{Cr,r}, 0.05)$, set $Cr_i = min(1, max(0, Cr_i))$
Randomly select base index $r_1 = randi(1, N^g)$
Set elite fraction $pb = 0.7 \cdot e^{-7 \cdot SR}$
repeat
  Select elite solution index $pbest = randi(1, N^g \cdot pb)$
  Generate index r_2 via rank-based selection
  Randomly choose $r_3 = randi(1, N^g)$
until indices r_1, r_2, r_3, and $pbest$ differ
Compute mutant vector: $\vec{v}_i = \vec{x}_{r_1}^{new} + F_i(\vec{x}_{pbest}^{top} - \vec{x}_i^{new}) + F_i(\vec{x}_{r_2}^{new} - \vec{x}_{r_3}^{top})$
Randomly select $j_{rand} = randi(1, D)$
Set $count_{Cr} = 0$
for $j = 1$ to D **do**
  Generate random values $rand_{i,j}^{(1)} \sim U(0, 1)$ and $rand_{i,j}^{(2)} \sim U(0, 1)$
  if $rand_{i,j}^{(1)} \leq CR_i$ or $j = j_{rand}$ **then**
    $u_{i,j}^g = v_{i,j}^g$
    $count_{Cr} = count_{Cr} + 1$
  else if $rand_{i,j}^{(1)} > CR_i$ and $rand_{i,j}^{(2)} \leq JR_j$ **then**
    $u_{i,j}^g = rndc(x_{i,j}^g, 0.1)$ (Cauchy perturbation)
  else
    $u_{i,j}^g = x_{i,j}^g$
  end if
end for
Calculate actual crossover rate: $Cr_a = count_{Cr}/D$
return $\vec{u}_i^g, Cr_a$

Finally, L-SRTDE computes the success rate for the generation, $SR = Ns/N$. This value of SR is then used to update the adaptive parameters for the next generation. In particular, the formulas for m_F and p_b are applied using the newly computed SR so that the mutation in the next iteration will adjust to the algorithm's recent performance. The memory for crossover rate is also updated at this point: if any trials were successful, the stored mean(s) M_{Cr} are recalculated based on the collected set of successful Cr_a values, and the index for memory update is advanced (cyclically). The evolutionary loop then proceeds to the next generation (mutation, crossover, selection) with the reduced population size N' and tuned parameters. This process repeats until a termination condition is met. The pseudocode of the proposed algorithm is detailed in Algorithms 1 and 2.

5. Experimental study

5.1. Experimental setup

We evaluated the proposed algorithm through comparative experiments against 11 robust DE variants: L-SRTDE [9], BWDE [6], DPSDE [7], FODE [8], MultiSelection-based Differential Evolution (MSDE) [35], Reconstructed Differential Evolution (RDE) [36], Ring Sub-population architecture-based Differential Evolution (RSDE) [37], Serial Multilevel-Learned Differential Evolution (SMLDE) [38], NL-SHADE-LBC [30], NL-SHADE-RSP [29], and iLSHADE-RSP [5]. Among these, L-SRTDE [9], NL-SHADE-LBC [30], and NL-SHADE-RSP [29] are the most effective L-SHADE variants, having won the IEEE CEC competitions in 2024, 2022, and 2021, respectively. The remaining eight—BWDE [6], DPSDE [7], FODE [8], MSDE [35], RDE [36], RSDE [37], SMLDE [38], and iLSHADE-RSP [5]—are state-of-the-art DE methods that employ traditional Cauchy perturbation. Control parameters for each variant followed the settings recommended in their original publications. For the proposed algorithm, the minimum and maximum jumping rates were set to 0.1 and 0.2, respectively. Table 1 summarizes the parameter configurations for all algorithms.

Table 1. Parameter configurations for all algorithms.

Algorithm	Year	Parameter settings
L-SRTDE-ADCP	—	$N_{init} = 20D, N_{min} = 4, H = 5, k = 1, nc = 1, kp = 3, M_{Cr,r} = 1, SR = 0.5, JR_{min} = 0.1, JR_{max} = 0.2$
L-SRTDE	2024	$N_{init} = 20D, N_{min} = 4, H = 5, k = 1, nc = 1, kp = 3, M_{Cr,r} = 1, SR = 0.5$
NL-SHADE-LBC	2022	$N_{init} = 23D, N_{min} = 4, H = 20D, k = 1, NA = 1.0NP, n_A = 0.5, M_{F,r} = 0.5, M_{Cr,r} = 0.9$
NL-SHADE-RSP	2021	$N_{init} = 30D, N_{min} = 4, H = 20D, k = 1, NA = 2.1NP, n_A = 0.5, M_{F,r} = 0.2, M_{Cr,r} = 0.2$
BWDE	2024	$N_{init} = 18D, N_{min} = 4, k = 3, p = 0.11, P_j = 0.2, q = 0.3, u = 0.5$
DPSDE	2024	$N_{init} = 18D, N_{min} = 4, k = 30, \Delta = 0.3$
FODE	2025	$N_{init} = 18D, N_{min} = 4, k = 3, p = 0.11, P_j = 0.2, \alpha = 0.995, r = 6$
MSDE	2024	$N_{init} = 18D, N_{min} = 4, H = 5, k = 3, p = 0.11, P_j = 0.2, NA = 1.0NP, \theta_1 = 10, \theta_2 = 25$
RDE	2024	$N_{init} = 18D, N_{min} = 4, H = 5, k = 3, p = 0.25, P_j = 0.2, NA = 1.0NP, \mu_F = 0.3, \mu_{Cr} = 0.8, \gamma_1 = 0.5, \gamma_2 = 0.5$
RSDE	2024	$N_{init} = 18D, N_{min} = 4, k = 3, p = 0.11, P_j = 0.2, \eta = 0.2, \gamma = 0.15, \kappa = 0.21$
SMLDE	2024	$N_{init} = 75D^{\frac{2}{3}}, N_{min} = 4, k = 3, p = 0.11, P_j = 0.2, \zeta = 0.5$
iLSHADE-RSP	2021	$N_{init} = 75D^{\frac{2}{3}}, N_{min} = 4, k = 3, p = [0.085, 0.17], P_j = 0.2$

The experiments used the IEEE CEC 2017 test suite [10], a widely adopted benchmark set for evaluating optimization algorithms. The suite contains two unimodal functions, seven basic multimodal functions, ten expanded multimodal functions, and ten hybrid composition functions. Functions with a single global optimum and no local optima are classified as unimodal, whereas functions with multiple local optima are classified as multimodal.

Each function was run independently 51 times, with the maximum number of function evaluations defined as $FES_{max} = 10000 \cdot D$. Performance was measured using the mean error and standard deviation across the 51 runs. Errors below 10^{-8} were recorded as zero for consistency. The best result for each comparison is highlighted in bold. Statistical significance between two algorithms was tested using the Wilcoxon rank-sum test [39], and comparisons among multiple algorithms employed the Friedman test [40].

All experiments were conducted on an Ubuntu 20.04.5 LTS system with an AMD Ryzen Threadripper 2990WX CPU and 64GB RAM. The algorithms were implemented in C++ and compiled using GCC.

5.2. Comparison with state-of-the-art DE variants

The proposed algorithm was compared with 11 well-known DE variants on 29 CEC 2017 benchmark functions to assess its overall performance. Detailed numerical results are provided in the supplementary material. For each dimensionality setting, the Wilcoxon rank-sum test [39] was used to assess pairwise performance differences, and the Friedman test [40] was employed to rank all algorithms. Post-hoc analyses using the Bonferroni–Dunn [41], Holm [42], and Hochberg [43] procedures were then performed to verify statistical significance.

- **30-dimensional problems:** As shown in Table 2, the proposed algorithm outperforms the competing methods on most benchmark functions. Compared with the top CEC competition winners, L-SRTDE, NL-SHADE-LBC, and NL-SHADE-RSP, it achieves significant improvements on 11, 22, and 23 out of 29 functions, respectively. Conversely, these algorithms outperform the proposed approach in at most 5, 2, and 3 functions. Against the other eight algorithms, the number of functions where it performs significantly better is 22, 22, 22, 20, 23, 23, 22, and 22. Conversely, these algorithms surpass the proposed approach on only 2–4 functions. The Friedman rankings (Table 3) place the proposed method first overall, and post hoc test results (Table 4) confirm its superiority over all competitors except L-SRTDE. Although the post hoc procedures do not detect a significant difference versus L-SRTDE, both the Wilcoxon outcomes and the Friedman ranks favor the proposed algorithm.

Table 2. Wilcoxon rank-sum results vs. L-SRTDE-ADCP on 30-dimensional problems.

L-SRTDE-ADCP vs.	+	−	≈
L-SRTDE	11	5	13
NL-SHADE-LBC	22	2	5
NL-SHADE-RSP	23	3	3
BWDE	22	2	5
DPSDE	22	2	5
FODE	22	2	5
MSDE	20	4	5
RDE	23	2	4
RSDE	23	2	4
SMLDE	22	3	4
iLSHADE-RSP	22	2	5

Table 3. Algorithm rankings derived from Friedman test on 30-dimensional problems.

Algorithm	Avg. Rank
L-SRTDE-ADCP	2.776
L-SRTDE	4.017
RDE	5.966
BWDE	6.052
SMLDE	6.224
FODE	6.276
DPSDE	6.500
RSDE	6.638
iLSHADE-RSP	7.034
MSDE	7.414
NL-SHADE-RSP	9.414
NL-SHADE-LBC	9.690

Table 4. Post Hoc p-values for compared DE algorithms on 30-dimensional problems.

L-SRTDE-ADCP vs.	z	p	Bonf.	Holm	Hoch.
L-SRTDE	1.311	0.190	1.000	0.190	0.190
NL-SHADE-LBC	7.302	0.000	0.000	0.000	0.000
NL-SHADE-RSP	7.010	0.000	0.000	0.000	0.000
BWDE	3.460	0.001	0.006	0.002	0.002
DPSDE	3.933	0.000	0.001	0.001	0.001
FODE	3.696	0.000	0.002	0.001	0.001
MSDE	4.898	0.000	0.000	0.000	0.000
RDE	3.369	0.001	0.008	0.002	0.002
RSDE	4.079	0.000	0.000	0.000	0.000
SMLDE	3.642	0.000	0.003	0.001	0.001
iLSHADE-RSP	4.498	0.000	0.000	0.000	0.000

- **50-pimensional problems:** The results in Table 5 show consistent performance gains. The proposed algorithm significantly outperforms L-SRTDE, NL-SHADE-LBC, and NL-SHADE-RSP on 14, 26, and 28 functions, respectively. Conversely, these algorithms outperform the proposed approach in at most 7, 1, and 0 functions. For the other eight algorithms, the corresponding counts are 24, 25, 23, 22, 24, 23, 23, and 24. Conversely, these algorithms surpass the proposed approach on only 1–2 functions. The Friedman rankings (Table 6) again place the proposed method first, and post hoc tests (Table 7) confirm its advantage; as in 30-D, no significant difference is detected versus L-SRTDE, yet Wilcoxon outcomes and Friedman ranks favor the proposed algorithm.

Table 5. Wilcoxon rank-sum results vs. L-SRTDE-ADCP on 50-dimensional problems.

L-SRTDE-ADCP vs.	+	−	≈
L-SRTDE	14	7	8
NL-SHADE-LBC	26	1	2
NL-SHADE-RSP	28	0	1
BWDE	24	2	3
DPSDE	25	1	3
FODE	23	2	4
MSDE	22	2	5
RDE	24	2	3
RSDE	23	2	4
SMLDE	23	2	4
iLSHADE-RSP	24	2	3

Table 6. Algorithm rankings derived from Friedman test on 50-dimensional problems.

Algorithm	Avg. Rank
L-SRTDE-ADCP	2.569
L-SRTDE	3.328
RSDE	5.121
MSDE	5.810
RDE	5.983
FODE	6.190
iLSHADE-RSP	6.259
SMLDE	6.638
BWDE	6.776
DPSDE	7.190
NL-SHADE-LBC	10.241
NL-SHADE-RSP	11.897

Table 7. Post Hoc p-values for compared DE algorithms on 50-dimensional problems.

L-SRTDE-ADCP vs.	z	p	Bonf.	Holm	Hoch.
L-SRTDE	0.801	0.423	1.000	0.423	0.423
NL-SHADE-LBC	8.103	0.000	0.000	0.000	0.000
NL-SHADE-RSP	9.851	0.000	0.000	0.000	0.000
BWDE	4.443	0.000	0.000	0.000	0.000
DPSDE	4.880	0.000	0.000	0.000	0.000
FODE	3.824	0.000	0.001	0.001	0.001
MSDE	3.423	0.001	0.007	0.002	0.002
RDE	3.605	0.000	0.003	0.001	0.001
RSDE	2.695	0.007	0.077	0.014	0.014
SMLDE	4.297	0.000	0.000	0.000	0.000
iLSHADE-RSP	3.897	0.000	0.001	0.001	0.001

- **100-dimensional problems:** Table 8 shows that the proposed algorithm maintains strong competitiveness even at higher dimensionalities. It significantly outperforms L-SRTDE, NL-SHADE-LBC, and NL-SHADE-RSP on 7, 25, and 28 functions, respectively. Conversely, these algorithms outperform the proposed approach in at most 4, 1, and 0 functions. Against the remaining eight algorithms, it achieves superior results on 22, 22, 22, 22, 21, 22, 21, and 22 functions. Conversely, these algorithms surpass the proposed approach on only 4–5 functions. The Friedman test results (Table 9) place the proposed algorithm first, and post hoc analyses (Table 10) confirm a statistically significant advantage over all algorithms except L-SRTDE; nonetheless, Wilcoxon outcomes and Friedman ranks consistently favor the proposed algorithm.

Table 8. Wilcoxon rank-sum results vs. L-SRTDE-ADCP on 100-dimensional problems.

L-SRTDE-ADCP vs.	+	−	≈
L-SRTDE	7	4	18
NL-SHADE-LBC	25	1	3
NL-SHADE-RSP	28	0	1
BWDE	22	5	2
DPSDE	22	4	3
FODE	22	5	2
MSDE	22	5	2
RDE	21	4	4
RSDE	22	5	2
SMLDE	21	5	3
iLSHADE-RSP	22	5	2

Table 9. Algorithm rankings derived from Friedman test on 100-dimensional problems.

Algorithm	Avg. Rank
L-SRTDE-ADCP	2.517
L-SRTDE	3.414
MSDE	4.983
RDE	5.759
RSDE	5.983
iLSHADE-RSP	6.276
SMLDE	6.362
FODE	6.397
BWDE	6.707
DPSDE	7.638
NL-SHADE-LBC	10.207
NL-SHADE-RSP	11.759

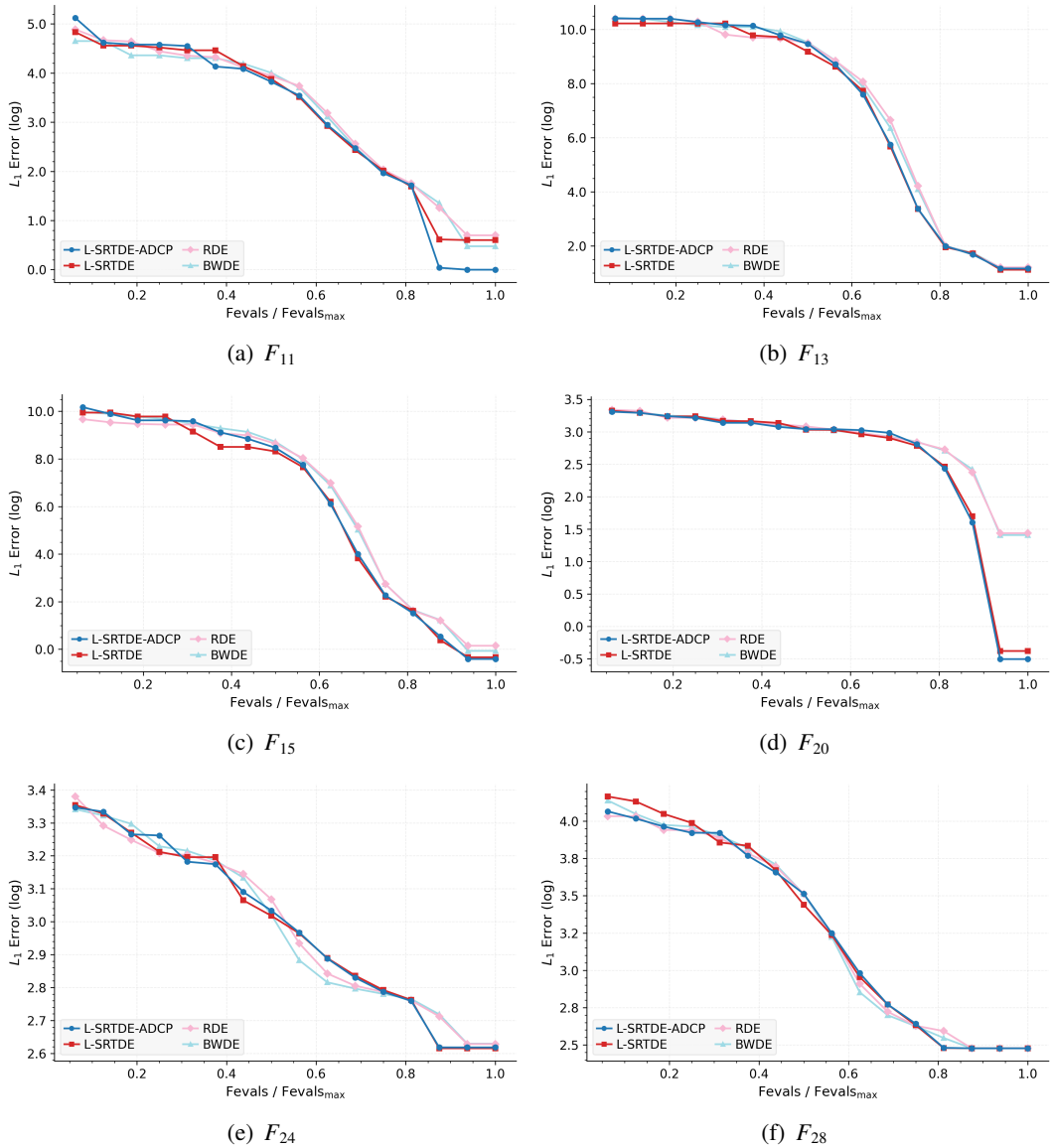
Table 10. Post Hoc p-values for compared DE algorithms on 100-dimensional problems.

L-SRTDE-ADCP vs.	z	p	Bonf.	Holm	Hoch.
L-SRTDE	0.947	0.344	1.000	0.344	0.344
NL-SHADE-LBC	8.121	0.000	0.000	0.000	0.000
NL-SHADE-RSP	9.760	0.000	0.000	0.000	0.000
BWDE	4.425	0.000	0.000	0.000	0.000
DPSDE	5.408	0.000	0.000	0.000	0.000
FODE	4.097	0.000	0.000	0.000	0.000
MSDE	2.604	0.009	0.101	0.018	0.018
RDE	3.423	0.001	0.007	0.002	0.002
RSDE	3.660	0.000	0.003	0.001	0.001
SMLDE	4.061	0.000	0.001	0.000	0.000
iLSHADE-RSP	3.970	0.000	0.001	0.000	0.000

Across all dimensionalities, the statistical tests consistently demonstrate that the proposed algorithm delivers the best overall performance among all compared DE variants, highlighting its robustness and strong optimization capability.

From these results, several observations can be made. First, L-SRTDE-ADCP not only achieves the best average Friedman rank in all dimensional settings but also exhibits a clear and statistically significant advantage over almost all competing variants, including several recent CEC competition winners. This indicates that the proposed dimension-wise perturbation mechanism provides a performance gain that is robust across different problem sizes and algorithmic baselines. Second, the superiority of L-SRTDE-ADCP is particularly pronounced when compared with advanced Cauchy-based and population-structuring methods such as NL-SHADE-LBC, NL-SHADE-RSP, BWDE, DPSDE, FODE, RSDE, and SMLDE. These algorithms already incorporate sophisticated parameter adaptation and diversity-preservation strategies, yet the proposed method still yields significantly lower objective values on the majority of benchmark functions. Third, the fact that the post hoc procedures

do not detect a statistically significant difference between L-SRTDE-ADCP and its base algorithm L-SRTDE, whereas the Wilcoxon outcomes and Friedman ranks consistently favor L-SRTDE-ADCP, indicating that the proposed mechanism improves performance in a gradual and stable manner without drastically altering the behavior of the underlying framework. In other words, ADCP refines the search dynamics of L-SRTDE by reallocating perturbation strength toward over-converged dimensions and attenuating unnecessary large jumps in less converged dimensions. This leads to better final solution quality and more reliable convergence. Overall, these insights confirm that the proposed dimension-wise adaptive Cauchy perturbation constitutes a meaningful enhancement of the L-SRTDE framework rather than a marginal or problem-specific modification.



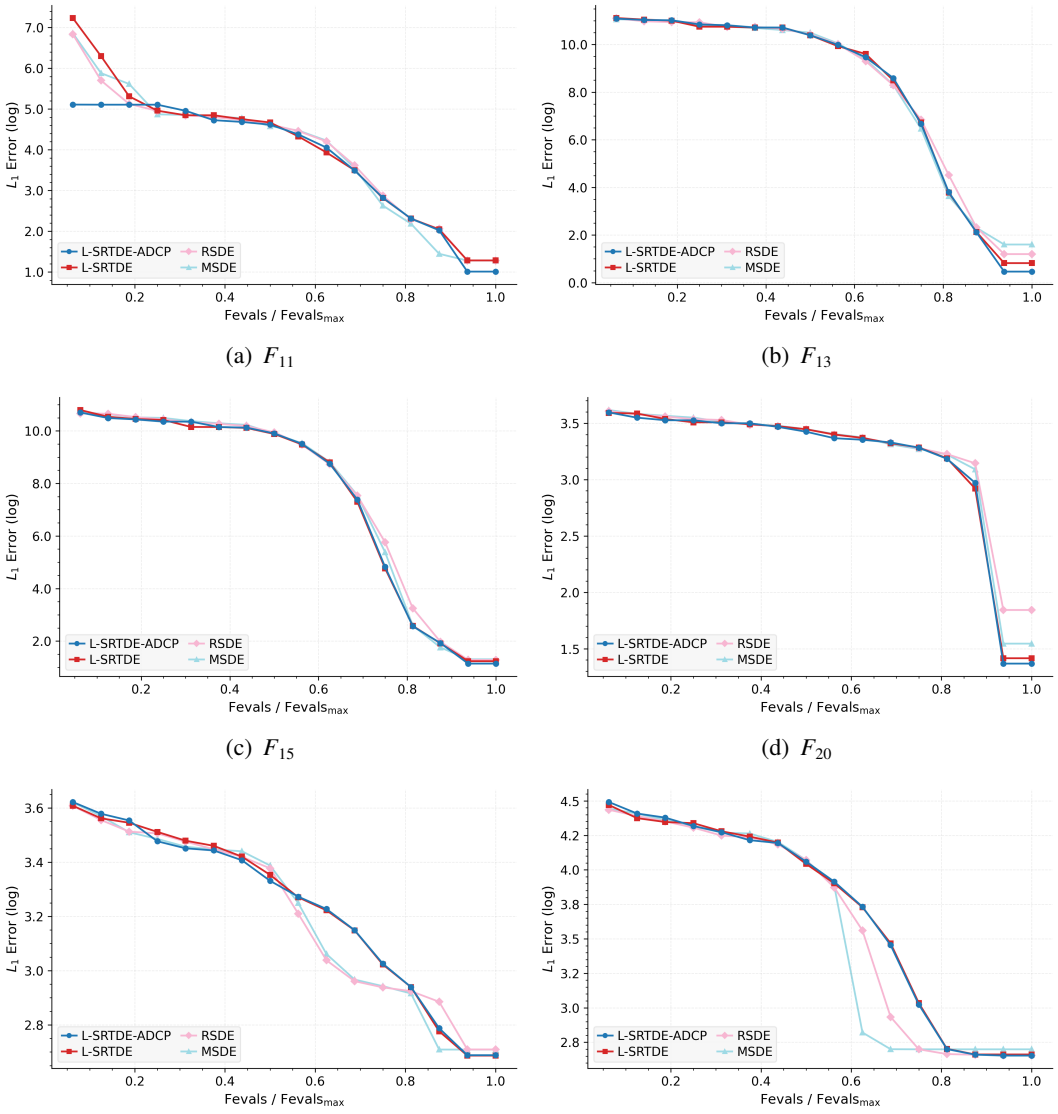
(a) F_{11} (b) F_{13}

(c) F_{15} (d) F_{20}

(e) F_{24} (f) F_{28}

Figure 3. Convergence plots depicting the median optimization errors of the proposed algorithm compared to the top three best-performing competitors across six benchmark functions in 30 dimensions.

The convergence plots shown in Figures 3–5 illustrate the comparison of median optimization errors between the proposed algorithm and the top three best-performing competitors. Six representative benchmark functions (F_{11}, F_{13}, F_{15}, F_{20}, F_{24}, and F_{28}) are selected for analysis. The results clearly demonstrate that the proposed algorithm consistently achieves the lowest average error at the end of execution, highlighting its superior search capability.

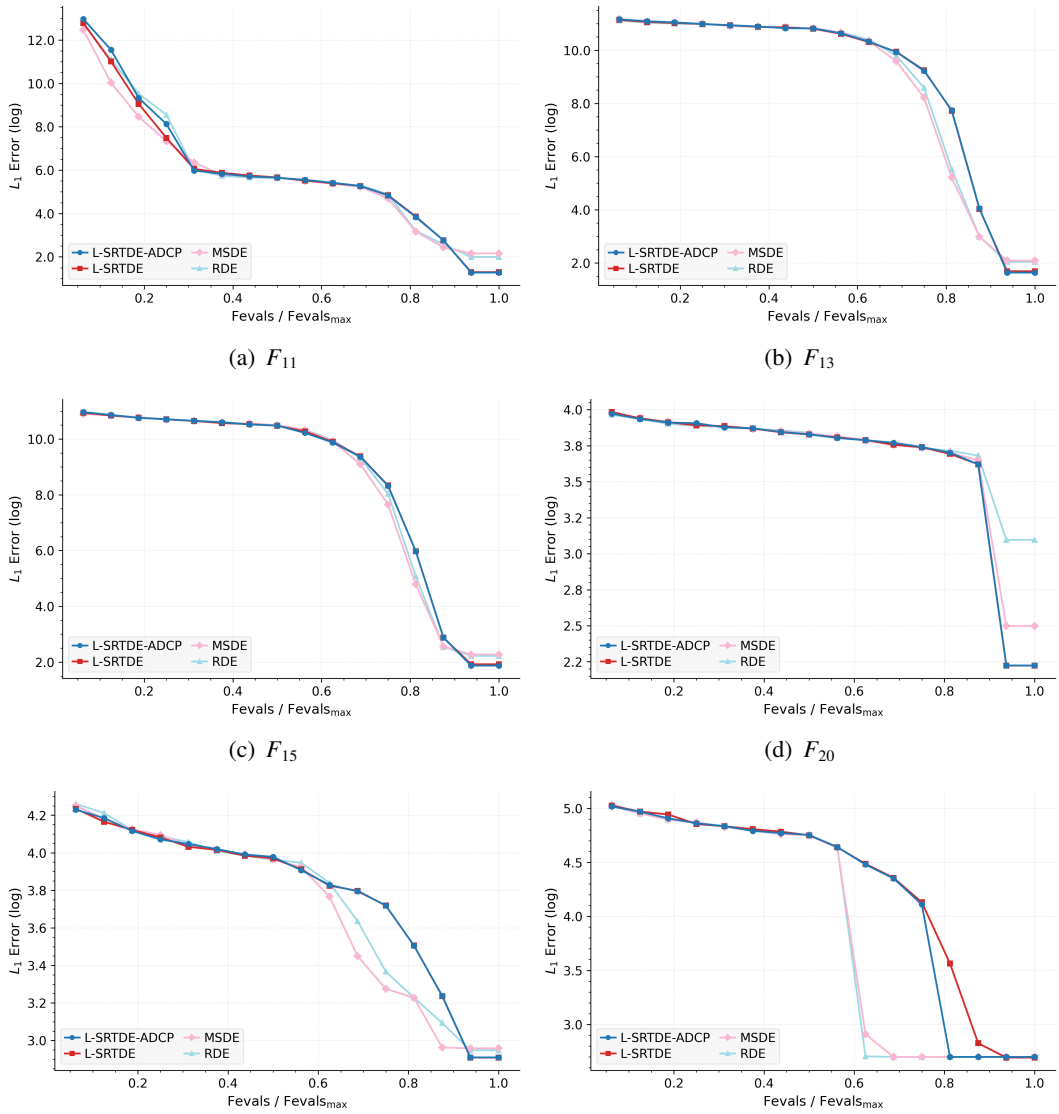


(a) F_{11} (b) F_{13}

(c) F_{15} (d) F_{20}

(e) F_{24} (f) F_{28}

Figure 4. Convergence plots depicting the median optimization errors of the proposed algorithm compared to the top three best-performing competitors across six benchmark functions in 50 dimensions.



(a) F_{11} (b) F_{13}

(c) F_{15} (d) F_{20}

(e) F_{24} (f) F_{28}

Figure 5. Convergence plots depicting the median optimization errors of the proposed algorithm compared to the top three best-performing competitors across six benchmark functions in 100 dimensions.

5.3. Comparison with traditional Cauchy perturbation

This section summarizes the results comparing the performance of the proposed algorithm against two variants: L-SRTDE-CP (L-SRTDE with traditional Cauchy perturbation) and the original L-SRTDE. The evaluations were conducted across 30-dimensional, 50-dimensional, and 100-dimensional problems. Detailed numerical results are provided in the supplementary material. For each dimensionality setting, the Wilcoxon rank-sum test [39] was used to assess pairwise performance differences.

- **30-dimensional problems:** As shown in Table 11, the proposed algorithm achieves modest

improvements compared to L-SRTDE-CP and the original L-SRTDE. It significantly outperforms L-SRTDE-CP and L-SRTDE on 7 and 11 functions, respectively. Conversely, it is outperformed on only 5 functions in both comparisons. The number of comparable outcomes (ties) is relatively high, indicating moderate differentiation among algorithms in this lower-dimensional setting. Given that L-SRTDE-ADCP is an enhanced version of L-SRTDE-CP, these improvements are attained without compromising overall performance.

- **50-dimensional problems:** In Table 12, the advantages of L-SRTDE-ADCP become clearer. It significantly outperforms L-SRTDE-CP and L-SRTDE on 13 and 14 functions, respectively, with relatively fewer instances where it underperforms (5 and 7 functions, respectively). The reduced number of ties suggests clearer algorithmic differentiation at this dimensionality, reinforcing the effectiveness of the proposed method. Again, this highlights that the proposed enhancements contribute positively without negatively impacting performance.
- **100-dimensional problems:** Table 13 indicates diminished differentiation at higher dimensions. L-SRTDE-ADCP significantly outperforms L-SRTDE-CP and L-SRTDE on only 5 and 7 functions, respectively. Comparable performances dominate (19 and 18 ties, respectively), suggesting a limited relative advantage in higher-dimensional settings. Importantly, even in the challenging high-dimensional scenario, the improvements in L-SRTDE-ADCP are realized without performance degradation relative to its predecessor.

Table 11. Wilcoxon rank-sum test results comparing L-SRTDE-ADCP against L-SRTDE-CP and L-SRTDE on 30-dimensional problems.

L-SRTDE-ADCP vs.	+	−	≈
L-SRTDE-CP	7	5	17
L-SRTDE	11	5	13

Table 12. Wilcoxon rank-sum test results comparing L-SRTDE-ADCP against L-SRTDE-CP and L-SRTDE on 50-dimensional problems.

L-SRTDE-ADCP vs.	+	−	≈
L-SRTDE-CP	13	5	11
L-SRTDE	14	7	8

Table 13. Wilcoxon rank-sum test results comparing L-SRTDE-ADCP against L-SRTDE-CP and L-SRTDE on 100-dimensional problems.

L-SRTDE-ADCP vs.	+	−	≈
L-SRTDE-CP	5	5	19
L-SRTDE	7	4	18

Overall, the proposed algorithm demonstrates notable performance advantages, particularly in 50-dimensional problems, with modest improvements in 30-dimensional problems and consistent competitiveness in 100-dimensional problems. These consistent gains underline that the enhancements introduced in L-SRTDE-ADCP contribute positively without compromising performance.

These observations provide additional insight into why the proposed ADCP mechanism is effective and how it differs from the traditional Cauchy perturbation. In the conventional scheme, a single global jumping rate JR is applied uniformly to all dimensions, regardless of their convergence state. This can lead to redundant or excessively large perturbations in coordinates that have not yet converged while still being insufficient to rediversify dimensions where the population has already collapsed. In contrast, L-SRTDE-ADCP uses the per-dimension convergence indicator CL_j^g in Eq (4.2) to modulate the jumping rate JR_j via Eq (4.3). This dimension-wise control focuses strong Cauchy perturbations on over-converged coordinates, improving the ability to escape local optima while keeping the search more stable in dimensions that remain diverse.

As shown in Tables 11–13, this targeted use of Cauchy noise yields a clear net benefit over both the base algorithm L-SRTDE and its traditional Cauchy-perturbed variant L-SRTDE-CP in 30- and 50-dimensional settings. In 100 dimensions, where all algorithms face a substantially more challenging search landscape, the advantage of L-SRTDE-ADCP becomes less pronounced but remains non-negative, indicating that the additional perturbation mechanism does not destabilize the search even in high-dimensional settings. Taken together, these findings support the design rationale of ADCP: by aligning the strength of Cauchy perturbations with the dimension-wise convergence state, the algorithm achieves a more favorable balance between exploration and exploitation than is possible with a fixed, global perturbation scheme.

5.4. Algorithm complexity

We measured the computational cost of the proposed method using the CEC 2017 benchmarking protocol. The calibration unit time T_0 was obtained by running the following micro-benchmark:

```
x=0.55;
for i = 1; i < 1000000; i = i + 1 do
    x=x+x; x=x/2; x=x*x; x=sqrt(x);
    x=log(x); x=exp(x); x=x/(x+2);
end
```

We denote by T_1 the elapsed time to evaluate function f_{18} for 200,000 calls (function-only cost). The quantity T_2 is the wall-clock time of an algorithm that performs 200,000 evaluations of the same function; $\hat{T}_2$ is the mean of five independent T_2 measurements. Following the CEC convention, we report the algorithmic overhead as the normalized quantity

$$\frac{\hat{T}_2 - T_1}{T_0},$$

which expresses the extra computation (in units of the calibration loop) required by the algorithm beyond pure function-evaluation time.

Table 14 presents these normalized overheads for L-SRTDE augmented with the proposed adaptive Cauchy perturbation (L-SRTDE-ADCP), L-SRTDE with the traditional Cauchy perturbation (L-SRTDE-CP), and the original L-SRTDE. For dimensions $D \in \{10, 30, 50\}$, L-SRTDE-ADCP yields overheads of 13.712085 (10D), 25.189362 (30D), and 33.193663 (50D). These are of the same magnitude as the corresponding values for L-SRTDE-CP and the original L-SRTDE, indicating that the proposed ADCP extension introduces no substantive runtime penalty while delivering the performance

improvements reported in Section 5.

Table 14. Algorithmic complexity results.

Algorithm	D	T_0	T_1	$\hat{T}_2$	$\frac{\hat{T}_2 - T_1}{T_0}$
L-SRTDE-ADCP	10	0.016789	0.088241	0.318453	13.712085
	30		0.441385	0.864289	25.189362
	50		1.186522	1.743810	33.193663
L-SRTDE-CP	10	0.016789	0.088241	0.306227	12.983847
	30		0.441385	0.841637	23.840145
	50		1.186522	1.701841	30.693859
L-SRTDE	10	0.016789	0.088241	0.319467	13.772470
	30		0.441385	0.861633	25.031163
	50		1.186522	1.713155	31.367717

Time complexity: Let D denote the problem dimension and N_g the population size in generation g with $N_g \leq N_{\max} = 20D$. Let $N_{\mathrm{FE}}^{\max}$ be the evaluation budget and T_f the cost of a single objective evaluation. For the baseline L-SRTDE algorithm, the cost of mutation, crossover, selection, and ranking in one generation is

$$T_{\mathrm{gen}}^{\text{L-SRTDE}} = O(N_g D + N_g \log N_g + N_g T_f),$$

so over the whole run, we obtain

$$T_{\text{L-SRTDE}} = O(N_{\mathrm{FE}}^{\max} D) + O(N_{\mathrm{FE}}^{\max} T_f).$$

The traditional Cauchy perturbation modifies only the recombination operator. In each generation, for every individual and every coordinate the operator performs a constant number of random draws, comparisons, and assignments according to (4.1), that is $O(D)$ work per individual. Therefore, the per-generation cost of the L-SRTDE-CP variant remains

$$T_{\mathrm{gen}}^{\mathrm{CP}} = O(N_g D + N_g \log N_g + N_g T_f),$$

and the overall time complexity is

$$T_{\mathrm{CP}} = O(N_{\mathrm{FE}}^{\max} D) + O(N_{\mathrm{FE}}^{\max} T_f).$$

The proposed ADCP mechanism adds two ingredients: (i) the computation of the per-dimension standard deviations σ_j^g and convergence levels CL_j^g together with the updated jumping rates JR_j via (4.2) and (4.3), and (ii) the dimension-wise Cauchy perturbation in (4.4). The first part requires a single pass over the population for each dimension and thus $O(N_g D)$ time per generation. The second part is again $O(D)$ per individual because generating a uniform or Cauchy random variable is constant time. Consequently, the per-generation cost of L–SRTDE–ADCP is

$$T_{\mathrm{gen}}^{\mathrm{ADCP}} = O(N_g D + N_g \log N_g + N_g T_f),$$

and the overall time complexity satisfies

$$T_{\mathrm{ADCP}} = O(N_{\mathrm{FE}}^{\max} D) + O(N_{\mathrm{FE}}^{\max} T_f).$$

In other words, both the traditional Cauchy perturbation and the proposed ADCP mechanism preserve the $O(N_{\mathrm{FE}}^{\max} D)$ algorithmic time complexity of L-SRTDE, incurring only a small constant-factor overhead.

5.5. Analysis of parameter settings

This section presents experimental results evaluating the robustness of the proposed method. The method involves three control parameters: JR_{min}, JR_{max}, and k, which are used to calculate JR_j. Specifically, JR_{min} and JR_{max} define the minimum and maximum allowable values of JR_j, respectively, while the parameter k controls the steepness of the exponential growth in the scaling function used to determine JR_j.

The robustness evaluations were conducted on 29 50-dimensional CEC 2017 benchmark functions. Two nonparametric statistical tests, Friedman's test [40] and the Iman–Davenport test [44], were utilized to analyze the sensitivity and robustness of the proposed algorithm.

5.5.1. Sensitivity of $JR_{\max}$

We assessed the effect of $JR_{\max}$ by fixing $JR_{\min} = 0.1$ and $k = 10$ and evaluating five values of $JR_{\max} \in \{0.2, 0.4, 0.6, 0.8, 1.0\}$ across the benchmark suite. In the algorithm, $JR_{\max}$ is the probability of applying a Cauchy perturbation once the per-dimension convergence level CL_j reaches 1.0 (i.e., when all individuals have collapsed to the same point on dimension j).

The Friedman omnibus test shown in Table 15 indicated a significant difference among the five configurations (Friedman $\chi^2_{(4)} = 9.899$, $p = 0.042$); however, the effect size is small (Kendall's $W \approx 0.085$). The Iman–Davenport correction yields $F_{4,112} \approx 2.612$, $p \approx 0.039$. Because the omnibus effect is modest, we followed up with pairwise post hoc tests against the reference configuration $JR_{\max} = 0.2$.

Pairwise Wilcoxon rank-sum tests versus the reference are summarized in Table 16. Table 17 lists the per-comparison z-statistics, raw p-values, and family-wise adjustments (Bonferroni, Holm, Hochberg). After correction, only $JR_{\max} = 1.0$ differs from the reference at $\alpha = 0.05$ (raw $p = 0.010$; Holm $p = 0.040$). The effect size for this comparison, computed as $r = z/\sqrt{N}$, is $r \approx 0.48$, indicates a moderate-to-large effect.

Average Friedman ranks (Table 18) show $JR_{\max} = 0.2$ has the best rank and $JR_{\max} = 1.0$ the worst. Taken together, the statistical evidence suggests that values of $JR_{\max}$ in the range 0.2–0.4 are preferable for the proposed algorithm on these benchmarks.

Table 15. Friedman and Iman–Davenport test results for five JR_{max} configurations.

Friedman value	χ^2 value	p-value	Iman–Davenport value	F_F value	p-value
9.899	9.488	0.042	2.612	2.453	0.039

Table 16. Wilcoxon rank-sum results vs. $JR_{max} = 0.2$ on 50-dimensional problems.

$JR_{min} = 0.1$, $JR_{max} = 0.2$ vs.	+	−	≈
$JR_{min} = 0.1$, $JR_{max} = 0.4$	10	6	13
$JR_{min} = 0.1$, $JR_{max} = 0.6$	11	5	13
$JR_{min} = 0.1$, $JR_{max} = 0.8$	13	3	13
$JR_{min} = 0.1$, $JR_{max} = 1.0$	12	8	9

Table 17. Post Hoc p-values for five JR_{max} configurations on 50-dimensional problems.

$JR_{min} = 0.1$, $JR_{max} = 0.2$ vs.	z	p	Bonf.	Holm	Hoch.
$JR_{min} = 0.1$, $JR_{max} = 0.4$	0.249	0.803	1.000	0.803	0.803
$JR_{min} = 0.1$, $JR_{max} = 0.6$	1.080	0.280	1.000	0.561	0.561
$JR_{min} = 0.1$, $JR_{max} = 0.8$	1.910	0.056	0.225	0.168	0.168
$JR_{min} = 0.1$, $JR_{max} = 1.0$	2.574	0.010	0.040	0.040	0.040

Table 18. Algorithm rankings derived from Friedman test with five JR_{max} configurations.

Algorithm	Avg. Rank
$JR_{min} = 0.1$, $JR_{max} = 0.2$	2.517
$JR_{min} = 0.1$, $JR_{max} = 0.4$	2.621
$JR_{min} = 0.1$, $JR_{max} = 0.6$	2.966
$JR_{min} = 0.1$, $JR_{max} = 0.8$	3.310
$JR_{min} = 0.1$, $JR_{max} = 1.0$	3.586

5.5.2. Sensitivity of k

We evaluated the effect of the parameter k by comparing five settings $k \in \{0, 5, 10, 20, 30\}$ across the benchmark suite. In the algorithm, k controls the steepness of the exponential scaling function used to compute JR_j. Larger values of k produce a sharper, more rapid transition from $JR_{\min}$ to $JR_{\max}$ as the convergence level increases, while smaller values of k yield a smoother, more gradual rise. Figure 6 illustrates the effect of varying k on the shape of this scaling function for the fixed bounds $JR_{\min} = 0.1$ and $JR_{\max} = 1.0$.

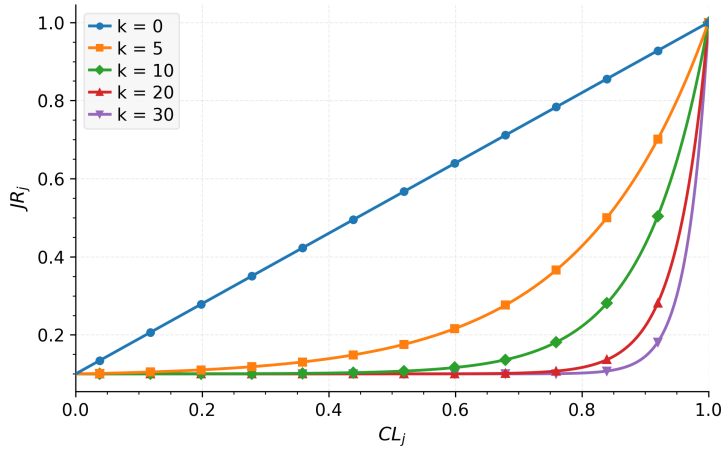


Figure 6. Comparison of jumping rate functions for different values of the scaling parameter k. Lower k values yield a linear or gently curved relationship, whereas higher k values lead to increasingly nonlinear, exponential adaptation behavior.

The Friedman omnibus statistic shown in Table 19 does not indicate a statistically significant difference among the five configurations (Friedman $\chi^2_{(4)} = 7.299$, $p = 0.121$). Consistent with the Friedman result, the Iman–Davenport corrected test yields $F_{4,112} \approx 1.880$, $p \approx 0.119$; that is, the omnibus test fails to reach $\alpha = 0.05$. The associated effect size is very small: Kendall's $W \approx \frac{7.299}{29\cdot(5-1)} \approx 0.063$, indicating only a negligible degree of concordance among ranks across problems.

Table 19. Friedman and Iman–Davenport test results for five k configurations.

Friedman value	χ^2 value	p-value	Iman–Davenport value	F_F value	p-value
7.299	9.488	0.121	1.880	2.453	0.119

Because the omnibus test is not significant, strict inferential practice does not require pairwise post hoc comparisons; nevertheless, the average Friedman ranks in Table 20 provide an informative descriptive ordering. The configuration $k = 30$ attains the best mean rank (2.448), and $k = 0$ is worst (3.500), with the remaining settings lying close together (ranks $\approx$ 2.95–3.07). Taken together, these results indicate that any practical differences among the tested k values are small on this benchmark set. Although $k = 30$ yields the best average rank, we adopt $k = 10$ as the default because it provides comparably strong performance while exhibiting slightly more robust run-to-run behavior across functions (i.e., lower typical variability in the final error), and it corresponds to a less aggressive mapping from CL_j^g to JR_j that reduces the risk of overly disruptive perturbations.

Table 20. Algorithm rankings derived from Friedman test with five k configurations.

Algorithm	Avg. Rank
$k = 30$	2.448
$k = 10$	2.948
$k = 5$	3.034
$k = 20$	3.069
$k = 0$	3.500

6. Conclusions

We proposed an adaptive dimension-wise Cauchy perturbation (ADCP) that estimates the per-dimension convergence level each generation and adjusts the perturbation probability accordingly. Integrated into the L-SRTDE framework, ADCP targets dimensions that have prematurely converged while preserving diversity where it already exists. Extensive experiments on the CEC 2017 test suite across 30-, 50-, and 100-dimensional settings showed consistent improvements in convergence speed and final solution quality over the original L-SRTDE and a range of recent DE variants. The proposed method achieved top average ranks in Friedman analyses, with gains most pronounced in 50-dimensional problems, and introduced no substantive runtime overhead relative to strong baselines. Sensitivity studies further indicated that ADCP is robust to its hyperparameters, with JR_{max} around 0.2 and k in the 10–30 range providing reliable performance.

Despite these encouraging results, the present study has several limitations. First, the evaluation focused on unconstrained, single-objective, continuous optimization problems and on dimensions up to 100. The behavior of ADCP on very high-dimensional problems, constrained or noisy optimization tasks, or combinatorial search spaces remains to be systematically assessed. Second, ADCP was studied in depth only within the L-SRTDE framework; although its design is modular, we have not yet quantified how its benefits transfer to other metaheuristics.

These limitations naturally suggest several directions for future research. One avenue is to integrate ADCP into other state-of-the-art swarm and evolutionary frameworks that suffer from premature convergence and to benchmark them on more diverse suites (including large-scale, constrained, and noisy problems) and higher-dimensional settings. It is also of interest to couple ADCP with constraint-handling and multiobjective selection strategies and to evaluate its effectiveness on real-world engineering and machine-learning applications, including finite-dimensional formulations of impulsive differential systems [45] and other differential or integro-differential models arising in practice.

Author contributions

Tae Jong Choi: Conceptualization, formal analysis, funding acquisition, investigation, methodology, resources, software, visualization, writing–original draft; Yeji An: Data curation, funding acquisition, investigation, project administration, supervision, validation, writing–review & editing. All authors have read and approved the final version of the manuscript for publication.

Use of Generative-AI tools declaration

The authors declare they have not used Artificial Intelligence (AI) tools in the creation of this article.

Acknowledgments

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. RS-2023-00214326). This work was supported by the Ministry of Education of the Republic of Korea and the National Research Foundation of Korea (NRF-2023S1A5A8079831).

Conflict of interest

Authors Tae Jong Choi and Yeji An are spouses. The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

1. R. Storn, K. Price, Differential evolution–A simple and efficient heuristic for global optimization over continuous spaces, *J. Global Optim.*, **11** (1997), 341–359. https://doi.org/10.1023/a:1008202821328
2. M. Črepinšek, S.-H. Liu, M. Mernik, Exploration and exploitation in evolutionary algorithms: A survey, *ACM Comput. Surv.*, **45** (2013), 1–33. https://doi.org/10.1145/2480741.2480752
3. A. E. Eiben, C. A. Schippers, On evolutionary exploration and exploitation, *Fund. Inform.*, **35** (1998), 35–50. https://doi.org/10.3233/fi-1998-35123403
4. K. A. D. Jong, An analysis of the behavior of a class of genetic adaptive systems, PhD Thesis, University of Michigan, 1975.
5. T. J. Choi, C. W. Ahn, An improved LSHADE-RSP algorithm with the Cauchy perturbation: iLSHADE-RSP, *Knowl.-Based Syst.*, **215** (2021), 106628. https://doi.org/10.1016/j.knosys.2020.106628
6. Q. Y. Sui, Y. Yu, K. Y. Wang, L. Zhong, Z. Y. Lei, S. C. Gao, Best-worst individuals driven multiple-layered differential evolution, *Inform. Sciences*, **655** (2024), 119889. https://doi.org/10.1016/j.ins.2023.119889
7. J. R. Yang, K. Y. Wang, Y. R. Wang, J. H. Wang, Z. Y. Lei, S. C. Gao, Dynamic population structures-based differential evolution algorithm, *IEEE Transactions on Emerging Topics in Computational Intelligence*, **8** (2024), 2493–2505. https://doi.org/10.1109/tetci.2024.3367809
8. K. Y. Wang, S. C. Gao, M. C. Zhou, Z.-H. Zhan, J. J. Cheng, Fractional order differential evolution, *IEEE T. Evolut. Comput*, **29** (2025), 822–835. https://doi.org/10.1109/TEVC.2024.3382047
9. V. Stanovov, E. Semenkin, Success rate-based adaptive differential evolution L-SRTDE for CEC 2024 competition, *2024 IEEE Congress on Evolutionary Computation (CEC)*, Yokohama, Japan, 2024, 1–8. https://doi.org/10.1109/cec60901.2024.10611907
10. N. H. Awad, M. Z. Ali, J. J. Liang, B. Y. Qu, P. N. Suganthan, Problem definitions and evaluation criteria for the CEC 2017 special session and competition on single objective bound constrained real-parameter numerical optimization, Technical Report, Singapore: Nanyang Technological University, 2016, 1–34.
11. T. J. Choi, C. W. Ahn, J. An. An adaptive Cauchy differential evolution algorithm for global numerical optimization, *Sci. World J.*, **2013** (2013), 969734. https://doi.org/10.1155/2013/969734
12. T. J. Choi, C. W. Ahn, An adaptive population resizing scheme for differential evolution in numerical optimization, *J. Comput. Theor. Nanos.*, **12** (2015), 1336–1350.
13. T. J. Choi, J. Togelius, Y.-G. Cheong, Advanced Cauchy mutation for differential evolution in numerical optimization, *IEEE Access*, **8** (2020), 8720–8734. https://doi.org/10.1109/access.2020.2964222

14. T. J. Choi, J. Togelius, Y.-G. Cheong, A fast and efficient stochastic opposition-based learning for differential evolution in numerical optimization, *Swarm Evol. Comput.*, **60** (2021), 100768. https://doi.org/10.1016/j.swevo.2020.100768

15. T. J. Choi, An efficient eigenvector-based crossover for differential evolution: Simplifying with rank-one updates, *AIMS Mathematics*, **10** (2025), 3500–3522. https://doi.org/10.3934/math.2025162

16. T. J. Choi, C. W. Ahn, Artificial life based on boids model and evolutionary chaotic neural networks for creating artworks, *Swarm Evol. Comput.*, **47** (2019), 80–88. https://doi.org/10.1016/j.swevo.2017.09.003

17. T. J. Choi, J. Togelius, Self-referential quality diversity through differential MAP-Elites. In: *Proceedings of the Genetic and Evolutionary Computation Conference*, New York: Association for Computing Machinery, 2021, 502–509. https://doi.org/10.1145/3449639.3459383

18. T. J. Choi, J.-H. Lee, H. Y. Youn, C. W. Ahn, Adaptive differential evolution with elite opposition-based learning and its application to training artificial neural networks, *Fund. Inform.*, **164** (2019), 227–242. https://doi.org/10.3233/fi-2019-1764

19. Y. Zhang, D.-W. Gong, X.-Z. Gao, T. Tian, X.-Y. Sun, Binary differential evolution with self-learning for multi-objective feature selection, *Inform. Sciences*, **507** (2020), 67–85. https://doi.org/10.1016/j.ins.2019.08.040

20. S. Das, P. N. Suganthan, Differential evolution: A survey of the state-of-the-art, *IEEE T. Evolut. Comput.*, **15** (2010), 4–31. https://doi.org/10.1109/tevc.2010.2059031

21. S. Das, S. S. Mullick, P. N. Suganthan, Recent advances in differential evolution–An updated survey, *Swarm Evol. Comput.*, **27** (2016), 1–30. https://doi.org/10.1016/j.swevo.2016.01.004

22. X. G. Ye, J. P. Li, P. Wang, P. N. Suganthan, A comprehensive survey of adaptive strategies in differential evolutionary algorithms, *Swarm Evol. Comput.*, **98** (2025), 102081. https://doi.org/10.1016/j.swevo.2025.102081

23. M. Ali, M. Pant, Improving the performance of differential evolution algorithm using Cauchy mutation, *Soft Comput.*, **15** (2011), 991–1007. https://doi.org/10.1007/s00500-010-0655-2

24. J. Q. Zhang, A. C. Sanderson, JADE: Adaptive differential evolution with optional external archive, *IEEE T. Evolut. Comput.*, **13** (2009), 945–958. https://doi.org/10.1109/tevc.2009.2014613

25. R. Tanabe, A. Fukunaga, Success-history based parameter adaptation for differential evolution, *2013 IEEE Congress on Evolutionary Computation*, Cancun, Mexico, 2013, 71–78. https://doi.org/10.1109/cec.2013.6557555

26. R. Tanabe, A. S. Fukunaga, Improving the search performance of SHADE using linear population size reduction, *2014 IEEE Congress on Evolutionary Computation (CEC)*, Beijing, China, 2014, 1658–1665. https://doi.org/10.1109/cec.2014.6900380

27. J. Brest, M. S. Maučec, B. Bošković, Single objective real-parameter optimization: Algorithm jSO, *2017 IEEE Congress on Evolutionary Computation (CEC)*, Donostia, Spain, 2017, 1311–1318. https://doi.org/10.1109/cec.2017.7969456

28. V. Stanovov, S. Akhmedova, E. Semenkin, LSHADE algorithm with rank-based selective pressure strategy for solving CEC 2017 benchmark problems, *2018 IEEE Congress on Evolutionary Computation (CEC)*, Rio de Janeiro, Brazil, 2018, 1–8. https://doi.org/10.1109/cec.2018.8477977

29. V. Stanovov, S. Akhmedova, E. Semenkin, NL-SHADE-RSP algorithm with adaptive archive and selective pressure for CEC 2021 numerical optimization, *2021 IEEE Congress on Evolutionary Computation (CEC)*, Kraków, Poland, 2021, 809–816. https://doi.org/10.1109/cec45853.2021.9504959

30. V. Stanovov, S. Akhmedova, E. Semenkin, NL-SHADE-LBC algorithm with linear parameter adaptation bias change for CEC 2022 numerical optimization, *2022 IEEE Congress on Evolutionary Computation (CEC)*, Padua, Italy, 2022, 1–8. https://doi.org/10.1109/cec55065.2022.9870295

31. V. Stanovov, S. Akhmedova, E. Semenkin, Dual-population adaptive differential evolution algorithm L-NTADE, *Mathematics*, **10** (2022), 4666. https://doi.org/10.3390/math10244666

32. D. Chauhan, A. Trivedi, Shivani, A multi-operator ensemble LSHADE with restart and local search mechanisms for single-objective optimization, 2024, arXiv:2409.15994. https://doi.org/10.48550/arXiv.2409.15994

33. A. Stacey, M. Jancic, I. Grundy, Particle swarm optimization with mutation, *The 2003 Congress on Evolutionary Computation (CEC'03)*, Canberra, ACT, Australia, 2003, 1425–1430. https://doi.org/10.1109/CEC.2003.1299838

34. T. J. Choi, C. W. Ahn, Accelerating differential evolution using multiple exponential Cauchy mutation, In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, New York: Association for Computing Machinery, 2018, 207–208. https://doi.org/10.1145/3205651.3205689

35. Z. H. Cai, S. C. Gao, X. Yang, M. C. Zhou, Multiselection-based differential evolution, *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, **54** (2024), 7318–7330. https://doi.org/10.1109/tsmc.2024.3447051

36. S. C. Tao, R. H. Zhao, K. Y. Wang, S. C. Gao, An efficient reconstructed differential evolution variant by some of the current state-of-the-art strategies for solving single objective bound constrained problems, 2024, arXiv:2404.16280. https://doi.org/10.48550/arXiv.2404.16280

37. Z. Li, K. Y. Wang, C. X. Xue, H. T. Li, Y. Todo, Z. Y. Lei, et al., Differential evolution with ring sub-population architecture for optimization, *Knowl.-Based Syst.*, **305** (2024), 112590. https://doi.org/10.1016/j.knosys.2024.112590

38. J. T. Y. Yu, K. Y. Wang, Z. Y. Lei, J. J. Cheng, S. C. Gao, Serial multilevel-learned differential evolution with adaptive guidance of exploration and exploitation, *Expert Syst. Appl.*, **255** (2024), 124646. https://doi.org/10.1016/j.eswa.2024.124646

39. F. Wilcoxon, Individual comparisons by ranking methods, *Biometrics Bulletin*, **1** (1945), 80–83. https://doi.org/10.2307/3001968

40. M. Friedman, The use of ranks to avoid the assumption of normality implicit in the analysis of variance, *J. Am. Stat. Assoc.*, **32** (1937), 675–701. https://doi.org/10.2307/2279372

41. O. J. Dunn, Multiple comparisons among means, *J. Am. Stat. Assoc.*, **56** (1961), 52–64. https://doi.org/10.2307/2282330

42. S. Holm, A simple sequentially rejective multiple test procedure, *Scand. J. Statist.*, **6** (1979), 65–70.

43. Y. Hochberg, A sharper Bonferroni procedure for multiple tests of significance, *Biometrika*, **75** (1988), 800–802. https://doi.org/10.2307/2336325

44. R. L. Iman, J. M. Davenport, Approximations of the critical region of the friedman statistic, *Commun. Stat.-Theor. M.*, **9** (1980), 571–595. https://doi.org/10.1080/03610928008827904

45. B. Hu, Y. T. Qiu, W. T. Zhou, L. Y. Zhu, Existence of solution for an impulsive differential system with improved boundary value conditions, *AIMS Mathematics*, **8** (2023), 17197–17207. https://doi.org/10.3934/math.2023878

AIMS Press