

Article

DSD-LI-PO: A Bioinspired Parrot Optimizer with Componentwise Swarm Hybridization and Contracted Lens Imaging Refinement for Constrained Engineering Optimization

Meiqi Qin  and Yongmei Ding *

School of Mathematics and Systems Science, Wuhan University of Science and Technology, Wuhan 430065, China; qinmeiqi@wust.edu.cn

* Correspondence: dingyongmei@wust.edu.cn

Abstract

Bioinspired swarm optimizers are widely used for constrained engineering optimization, yet whole-vector updates can disrupt well-evolved components and insufficient elite refinement can limit late-stage accuracy. This study proposes DSD-LI-PO, an enhanced Parrot Optimizer (PO) integrating dynamic swarm dimension hybridization (DSD) with stagnation-triggered variable precision lens imaging (LI). DSD selectively inherits components from the current best solution to preserve useful dimensional information, whereas LI generates a progressively contracted elite-derived candidate when stagnation occurs. DSD-LI-PO is evaluated on CEC2017, six representative functions at 30, 50, and 100 dimensions, ablation variants, and five constrained engineering design problems. On CEC2017, it ranks second overall with an average rank of 1.8966 and records 28/1/0 wins/ties/losses against PO and 29/0/0 against IPO. In the scalability tests, it remains second in all three dimensions, with average ranks of 1.9167, 1.9167, and 1.7500. The complete method ranks first in the ablation study and second in the engineering tests, with feasible solutions in all runs. These results demonstrate that DSD-LI-PO provides a consistent and targeted improvement over PO while maintaining competitive performance against strong non-PO optimizers.

Keywords: Parrot Optimizer; swarm intelligence; dynamic swarm dimension hybridization; lens imaging; constrained engineering optimization

1. Introduction

Constrained engineering optimization often involves nonlinear objectives, coupled variables, irregular feasible regions, and inequality constraints, which can make gradient-based methods difficult to apply when derivative information is unavailable or the search landscape is complex [1–4]. Bioinspired and evolutionary optimizers provide derivative-free alternatives by searching with populations of candidate solutions and have therefore been widely studied for numerical and engineering optimization [5–7]. Optimization is also increasingly embedded in simulation-based engineering workflows; for example, IROSim integrates CAD-based robotic task planning with trajectory and time optimization [8]. Their performance, however, still depends strongly on how information is exchanged within the population and how promising solutions are refined.

The Parrot Optimizer (PO) is a recent swarm intelligence algorithm based on foraging, staying, communication, and fear responses [9]. Although recent studies have extended PO through different search mechanisms and applications, two issues remain relevant for



Academic Editor: Heming Jia

Received: 22 July 2026

Revised: 13 August 2026

Accepted: 18 August 2026

Published: 20 August 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

single-objective numerical and constrained engineering optimization. First, the original behavioral rules mainly update solution vectors as a whole. When different variables evolve at different rates, these updates may disturb components that already contain useful information, especially on multimodal, hybrid, nonseparable, or constrained landscapes. Second, PO has no explicit mechanism that generates an additional elite-derived candidate when the global best solution stagnates. This may limit refinement accuracy in the later search stage.

To address these issues, DSD-LI-PO integrates two complementary mechanisms into the original PO framework. Dynamic swarm dimension hybridization (DSD) performs selective componentwise inheritance from the current best solution, allowing useful dimensional information to be exchanged without replacing the whole candidate vector. Variable precision lens imaging (LI) is activated after a predefined period of stagnation and generates a contracted candidate from the current best solution for greedy refinement and population injection. The proposed method therefore targets component disruption and stagnation separately while retaining the basic behavioral structure of PO. Detailed relationships with existing PO variants, componentwise search, and lens imaging methods are reviewed in Section 2.

The main contributions of this study are summarized as follows:

- A DSD strategy is introduced to perform variable-level information exchange within the PO framework and reduce unnecessary disturbance of well-evolved components caused by whole-vector updates.
- A stagnation-triggered LI strategy is developed to generate a contracted elite-derived candidate and provide an additional refinement opportunity when the current best solution stops improving.
- DSD-LI-PO is evaluated through the 2017 IEEE Congress on Evolutionary Computation (CEC2017) benchmark suite for single-objective bound-constrained real-parameter numerical optimization [10], dimensional scalability tests, ablation analysis, and five constrained engineering design problems. Average ranks, convergence metrics, runtime, Wilcoxon rank sum tests, and Cliff's delta are used to assess accuracy, convergence behavior, computational cost, and statistical significance.

The remainder of this article is organized as follows. Sections 2–4 review related work, summarize the baseline PO, and present DSD-LI-PO. Sections 5–7 describe the experimental evaluation and constrained engineering applications, followed by the Discussion and Conclusions.

2. Related Work

2.1. Parrot Optimizer and Its Recent Variants

PO was introduced as a bioinspired swarm optimizer that models several behavioral patterns of parrots, including foraging, staying, communication, and fear responses [9]. Because it is recent and structurally simple, PO has attracted several follow-up extensions. For example, the multiobjective Parrot Optimizer extends PO by introducing external archive management and Pareto-dominance-based selection for multiobjective problems [11]. The chaotic Parrot Optimizer incorporates chaotic maps to strengthen diversification and reduce premature convergence [12]. SNCPO integrates competitive swarm learning and salp navigation into PO for global numerical optimization and extreme learning machine training [13]. IPO improves PO for global optimization and multilayer perceptron classification by modifying several behavioral search components [14]. Other recent variants modify PO for wireless sensor network node deployment or related applications by introducing new search phases, local restart components, or problem-specific adaptation mechanisms [15].

These studies show that PO is an active research topic, but their design emphases differ from the present work. Some variants focus on multiobjective archiving, chaotic initialization, competitive population partitioning, neural-network training, or domain-specific deployment. In contrast, DSD-LI-PO focuses on single-objective numerical and constrained engineering optimization and specifically targets two issues in the search behavior of the baseline PO: disturbance of useful components caused by whole-vector updates and insufficient elite refinement after stagnation. Table 1 summarizes the representative recent PO variants and their main distinctions from DSD-LI-PO.

Table 1. Comparison of representative recent PO variants and the present method.

Method	Main Modification	Primary Focus	Distinction from DSD-LI-PO
Multi-objective PO [11]	External archive and Pareto-based selection	Multiobjective global and engineering optimization	Emphasizes Pareto management rather than component-wise inheritance or stagnation-triggered elite refinement
Chaotic PO [12]	Chaotic search mechanisms	Diversification for complex optimization and image segmentation	Strengthens search diversity through chaotic behavior
SNCPPO [13]	Competitive swarm learning and salp navigation	Global numerical optimization and extreme learning machine training	Uses a hybrid population-learning structure rather than the DSD-LI mechanism
IPO [14]	Modified PO behavioral search components	Global optimization and multi-layer perceptron classification	Provides a recent PO-based comparator with a different behavioral redesign
DSD-LI-PO	Componentwise DSD and stagnation-triggered LI	Single-objective numerical and constrained engineering optimization	Targets whole-vector component disturbance and stagnant elite refinement within the original PO framework

2.2. Componentwise and Elite-Guided Search Mechanisms

Componentwise information exchange is widely used in evolutionary and swarm-based optimization. Differential evolution introduced mutation and crossover as variable-level recombination operations for global optimization [16]. Variable-interaction analysis and dimension-oriented inheritance further show that optimization can benefit from identifying, grouping, or selectively inheriting informative variables [17,18]. In swarm optimization, particle swarm optimization and dimension-selection mechanisms also reflect the importance of dimensionwise information preservation and update control [19,20]. Recent adaptive differential evolution studies provide further evidence that dynamic perturbation and adaptive strategy design can improve variable-level search behavior [21,22]. These studies suggest that variable-level operations can be useful when different dimensions exhibit different search requirements.

The DSD mechanism follows this general componentwise motivation, but it is designed within the PO behavioral update framework. Instead of introducing a full differential mutation operator or replacing the PO search rules, DSD uses a two-stage selection rule. An individual-level gate first determines whether hybridization is performed, and a dimension-level mask then decides which components inherit information from the current best solution. The decreasing probabilities reduce the inheritance frequency in the later stage, which helps limit unnecessary dependence on the current best solution.

2.3. Opposition-Based Learning and Lens Imaging Refinement

Opposition-based learning (OBL) generates complementary candidate solutions by considering opposite positions in the search space [23,24]. OBL and its adaptive variants have been used to improve initialization, population diversity, and search-space adaptation in different metaheuristic algorithms [25]. Lens imaging opposition-based learning further modifies the opposite-position idea by introducing a scaling factor so that the generated candidate can be dynamically adjusted [26].

The LI mechanism in this study differs from repeatedly applying full-range opposition during the entire search process. It is activated only when the global best solution has not improved for a predefined number of consecutive iterations. The generated candidate is a contracted lens imaging point controlled by a nonlinear scale factor, and it is evaluated through greedy selection before being injected into the population. Therefore, LI is used as a stagnation-triggered elite refinement mechanism rather than a general random diversification operator.

2.4. Metaheuristics for Constrained Engineering Optimization

Constrained engineering optimization remains a common application area for metaheuristic algorithms because many design models include nonlinear constraints, mixed variable types, and irregular feasible regions [1,3]. Classical constrained benchmarks such as pressure vessel design, welded beam design, tension/compression spring design, speed reducer design, and cantilever beam design are frequently used to compare algorithmic feasibility, robustness, and search accuracy under controlled settings.

However, these benchmark problems should be interpreted carefully. They provide standardized constrained test cases rather than direct evidence of industrial deployment. For this reason, the present study uses them to examine whether DSD-LI-PO can maintain feasible and competitive behavior under constrained benchmark settings, while the Discussion section explicitly distinguishes benchmark-level validation from large-scale simulation-based or physical engineering applications.

3. Baseline Parrot Optimizer

3.1. Basic Model of PO

PO [9] is a swarm intelligence algorithm inspired by several social behaviors of parrots. In each iteration, individuals are updated through one of four behavioral modes: foraging, staying, communication, and fear response. These behaviors provide different movement patterns for global exploration, local perturbation, information exchange, and escape from unfavorable regions.

Similar to other population-based metaheuristic algorithms, PO starts with a population of candidate solutions randomly generated within the given search boundaries:

$$X_i^0 = lb + r(ub - lb), \quad r \sim U(0, 1), \quad (1)$$

where lb and ub are the lower and upper boundary vectors, respectively. The position of individual i at iteration t is denoted as

$$X_i^t = (x_{i,1}^t, x_{i,2}^t, \dots, x_{i,D}^t), \quad (2)$$

where D is the problem dimension.

Table 2 summarizes the four behavioral modes instead of repeating all original PO update equations. This compressed presentation keeps the baseline mechanism clear while leaving the main methodological focus on the two modifications introduced in DSD-LI-PO.

Table 2. Summary of the four behavioral modes in the baseline PO.

Behavior	Main Search Role	Search Implication
Foraging	Movement guided by the current best solution, stochastic perturbation, and population information	Supports exploration of promising regions but updates the solution vector as a whole
Staying	Local perturbation around the current search state and best-solution information	Provides local variation while retaining stochastic movement
Communication	Information exchange based on population mean or decaying random movement	Maintains swarm-level interaction but still uses whole-vector operations
Fear response	Escape-oriented movement related to the current best solution and time-varying terms	Helps reduce premature aggregation but may also disturb well-evolved components

3.2. Limitations of PO

Although PO uses four behavioral patterns to balance exploration and exploitation, its update rules are mainly implemented in a whole-vector manner. This design is simple and efficient, but it can be less suitable when different variables evolve at different rates or have different constraint effects. In multimodal, hybrid, nonseparable, or constrained landscapes, some dimensions of an individual may already contain useful information, whereas others may still require broader search. A whole-vector movement may therefore overwrite useful components when the entire individual is perturbed by poorly evolved dimensions or stochastic search terms [17,20,27].

A second limitation is the lack of an explicit elite-based candidate generation mechanism under stagnation. In the later search stage, PO mainly relies on its original behavioral updates to refine candidate solutions. These updates do not generate an additional contracted complementary candidate for the current best solution. As a result, the algorithm may converge slowly near promising regions or retain residual errors when higher convergence accuracy is required.

These limitations motivate the two modifications developed in Section 4: DSD introduces selective componentwise information exchange after the original PO update, whereas LI adds a stagnation-triggered candidate-generation step for the current best solution.

4. Designed DSD-LI-PO

4.1. Motivation and Overall Framework

Based on the limitations above, DSD-LI-PO integrates two complementary mechanisms into the original PO framework. After each PO behavioral update, DSD applies a time-dependent individual gate and dimension mask so that only selected components inherit information from the current global best solution. LI acts separately: when the best solution has not improved for L_s consecutive iterations, it generates a contracted lens imaging candidate, evaluates it greedily, and injects it into the population. Figure 1 shows the resulting workflow and the correspondence between the two identified limitations and their respective modifications.

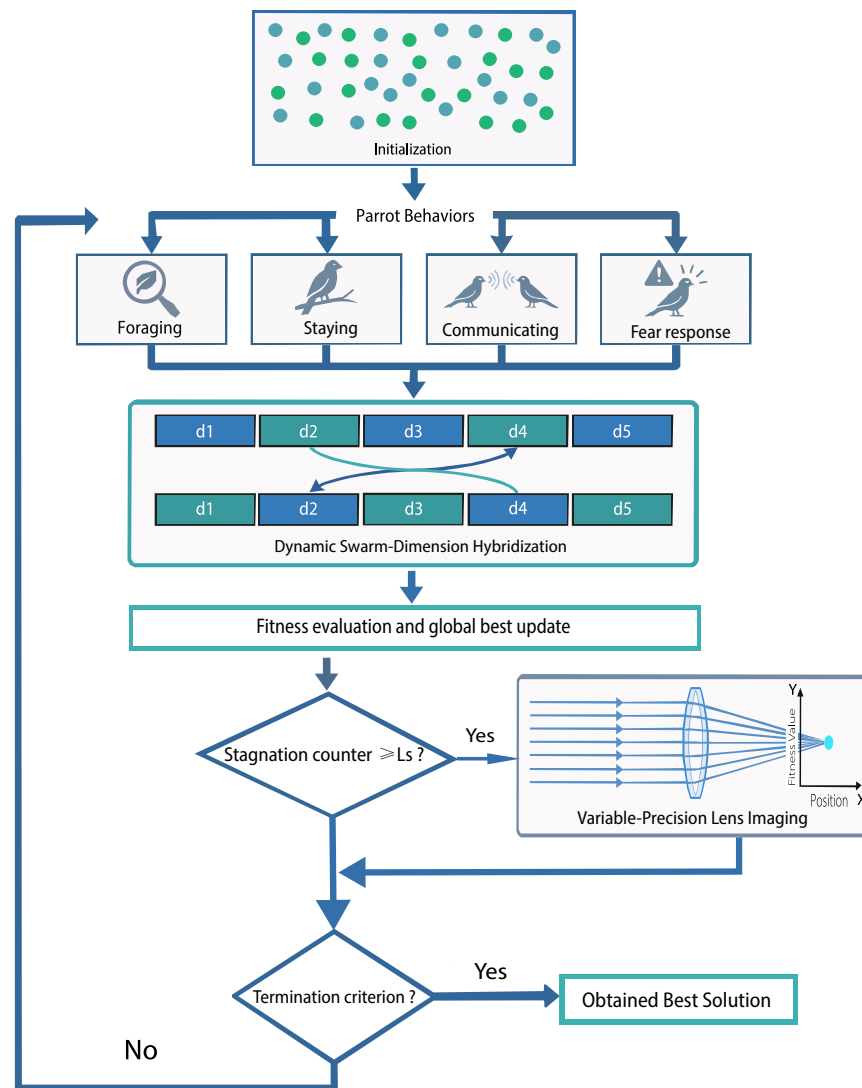


Figure 1. Overall framework of the designed DSD-LI-PO algorithm with componentwise hybridization and stagnation-triggered lens imaging refinement.

4.2. Dynamic Swarm Dimension Hybridization Strategy

In high-dimensional optimization, different dimensions of the same individual may not evolve synchronously [17,28]. Some dimensions may already contain useful information, whereas others may still need further exploration. For whole vector update rules, the whole vector is modified simultaneously. Such a mechanism is simple, but it may also disturb well-evolved dimensions when the update direction is affected by poorly evolved dimensions. This problem becomes more evident on hybrid or nonseparable functions, where variable interaction makes the contribution of each dimension uneven.

To reduce this componentwise interference, DSD introduces a selective inheritance mechanism. Let $\tilde{X}_i^{t+1}$ denote the candidate solution obtained after the original PO behavioral update. DSD first uses an individual-level hybridization gate controlled by $CR(t)$. If this gate is activated, a component mask is generated for the dimensions of the individual. The selected dimensions inherit the corresponding components from the current global best solution, whereas the remaining dimensions keep the values produced by the original PO update. Related componentwise inheritance and crossover ideas have also appeared in adaptive evolutionary search [18,20–22].

The individual-level hybridization gate is controlled by a linearly decreasing recombination probability:

$$CR(t) = CR_{\max} - (CR_{\max} - CR_{\min}) \frac{t}{T_{\max}}, \quad (3)$$

where $CR_{\max}$ and $CR_{\min}$ denote the initial and final recombination probabilities, respectively, and $T_{\max}$ is the maximum number of iterations. In this study, $CR_{\max} = 0.4$ and $CR_{\min} = 0.05$.

When the hybridization gate is activated, the component mask is controlled by a dimension selection probability:

$$p_d(t) = 0.2 \left(1 - \frac{t}{T_{\max}} \right). \quad (4)$$

Based on $CR(t)$ and $p_d(t)$, the componentwise hybridization rule is defined as

$$X_{i,j}^{t+1} = \begin{cases} X_{best,j}^t, & \text{if } r_i < CR(t) \text{ and } r_{i,j} < p_d(t), \\ \tilde{X}_{i,j}^{t+1}, & \text{otherwise,} \end{cases} \quad (5)$$

where $\tilde{X}_{i,j}^{t+1}$ denotes dimension j of the candidate solution obtained after the original PO update, $X_{best,j}^t$ denotes dimension j of the current global best solution, and r_i and $r_{i,j}$ are random numbers uniformly distributed in $[0, 1]$.

The decreasing designs of $CR(t)$ and $p_d(t)$ control how often information from the current global best solution is inherited. Early in the search, component exchange is more frequent, whereas both probabilities decrease as the search proceeds. This reduces explicit best-solution inheritance in later iterations and preserves the components generated by the original PO update. As illustrated in Figure 2, DSD does not replace the whole solution vector; it transfers selected components through a two-stage rule while leaving the remaining dimensions unchanged.

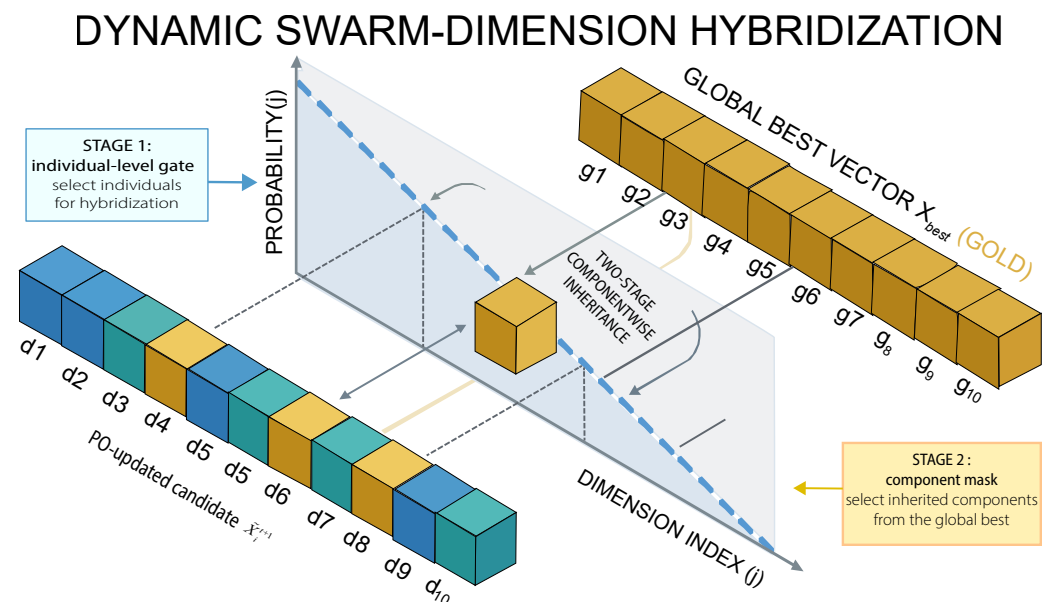


Figure 2. Two-stage componentwise information exchange in the DSD strategy.

The amount of explicit best-solution inheritance can be quantified from Equations (3) and (4). For a particular component, the unconditional inheritance probability is

$$P_{\text{inh}}(t) = CR(t)p_d(t). \quad (6)$$

With the default settings, $P_{\text{inh}}(0) = 0.08$ and $P_{\text{inh}}(0.5T_{\text{max}}) = 0.0225$, while it approaches zero as t approaches T_{max} . For the main setting $D = 30$, these probabilities correspond to expected inherited component counts of 2.4 at the beginning and 0.675 at the midpoint of the search, with the expected count approaching zero near the end. Thus, DSD uses the global best solution selectively rather than imposing persistent full-vector attraction, and its explicit dependence on best-solution inheritance decreases over time.

4.3. Variable Precision Lens Imaging

OBL [23,24] is often used to generate a complementary candidate solution from the opposite position of a given solution. However, the standard opposite solution is usually calculated with respect to fixed search boundaries. When the search process enters the later stage, directly using a full opposite solution may introduce an excessively large displacement and become less suitable for controlled elite solution updating [25].

To address this issue, this study introduces a stagnation-triggered variable precision lens imaging strategy. The proposed LI strategy generates a contracted lens imaging opposite candidate based on the current global best solution when the search process shows no improvement for several consecutive iterations. Let L_s denote the stagnation threshold. In this study, $L_s = 15$. When the current global best solution is not improved for L_s consecutive iterations, LI is activated. Let o_j be the center of dimension j , which is calculated as

$$o_j = \frac{lb_j + ub_j}{2}. \quad (7)$$

The dynamic scaling factor is defined as

$$k(t) = \left(1 + \left(\frac{t}{T_{\text{max}}}\right)^\gamma\right)\eta, \quad (8)$$

where γ is the nonlinear adjustment coefficient and η is the baseline scaling factor. In this study, $\gamma = 0.5$ and $\eta = 10$. The lens imaging candidate of the current global best solution is calculated as

$$X'_{\text{best},j} = o_j + \frac{o_j - X_{\text{best},j}^t}{k(t)} = \frac{lb_j + ub_j}{2} + \frac{\frac{lb_j + ub_j}{2} - X_{\text{best},j}^t}{k(t)}, \quad (9)$$

where $X_{\text{best},j}^t$ denotes dimension j of the current global best solution, and lb_j and ub_j are the lower and upper bounds of dimension j , respectively.

When $k = 1$, Equation (9) degenerates into the standard opposite solution. In the implementation, $k(t) > 1$, and therefore the generated candidate is a contracted opposite solution located between the search space center and the standard opposite point. As $k(t)$ increases with the iteration process, the distance between the generated lens imaging candidate and the search space center decreases. Therefore, LI does not repeatedly perform full-range opposite search. Instead, it provides a dynamically contracted complementary candidate for the current global best solution [26].

Under the default setting $\gamma = 0.5$ and $\eta = 10$, $k(t)$ increases from approximately 10 at the beginning of the search to 20 at the end. Accordingly, the offset of the LI candidate from the search-space center decreases from about 10% to 5% of the mirrored offset of the current best solution. This schedule makes LI a progressively more contracted complementary search step when stagnation occurs.

After the lens imaging candidate is generated, its fitness value is evaluated. If this candidate has a better fitness value than the current global best solution, it replaces the latter through greedy selection. Otherwise, the current global best solution is retained. In addition, the generated LI candidate is injected into the population by replacing the current worst individual. This operation allows the elite-derived complementary candidate to participate in the subsequent population update process.

Figure 3 illustrates the geometry of this operation. Because the candidate is contracted relative to standard opposition and generated only after stagnation, LI serves as an occasional complementary refinement step rather than a full-range opposition operator applied throughout the search.

Variable-Precision Lens Imaging Candidate Generation

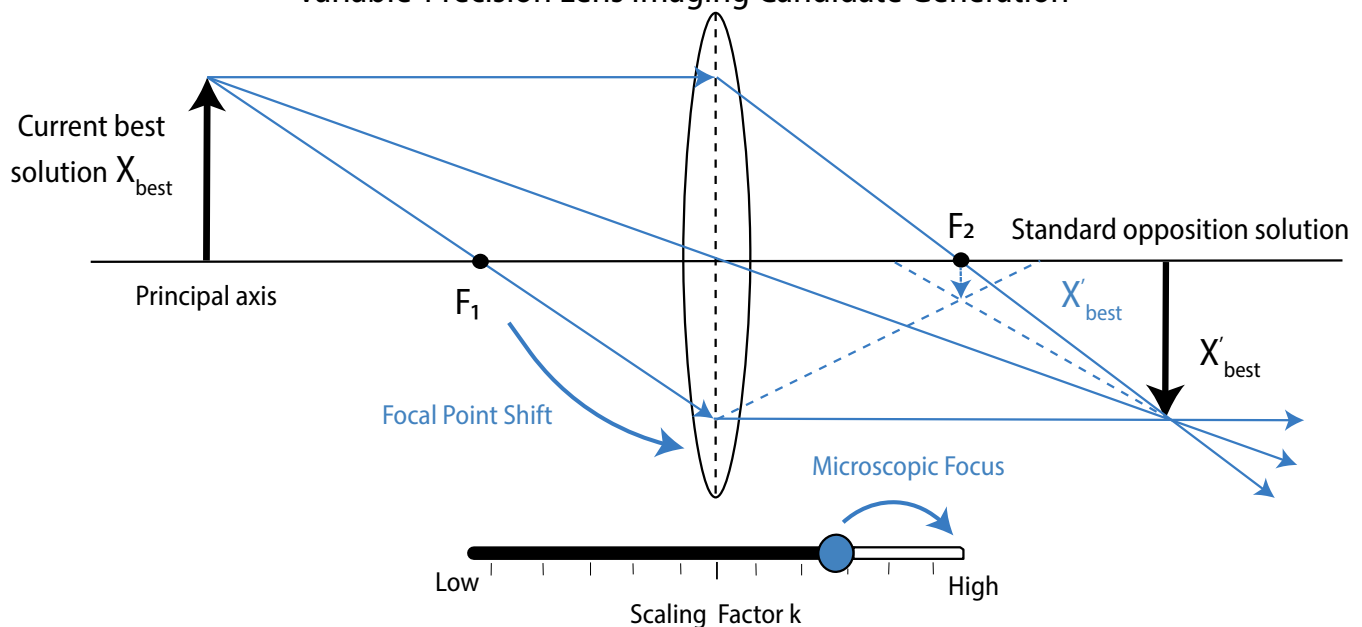


Figure 3. Geometric illustration of the contracted lens imaging candidate generation in LI.

4.4. Synergistic Mechanism and Algorithmic Procedure

DSD and LI operate at different stages of the search. After the original PO update, DSD selectively transfers components from the current best solution through the individual-level gate and dimension-level mask. LI is activated only after L_s consecutive non-improving iterations and generates one contracted lens imaging candidate for additional elite refinement. Thus, DSD controls componentwise information exchange during routine population updating, whereas LI provides a stagnation-triggered refinement step. Algorithm 1 summarizes their integration in the complete DSD-LI-PO procedure.

Algorithm 1 DSD-LI-PO

Require: $f, lb, ub, N, D, T_{\max}$, and controls $(CR_{\max}, CR_{\min}, \gamma, \eta, L_s)$
Ensure: Best solution X_{best} and best fitness $f(X_{best})$

- 1: Initialize population $X_i^0, i = 1, 2, \dots, N$, within $[lb, ub]$
- 2: Evaluate all individuals and determine X_{best}
- 3: Set stagnation counter $c_s \leftarrow 0$
- 4: **for** $t = 1$ to $T_{\max}$ **do**
- 5: **for** $i = 1$ to N **do**
- 6: Generate $\tilde{X}_i^{t+1}$ using one original PO behavioral update rule
- 7: Compute $CR(t)$ by Equation (3) and $p_d(t)$ by Equation (4)
- 8: **if** $r_i < CR(t)$ **then**
- 9: **for** $j = 1$ to D **do**
- 10: **if** $r_{i,j} < p_d(t)$ **then**
- 11: $X_{i,j}^{t+1} \leftarrow X_{best,j}^t$
- 12: **else**
- 13: $X_{i,j}^{t+1} \leftarrow \tilde{X}_{i,j}^{t+1}$
- 14: **end if**
- 15: **end for**
- 16: **else**
- 17: $X_i^{t+1} \leftarrow \tilde{X}_i^{t+1}$
- 18: **end if**
- 19: Apply boundary correction to X_i^{t+1} and evaluate fitness
- 20: **end for**
- 21: Update X_{best} using the best individual in the population
- 22: **if** X_{best} is improved **then**
- 23: $c_s \leftarrow 0$
- 24: **else**
- 25: $c_s \leftarrow c_s + 1$
- 26: **end if**
- 27: **if** $c_s \geq L_s$ **then**
- 28: Generate the LI candidate X'_{best} using Equation (9)
- 29: Apply boundary correction and evaluate X'_{best}
- 30: **if** $f(X'_{best}) < f(X_{best})$ **then**
- 31: $X_{best} \leftarrow X'_{best}$
- 32: **end if**
- 33: Replace the worst individual in the population with X'_{best}
- 34: $c_s \leftarrow 0$
- 35: **end if**
- 36: **end for**
- 37: **return** X_{best} and $f(X_{best})$

4.5. Computational Complexity

The computational cost of DSD-LI-PO consists of population initialization, position updating, the DSD strategy, the stagnation-triggered LI strategy, and fitness evaluation. Let N denote the population size, D denote the problem dimension, $T_{\max}$ denote the maximum number of iterations, and C_f denote the cost of one fitness evaluation. The initialization of the population requires $O(ND)$ time, and the initial fitness evaluation requires $O(NC_f)$ time.

In each iteration, the original PO behavioral update rules perform vector operations for all individuals and require $O(ND)$ time. The DSD strategy performs individual-level gate checking, componentwise probability comparison, and assignment operations. Its time complexity is also $O(ND)$ in each iteration. The LI strategy is activated only when the stagnation condition is satisfied. If LI is triggered, generating the contracted lens imaging candidate requires $O(D)$ time, and evaluating this candidate introduces one additional fitness evaluation.

Let N_{LI} denote the number of LI activations during the whole search process, where $0 \leq N_{LI} \leq T_{\max}$. The total computational complexity can be expressed as

$$O(ND + NC_f) + O(T_{\max}(ND + NC_f)) + O(N_{LI}(D + C_f)). \quad (10)$$

Since $N_{LI} \leq T_{\max}$ and the additional LI evaluation is applied to only one candidate when triggered, the dominant asymptotic order remains

$$O(T_{\max}(ND + NC_f)). \quad (11)$$

When the fitness evaluation cost is proportional to the problem dimension, this reduces to $O(T_{\max}ND)$. Thus, DSD and LI add componentwise operations and occasional elite candidate evaluations, while keeping the same dominant asymptotic order as the original PO.

5. Experimental Design and Evaluation Protocol

This section describes the numerical evaluation protocol for DSD-LI-PO [29]. The evaluation comprises the main CEC2017 comparison, convergence analysis, dimensional scalability tests, parameter sensitivity analysis, and ablation experiments. The constrained engineering benchmarks are evaluated separately in Section 7.

5.1. Benchmark Functions and Compared Algorithms

The IEEE CEC2017 benchmark functions [10] are used to evaluate the numerical optimization performance of DSD-LI-PO. The benchmark functions provide different types of search landscapes, including unimodal, multimodal, hybrid, and composition characteristics, and are therefore suitable for examining convergence accuracy, local optimum avoidance, and robustness on complex search spaces.

DSD-LI-PO is compared with seven representative optimizers, including the original PO, improved Parrot Optimizer (IPO) [14], linear population size reduction success history based adaptive differential evolution (LSHADE) [30], Golden Jackal Optimization (GJO) [31], Dung Beetle Optimizer (DBO) [32], Whale Optimization Algorithm (WOA) [33], and Harris Hawks Optimization (HHO) [34]. The original PO is used as the direct baseline for evaluating the effect of the DSD and LI strategies. IPO is included as a recent PO variant to strengthen the comparison with PO-based methods. The IPO implementation follows the published mathematical formulation and algorithmic description in its original article. The other algorithms are selected as representative competitors with different search mechanisms, including differential evolution, swarm intelligence, and recently proposed population-based optimization methods.

All benchmark functions are treated as minimization problems. Final best fitness values are used for the main numerical comparison and ranking. For convergence visualization, the deviation from the best observed final fitness on each function is plotted with a logarithmic vertical axis, providing a common non-negative reference for the displayed curves. The same deviation sequence is used to derive the convergence-speed metrics.

5.2. Parameter Settings and Evaluation Criteria

For the main CEC2017 comparison, the population size was set to $N = 30$, the problem dimension to $D = 30$, and the maximum number of iterations to 1000. The search range was $[-100, 100]$, and each algorithm was independently executed 30 times on each function. Algorithm-specific parameters followed the original publications or commonly used settings.

All experiments were implemented under Linux 5.15.0 using Python 3.10.8, NumPy 1.24.1, and SciPy 1.15.3. Computations were performed on an AMD EPYC 9654 CPU-based AutoDL cloud instance with 32 vCPU cores and 60 GB memory.

Performance was summarized by the mean and standard deviation of the final best fitness values and by tie-aware average ranks. Pairwise differences between DSD-LI-PO and each comparator were evaluated with the Wilcoxon rank sum test [35,36] at the 0.05 significance level, together with Cliff's delta as the effect-size measure [37]. The symbols "+", "=", and "−" denote significantly better, statistically similar, and significantly worse performance of DSD-LI-PO, respectively. Complete mean and standard deviation results are reported in Supplementary Table S1. Functionwise ranks, pairwise statistical tests, runtime summaries, detailed scalability results, sensitivity results, ablation results, and engineering validation statistics are provided in Supplementary Data S1.

6. Results

6.1. Overall Comparison on CEC2017

The expanded CEC2017 comparison results are summarized in Table 3, which reports the tie-aware average rank, overall rank, and average runtime per independent run.

Table 3. Overall ranking and computational cost on the CEC2017 benchmark suite.

Algorithm	Average Rank	Overall Rank	Runtime (s)
HHO	1.7586	1	0.5912 ± 0.0833
DSD-LI-PO	1.8966	2	0.9271 ± 0.0854
WOA	2.8448	3	0.7132 ± 0.0821
GJO	4.2414	4	1.8349 ± 0.1024
LSHADE	5.1552	5	2.4669 ± 0.1167
PO	5.2414	6	0.8877 ± 0.0859
IPO	6.8966	7	1.0836 ± 0.0910
DBO	7.9655	8	0.1509 ± 0.0163

Note: Average ranks are calculated from functionwise mean final fitness values with average ranks assigned to exact ties. Runtime is reported as mean $\pm$ standard deviation over independent runs.

Table 3 shows that HHO obtains the lowest average rank of 1.7586, followed closely by DSD-LI-PO with an average rank of 1.8966. DSD-LI-PO remains well ahead of the original PO (5.2414) and IPO (6.8966), which supports the effectiveness of the proposed modifications within the PO family. Its runtime is slightly higher than that of PO because of the componentwise operations and occasional LI evaluations, but it remains lower than the runtimes of IPO, GJO, and LSHADE under the same computing environment.

Complete functionwise mean $\pm$ standard deviation results for all compared algorithms are provided in Supplementary Table S1. These data complement the rank and statistical summaries with the absolute numerical performance on each CEC2017 function.

The pairwise statistical and effect-size summaries are reported in Table 4. This table complements the rank-based comparison by showing whether the observed differences are statistically reliable and by indicating their magnitude.

The statistical results confirm the main trend in Table 3. DSD-LI-PO is clearly better than IPO, DBO, and GJO in the pairwise summary, and it also shows strong advantages over PO and LSHADE. Compared with WOA, DSD-LI-PO shows only a limited number of significant wins, and most comparisons are statistically similar. Compared with HHO, DSD-LI-PO is mostly statistically similar and is worse in a small number of cases. These results support a cautious interpretation: DSD-LI-PO improves PO and performs strongly overall, but it does not uniformly dominate all competitors on every landscape type.

Table 4. Pairwise statistical comparison and effect-size summary on the CEC2017 benchmark suite.

Compared Algorithm	DSD-LI-PO +/=/−	Mean Cliff's δ	Median Cliff's δ
DBO	29/0/0	−1.0000	−1.0000
GJO	29/0/0	−0.8470	−0.8667
HHO	0/26/3	0.1011	0.0000
IPO	29/0/0	−1.0000	−1.0000
LSHADE	28/0/1	−0.9173	−1.0000
PO	28/1/0	−0.9338	−1.0000
WOA	5/23/1	−0.0577	0.0000

Note: Cliff's δ is computed using the DSD-LI-PO samples relative to the compared algorithm samples. Negative values therefore indicate that DSD-LI-PO tends to obtain smaller final fitness values.

6.2. Results on Representative Benchmark Functions

To examine convergence behavior more directly, six representative CEC2017 functions are selected from different landscape types: F1, F8, F16, F20, F24, and F30. The convergence curves of all compared algorithms on these functions are shown in Figure 4.

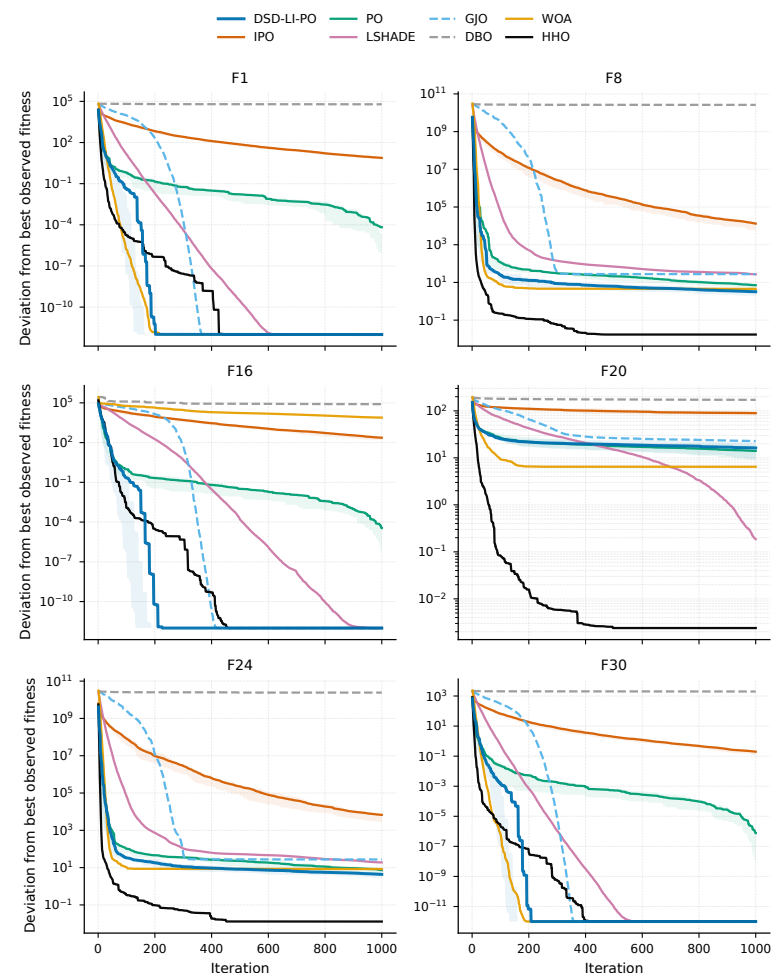


Figure 4. Convergence curves on six representative CEC2017 benchmark functions. Curves show the mean deviation from the best observed final fitness over 30 independent runs on a logarithmic vertical axis; the blue, orange, and green shaded bands correspond to DSD-LI-PO, IPO, and PO, respectively, and indicate the interquartile range (25th–75th percentiles).

Figure 4 shows that DSD-LI-PO reaches highly competitive final values on F1, F16, and F30, and remains among the leading algorithms on F8 and F24. HHO achieves the best final accuracy on F8, F20, and F24, which is consistent with the statistical summary in

Table 4. The F20 result also shows that the proposed method is not uniformly optimal on all hybrid landscapes. Therefore, the convergence curves support the overall conclusion that DSD-LI-PO provides a strong and stable improvement over PO, while some problem landscapes still favor other search mechanisms.

A convergence-speed summary based on the same representative functions is provided in Table 5. The area under the logarithmic deviation sequence (AUC) reflects both convergence speed and residual deviation during the search process. Smaller values indicate faster reduction of the deviation from the best observed final fitness.

Table 5. Convergence-speed summary on representative CEC2017 benchmark functions.

Algorithm	Mean AUC	Median AUC	Mean Final Deviation	Median Final Deviation
HHO	−6.2364	−6.1790	0.0054	0.0012
DSD-LI-PO	−4.7985	−4.8297	3.9698	1.6089
WOA	−3.4281	0.2102	1284.6700	5.5072
LSHADE	−2.4137	−1.6030	7.6996	0.0931
GJO	−2.3024	−2.4580	12.9539	11.3906
PO	−0.5695	−0.2869	4.7478	3.5733
IPO	3.1510	2.6470	3381.9877	161.5537
DBO	6.0043	4.8638	8.383×10^9	7.051×10^4

Note: AUC is computed from the logarithmic deviation sequence. Lower AUC and lower final deviation indicate better convergence behavior under this metric.

Table 5 indicates that HHO has the lowest AUC on the representative functions, while DSD-LI-PO obtains the second-lowest mean and median AUC values. This result is consistent with the convergence curves and prevents an overstatement that DSD-LI-PO is always the fastest converging method. A more accurate interpretation is that DSD-LI-PO provides competitive convergence speed while retaining the second-best overall average rank on the CEC2017 benchmark suite. The complete convergence curves of the main CEC2017 comparison are provided in Supplementary Figures S1–S3.

6.3. Scalability Analysis at Higher Dimensions

A separate dimensional scalability experiment was conducted on the six representative functions F1, F8, F16, F20, F24, and F30. The $D = 30$ results were taken from the main CEC2017 experiment, while additional experiments were performed at $D = 50$ and $D = 100$. DSD-LI-PO was compared with PO, IPO, and HHO. For the additional experiments, the population size ($N = 30$), maximum number of iterations (1000), search range ($[-100, 100]$), and number of independent runs (30) were kept the same as in the main experiment. The corresponding dimensional scalability results are summarized in Table 6.

Table 6. Additional dimensional scalability results on six representative CEC2017 benchmark functions.

Dimension	Average Rank				DSD-LI-PO +/−			DSD-LI-PO Runtime (s)
	DSD-LI-PO	PO	IPO	HHO	vs. PO	vs. IPO	vs. HHO	
30	1.9167	2.8333	4.0000	1.2500	5/1/0	6/0/0	0/3/3	0.9293
50	1.9167	2.8333	4.0000	1.2500	5/1/0	6/0/0	0/3/3	0.9423
100	1.7500	3.0000	4.0000	1.2500	5/1/0	6/0/0	0/3/3	1.0023

Note: Average ranks are recalculated among the four algorithms on the six representative functions, with average ranks assigned to exact ties. The +/− entries denote the numbers of functions on which DSD-LI-PO is significantly better than, statistically similar to, or significantly worse than the corresponding algorithm according to Wilcoxon rank sum tests at the 0.05 significance level. Runtime is the mean runtime per independent run averaged over the six functions.

DSD-LI-PO ranks second among the four algorithms at all three dimensions, with average ranks of 1.9167, 1.9167, and 1.7500 at $D = 30$, 50, and 100, respectively. Its pairwise statistical results are also unchanged across the three dimensions. DSD-LI-PO obtains

5/1/0 significant wins/ties/losses against PO and 6/0/0 against IPO at each dimension. Against HHO, the result is 0/3/3, showing statistical similarity on three functions and significantly worse performance on the other three.

The runtime of DSD-LI-PO increases from 0.9293 s at $D = 30$ to 0.9423 s at $D = 50$ and 1.0023 s at $D = 100$. Together with the stable rank and pairwise results, these data show that the advantage of DSD-LI-PO over PO and IPO is maintained within the tested dimensional range. However, HHO retains the best average rank of 1.2500 at all three dimensions and remains significantly better on three of the six functions. The scalability results therefore support the stability of the proposed PO improvement from 30 to 100 dimensions rather than uniform superiority over all competing search mechanisms.

6.4. Sensitivity Analysis of Key Control Parameters

To further investigate the robustness of the parameter settings used in DSD-LI-PO, a small-scale sensitivity analysis was conducted on three key control components: the DSD probability schedule, the nonlinear scaling factor of LI, and the stagnation threshold used for LI activation. The default setting follows the parameter configuration adopted in the main experiments, while each sensitivity variant modifies one component around the default value. The analysis was performed on the same six representative CEC2017 functions using the same population size, dimension, and maximum iteration settings as the main comparison.

Table 7 shows that the tested configurations remain within a competitive performance range, while the DSD probability schedule has a more noticeable influence than the LI scaling factor. The DSD-Low setting achieves the lowest average rank and mean final deviation among the tested schedules, whereas DSD-High is less favorable on the representative functions. Changing γ produces comparatively small differences, and $L_s = 10$ gives the lowest mean AUC among the tested stagnation thresholds. The sensitivity experiment is used as a diagnostic analysis rather than a post hoc tuning stage; therefore, the default configuration fixed for the main comparison is retained instead of being reselected from the representative-function results. The DSD schedule and stagnation threshold remain useful directions for future adaptive control.

Table 7. Sensitivity analysis of key control parameters on representative CEC2017 benchmark functions.

Group	Variant	Changed Setting	Avg. Rank	Mean AUC	Mean Final dev.	Runtime (s)
Baseline	Default	$CR_{\max} = 0.4, CR_{\min} = 0.05, p_{d,0} = 0.2, \gamma = 0.5, L_s = 15$	2.3333	−4.8107	3.6618	0.9530
DSD schedule	DSD-Low	$CR_{\max} = 0.3, CR_{\min} = 0.03, p_{d,0} = 0.15$	1.1667	−4.8674	2.9498	0.9310
DSD schedule	DSD-High	$CR_{\max} = 0.5, CR_{\min} = 0.10, p_{d,0} = 0.25$	3.1667	−4.7836	4.2024	0.9337
LI scaling	K-Fast	$\gamma = 1.0, \eta = 10$	2.5000	−4.7940	3.8731	0.9277
LI scaling	K-Slow	$\gamma = 0.3, \eta = 10$	3.0000	−4.7780	3.9391	0.9247
Stagnation threshold	LS-Short	$L_s = 10$	2.1667	−5.0725	3.4386	0.9272
Stagnation threshold	LS-Long	$L_s = 20$	3.1667	−4.5588	3.7574	0.9261

Note: The sensitivity analysis was conducted on F1, F8, F16, F20, F24, and F30 with 10 independent runs for each setting. The default setting is the parameter configuration used in the main experiments. $p_{d,0}$ denotes the initial coefficient of the DSD dimension selection probability. AUC is computed from the logarithmic deviation sequence. Lower average rank, AUC, and final deviation indicate better performance under the corresponding metric. Full functionwise sensitivity results are provided in Supplementary Data S1. Avg. denotes the average value, and dev. denotes the standard deviation.

6.5. Ablation Study

To examine the individual effects of the two strategies, the ablation study compares four variants: PO, DSD-PO, LI-PO, and DSD-LI-PO. PO denotes the original algorithm. DSD-PO introduces only the dynamic swarm dimension hybridization strategy.

LI-PO introduces only the variable precision lens imaging strategy. DSD-LI-PO integrates both strategies.

Table 8 reports the overall ablation results of PO, DSD-PO, LI-PO, and DSD-LI-PO. The complete DSD-LI-PO obtains the best average rank of 1.5517 and the lowest mean and median absolute errors. LI-PO ranks second with an average rank of 1.5862, while DSD-PO improves the original PO from 3.5172 to 3.3448.

Table 8. Summary of ablation results on the considered CEC2017 benchmark functions.

Algorithm	Avg Rank	Mean Absolute Error	Median Absolute Error	+/=/−
DSD-LI-PO	1.5517	0.6099	0.5011	−
LI-PO	1.5862	0.7883	0.8421	1/28/0
DSD-PO	3.3448	0.7854	0.6892	28/1/0
PO	3.5172	0.9220	0.8475	28/1/0

Note: The +/=/− column reports the number of functions on which DSD-LI-PO is significantly better than, statistically similar to, or significantly worse than each compared variant, according to Wilcoxon rank sum tests using the results from 30 independent runs at the 0.05 significance level.

The separate variants confirm that both mechanisms contribute to the improvement over PO. LI-PO remains close to the complete method in average rank, while DSD-PO also improves the baseline. The combined DSD-LI-PO retains the best average rank and the lowest overall error indicators; the different roles of DSD and LI are discussed further in Section 8.

The +/=/− values in Table 8 are based on Wilcoxon rank sum tests using results from independent runs. Detailed functionwise ablation results and statistical tests are provided in Supplementary Data S1, and the complete ablation convergence curves are provided in Supplementary Figure S4.

7. Constrained Engineering Design Applications

To further examine the applicability of DSD-LI-PO to constrained engineering optimization tasks, five classical engineering design problems are used: pressure vessel design (PVD) [38], welded beam design (WBD) [31], tension/compression spring design (TCSD) [31], speed reducer design (SRD) [31], and cantilever beam design (CBD) [38]. These problems involve nonlinear objective functions, inequality constraints, and different types of design variables. They are commonly used as engineering benchmarks to evaluate the search performance of metaheuristic algorithms under constrained settings.

7.1. Experimental Setting for Engineering Problems

Table 9 shows that the selected engineering problems cover different constrained design settings. PVD and SRD include mixed variable characteristics, while WBD, TCSD, and CBD are continuous constrained problems with different dimensionalities and constraint numbers. This problem set is therefore suitable for examining whether DSD-LI-PO can maintain feasible and competitive search behavior beyond unconstrained benchmark functions. In the engineering experiments, DSD-LI-PO is compared with PO, LSHADE, WOA, and HHO. For all algorithms, the population size is set to 30, the maximum number of iterations is set to 500, and each algorithm is independently run 30 times on each engineering problem. The lower and upper bounds of the design variables follow the standard definitions used in the implemented engineering benchmark functions.

Table 9. Characteristics of the selected engineering design problems.

Problem	Variable Type	Dimension	Constraints	Main Characteristic
PVD	Mixed	4	4	Discretized thickness variables and nonlinear constraints
WBD	Continuous	4	7	Low dimensional constrained structural design
TCSD	Continuous	3	4	Low dimensional constrained mechanical design
SRD	Mixed	7	11	Integer teeth variable and multiple nonlinear constraints
CBD	Continuous	5	1	Continuous structural optimization problem

Penalty-function approaches are a widely used class of constraint-handling methods in evolutionary optimization [39–41]. In this study, a quadratic exterior penalty function with a fixed penalty coefficient is used to handle all inequality constraints. For a constrained problem with objective function $f(x)$ and inequality constraints $g_i(x) \leq 0$, the penalized objective function is defined as

$$F(x) = f(x) + \sum_i 10^{15} \max(0, g_i(x))^2. \quad (12)$$

Thus, feasible solutions keep their original objective values, whereas infeasible solutions receive a penalty proportional to the squared constraint violation. In the statistical comparison, the raw objective value is used for feasible runs, while the penalized fitness value is used for infeasible runs. The same penalized objective function and boundary settings are used for all compared algorithms. In PVD, the shell thickness and head thickness are discretized as multiples of 0.0625 by upward rounding before objective and constraint evaluation. In SRD, the number of teeth is treated as an integer variable. The complete mathematical formulations, including objective functions, constraints, and variable bounds, are provided in Supplementary Note S1.

7.2. Comparative Results

The statistical results of the compared algorithms on the five engineering design problems are summarized in Table 10.

Table 10. Statistical results of the compared algorithms on engineering design problems.

Problem	Algorithm	Best	Mean	Std	Worst
PVD	DSD-LI-PO	5336.6226	5473.0385	125.8022	5789.9920
	PO	5371.8484	5531.3702	127.0481	5874.1060
	LSHADE	5336.6181	5345.6968	20.8797	5408.5057
	WOA	5870.5549	9074.7903	3723.3180	24,670.1100
	HHO	5777.8948	9148.7288	2431.9700	15,191.5200
WBD	DSD-LI-PO	1.7339	1.8231	0.1408	2.4526
	PO	1.7287	1.8073	0.0881	2.2287
	LSHADE	1.7249	1.7249	3.34×10^{-16}	1.7249
	WOA	1.9307	3.1196	0.9830	5.7360
	HHO	1.7972	2.8570	0.7939	4.6367
TCSD	DSD-LI-PO	0.012728	0.013713	0.001151	0.017748
	PO	0.012766	0.013923	0.001684	0.017791
	LSHADE	0.012665	0.012666	1.18×10^{-6}	0.012671
	WOA	0.012667	0.014012	0.001238	0.017805
	HHO	0.012785	1.336609×10^{14}	1.336609×10^{14}	8.655466×10^{14}
SRD	DSD-LI-PO	2986.5950	3000.5087	10.4477	3026.6048
	PO	2993.2570	3019.1771	25.1772	3101.2347
	LSHADE	2985.8609	2985.8609	1.02×10^{-11}	2985.8609
	WOA	3102.9708	1.473756×10^{10}	7.936417×10^{10}	4.421267×10^{11}
	HHO	3609.2332	4.377840×10^{11}	7.084801×10^{11}	3.279169×10^{12}
CBD	DSD-LI-PO	1.340406	1.343453	0.002843	1.354895
	PO	1.340745	1.343344	0.003210	1.354896
	LSHADE	1.339956	1.339956	2.72×10^{-16}	1.339956
	WOA	1.796195	2.745464	0.717418	4.398584
	HHO	1.378011	1.591333	0.137923	2.026176

Note: Best, mean, standard deviation, and worst are computed from the comparison values. For feasible runs, the comparison value is the raw objective value; for infeasible runs, it is the penalized fitness value.

Table 10 reports the best, mean, standard deviation, and worst values obtained over 30 independent runs. The results show that DSD-LI-PO obtains feasible and competitive solutions on the selected engineering design problems. According to the average rank based on the reported mean values in Table 10, where infeasible runs are evaluated using the penalized fitness value, LSHADE ranks first on this group of engineering benchmarks, while DSD-LI-PO ranks second. This ranking indicates that DSD-LI-PO improves the engineering performance of the original PO, although it does not dominate all compared algorithms.

Compared with the original PO, DSD-LI-PO reduces the reported mean value on PVD, TCSD, and SRD. The improvement is particularly clear on SRD, where the reported mean value is reduced from 3019.1771 to 3000.5087. The Wilcoxon rank sum test also shows that the improvement of DSD-LI-PO over PO on SRD is statistically significant at the 0.05 level. On PVD and TCSD, DSD-LI-PO obtains lower mean values than PO, but the differences are not statistically significant. On WBD and CBD, the two algorithms produce similar results, with PO giving slightly lower mean values.

DSD-LI-PO also performs better than WOA and HHO on most engineering problems. In particular, WOA and HHO suffer from large penalized fitness values in some constrained cases, such as TCSD and SRD, because infeasible runs are evaluated using the penalized objective function. In contrast, DSD-LI-PO obtains feasible solutions in all independent runs on the five engineering problems. These results indicate that the proposed strategies improve the engineering applicability of PO, although they do not make DSD-LI-PO uniformly superior to all competitors.

Table 11 lists the best feasible design variables obtained by DSD-LI-PO. These solutions correspond to the best objective values reported for DSD-LI-PO in Table 10 and provide the feasible design vectors identified by the algorithm.

Table 11 shows that DSD-LI-PO can generate feasible design solutions for all five engineering problems under the same experimental setting. For PVD and SRD, the reported variables are the repaired design variables used in objective and constraint evaluation, which is consistent with the mixed variable treatment described in Section 7.1. Additional engineering validation statistics are provided in Supplementary Data S1.

Table 11. Best feasible design variables obtained by DSD-LI-PO on engineering design problems.

Problem	Best Objective Value	Design Variables
PVD	5336.622640	1.3125, 0.6250, 65.225262, 10.000000
WBD	1.733918	0.203491, 3.541172, 9.038247, 0.206089
TCSD	0.012728	0.050000, 0.317325, 14.043704
SRD	2986.594954	3.500031, 0.700000, 17.000000, 7.300000, 7.722029, 3.351401, 5.287087
CBD	1.340406	6.019353, 5.250677, 4.479897, 3.501562, 2.229377

8. Discussion

The expanded CEC2017 comparison provides a broader view of the proposed method because it includes both the original PO and the recent IPO variant. With average ranks assigned to exact ties, DSD-LI-PO ranks second overall and retains a clear statistical advantage over PO and IPO. HHO obtains the lowest average rank and remains stronger on several representative functions. The main contribution of DSD-LI-PO is therefore its consistent and statistically supported improvement of the PO search framework while remaining competitive with strong non-PO optimizers.

The dimensional tests show that DSD-LI-PO preserves its relative advantage over PO and IPO from 30 to 100 dimensions. HHO remains stronger on part of the representative

set, so the evidence supports stable performance over the tested dimensional range rather than unrestricted large-scale superiority.

The ablation results clarify the different contributions of the two mechanisms. DSD-PO improves on the original PO, indicating that selective componentwise inheritance improves the population-level information-exchange pattern. LI-PO achieves an average rank close to that of the complete method, showing that stagnation-triggered contracted refinement contributes strongly to final accuracy. DSD-LI-PO nevertheless obtains the best average rank and the lowest overall error indicators among the four variants, supporting the complementary use of DSD for componentwise cooperation and LI for stagnation relief.

The engineering benchmarks provide complementary evidence under constrained search. DSD-LI-PO obtains feasible solutions in all independent runs and improves the original PO on several problems, with a statistically significant improvement on speed reducer design. LSHADE remains the strongest method on this benchmark set, indicating that the relative performance of DSD-LI-PO still depends on the constraint structure and landscape characteristics of the problem.

The method is most suitable for single-objective continuous or mixed-variable optimization problems in which the search process may suffer from premature component disruption and stagnation around elite solutions. Typical examples include multimodal, hybrid, nonseparable, or constrained benchmark problems where different variables contribute unevenly to the objective and constraints. By contrast, direct use of DSD-LI-PO for multiobjective, dynamic, discrete combinatorial, or large-scale simulation-based engineering optimization would require additional mechanisms, such as archive management, dynamic response strategies, discrete encoding, or problem-specific constraint handling.

The benchmark functions and engineering models are treated as black-box minimization problems under the same computational budget, boundary handling, and quadratic exterior penalty treatment, while DSD and LI use fixed control settings. The engineering tests are standardized benchmark studies rather than physical or large-scale simulation-based deployments. These choices support a controlled algorithmic comparison, while industrial deployment would require domain-specific modeling and validation. Future work will therefore consider adaptive parameter control, more advanced constraint handling, and extensions to multiobjective, dynamic, discrete, and larger-scale simulation-based engineering optimization.

9. Conclusions

This study presented DSD-LI-PO, an enhanced PO variant that combines componentwise DSD with stagnation-triggered LI refinement. DSD regulates the transfer of current-best information at the component level, while LI adds a contracted candidate only when the best solution stops improving.

On the CEC2017 benchmark suite, DSD-LI-PO ranks second overall with a tie-aware average rank of 1.8966 and records 28/1/0 and 29/0/0 wins/ties/losses against PO and IPO, respectively. Its relative position is maintained in the 30-, 50-, and 100-dimensional scalability tests. The complete method also ranks first in the ablation study and second in the engineering tests, with feasible solutions obtained in all runs. Overall, the results support DSD-LI-PO as a consistent improvement of the PO framework for single-objective continuous or mixed-variable problems where componentwise progress is uneven and stagnation can restrict later search. Future work will examine adaptive control, advanced constraint handling, and extensions to multiobjective, dynamic, discrete, and larger-scale engineering optimization.

Supplementary Materials: The following supporting information can be downloaded at: <https://www.mdpi.com/article/10.3390/biomimetics11080598/s1>, Supplementary Table S1, complete functionwise mean and standard deviation results for the CEC2017 comparison; Supplementary Figures S1–S3, complete convergence curves for the main CEC2017 comparison; Supplementary Figure S4, complete convergence curves for the ablation study; Supplementary Note S1, complete mathematical formulations of the five constrained engineering design problems; Supplementary Data S1, an Excel workbook containing functionwise ranks, pairwise statistical tests, runtime summaries, detailed dimensional scalability results, parameter sensitivity results, ablation results, and engineering validation statistics.

Author Contributions: Conceptualization, M.Q. and Y.D.; methodology, software, validation, formal analysis, investigation, data curation, and visualization, M.Q.; writing—original draft preparation, M.Q.; writing—review and editing, M.Q. and Y.D.; supervision, Y.D. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The numerical summaries, statistical results, dimensional scalability data, parameter sensitivity results, ablation results, engineering validation statistics, and engineering model formulations used in this study are provided in the Supplementary Materials and Supplementary Data S1. A public GitHub repository (v1.0.0) for the source code, experiment scripts, and reproducibility files is available at <https://github.com/Eira-QmeiQ/DSD-LI-PO> (accessed on 21 July 2026).

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

References

1. Houssein, E.H.; Saeed, M.K.; Hu, G.; Al-Sayed, M.M. Metaheuristics for Solving Global and Engineering Optimization Problems: Review, Applications, Open Issues and Challenges. *Arch. Comput. Methods Eng.* **2024**, *31*, 4485–4519. [\[CrossRef\]](#)
2. Boussaïd, I.; Lepagnot, J.; Siarry, P. A survey on optimization metaheuristics. *Inf. Sci.* **2013**, *237*, 82–117. [\[CrossRef\]](#)
3. Lagaros, N.D.; Kournoutos, M.; Kallioras, N.A.; Nordas, A.N. Constraint handling techniques for metaheuristics: A state-of-the-art review and new variants. *Optim. Eng.* **2023**, *24*, 2251–2298. [\[CrossRef\]](#)
4. Michalewicz, Z.; Schoenauer, M. Evolutionary Algorithms for Constrained Parameter Optimization Problems. *Evol. Comput.* **1996**, *4*, 1–32. [\[CrossRef\]](#)
5. Yang, X.; Wang, R.; Li, K.; Ishibuchi, H. Meta-Black-Box optimization for evolutionary algorithms: Review and perspective. *Swarm Evol. Comput.* **2025**, *93*, 101838. [\[CrossRef\]](#)
6. Das, A.; Rai, R.; Sasmal, B.; Dhal, K.G.; Khurma, R.A.; Saha, R. Metaheuristic Algorithms Since 2020: Development, Taxonomy, Analysis, and Applications. *Arch. Comput. Methods Eng.* **2025**. [\[CrossRef\]](#)
7. Shaikh, M.S.; Raj, S.; Zheng, G.; Xie, S.; Wang, C.; Dong, X.; Lin, Y.; Wang, C.; Junejo, N.U.R. Applications, classifications, and challenges: A comprehensive evaluation of recently developed metaheuristics for search and analysis. *Artif. Intell. Rev.* **2025**, *58*, 390. [\[CrossRef\]](#)
8. Baizid, K.; Čuković, S.; Iqbal, J.; Yousnadj, A.; Chellali, R.; Meddahi, A.; Devedžić, G.; Ghionea, I. IROSim: Industrial Robotics Simulation Design Planning and Optimization platform based on CAD and knowledgeware technologies. *Robot. Comput.-Integr. Manuf.* **2016**, *42*, 121–134. [\[CrossRef\]](#)
9. Lian, J.; Hui, G.; Ma, L.; Zhu, T.; Wu, X.; Heidari, A.A.; Chen, Y.; Chen, H. Parrot optimizer: Algorithm and applications to medical problems. *Comput. Biol. Med.* **2024**, *172*, 108064. [\[CrossRef\]](#) [\[PubMed\]](#)
10. Awad, N.H.; Ali, M.Z.; Liang, J.J.; Qu, B.Y.; Suganthan, P.N. *Problem Definitions and Evaluation Criteria for the CEC 2017 Special Session and Competition on Single Objective Bound Constrained Real-Parameter Numerical Optimization*; Technical Report; Nanyang Technological University: Singapore, 2016.
11. Saad, M.R.; Emam, M.M.; Houssein, E.H. An efficient multi-objective parrot optimizer for global and engineering optimization problems. *Sci. Rep.* **2025**, *15*, 5126. [\[CrossRef\]](#) [\[PubMed\]](#)

12. Sayyouri, A.; Bencherqui, A.; Mansouri, H.; Karmouni, H.; Moustabchir, H.; Sayyouri, M.; Bourkane, A.; Cherkaoui, A.; Askar, S.S.; Abouhawwash, M. Improved optimization based on parrot's chaotic optimizer for solving complex problems in engineering and medical image segmentation. *Sci. Rep.* **2025**, *15*, 26317. [[CrossRef](#)] [[PubMed](#)]
13. Adegbeye, O.R.; Feda, A.K.; Tejani, G.G.; Smerat, A.; Kumar, P.; Agyekum, E.B. Salp Navigation and Competitive based Parrot Optimizer (SNCPPO) for efficient extreme learning machine training and global numerical optimization. *Sci. Rep.* **2025**, *15*, 13704. [[CrossRef](#)] [[PubMed](#)]
14. Li, F.; Dai, C.; Hussien, A.G.; Zheng, R. IPO: An Improved Parrot Optimizer for Global Optimization and Multilayer Perceptron Classification Problems. *Biomimetics* **2025**, *10*, 358. [[CrossRef](#)] [[PubMed](#)]
15. Lan, L.; Qi, Z. An enhanced parrot optimizer with multiple strategies for wireless sensor network node deployment. *Sci. Rep.* **2026**, *16*, 1285. [[CrossRef](#)] [[PubMed](#)]
16. Storn, R.; Price, K. Differential Evolution—A Simple and Efficient Heuristic for global Optimization over Continuous Spaces. *J. Glob. Optim.* **1997**, *11*, 341–359. [[CrossRef](#)]
17. Omidvar, M.N.; Yang, M.; Mei, Y.; Li, X.; Yao, X. DG2: A Faster and More Accurate Differential Grouping for Large-Scale Black-Box Optimization. *IEEE Trans. Evol. Comput.* **2017**, *21*, 929–942. [[CrossRef](#)]
18. Zhang, S.X.; Liu, Y.H.; Hu, X.R.; Luo, J.T.; Zheng, L.M.; Zheng, S.Y. Differential evolution with dimensionally adaptive inheritance. *Eng. Appl. Artif. Intell.* **2026**, *163*, 112587. [[CrossRef](#)]
19. Kennedy, J.; Eberhart, R. Particle swarm optimization. In Proceedings of the ICNN'95—International Conference on Neural Networks, Perth, WA, Australia, 27 November–1 December 1995; Volume 4, pp. 1942–1948. [[CrossRef](#)]
20. Shami, T.M.; Al-Tashi, Q.; Khodadadi, N.; Abdulkadir, S.J.; Ahmed, A.; Mirjalili, S. Dimension selection: An innovative metaheuristic strategy for particle swarm optimization. *Clust. Comput.* **2025**, *28*, 379. [[CrossRef](#)]
21. Zhou, W.; Meng, Z. An adaptive differential evolution with dynamic perturbation and dimensional bidirectional crossover mechanism for diversity enhancement. *Eng. Appl. Artif. Intell.* **2025**, *141*, 109750. [[CrossRef](#)]
22. Ye, X.; Li, J.; Wang, P.; Suganthan, P.N. A comprehensive survey of adaptive strategies in differential evolutionary algorithms. *Swarm Evol. Comput.* **2025**, *98*, 102081. [[CrossRef](#)]
23. Tizhoosh, H.R. Opposition-Based Learning: A New Scheme for Machine Intelligence. In Proceedings of the International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce, Vienna, Austria, 28–30 November 2005; Volume 1, pp. 695–701. [[CrossRef](#)]
24. Rahnamayan, S.; Tizhoosh, H.; Salama, M. Opposition-Based Differential Evolution. *IEEE Trans. Evol. Comput.* **2008**, *12*, 64–79. [[CrossRef](#)]
25. Choi, T.J.; Pachauri, N. Adaptive search space for stochastic opposition-based learning in differential evolution. *Knowl.-Based Syst.* **2024**, *300*, 112172. [[CrossRef](#)]
26. Yu, F.; Guan, J.; Wu, H.; Chen, Y.; Xia, X. Lens imaging opposition-based learning for differential evolution with cauchy perturbation. *Appl. Soft Comput.* **2024**, *152*, 111211. [[CrossRef](#)]
27. Liang, J.; Qin, A.; Suganthan, P.; Baskar, S. Comprehensive learning particle swarm optimizer for global optimization of multimodal functions. *IEEE Trans. Evol. Comput.* **2006**, *10*, 281–295. [[CrossRef](#)]
28. Chen, Y.; Ouyang, C.; Liu, Y.; Zhang, H.; Gan, Z. Dynamic cooperative coevolution based on variable importance for non-separable large-scale global optimization. *Appl. Soft Comput.* **2025**, *180*, 113363. [[CrossRef](#)]
29. LaTorre, A.; Molina, D.; Osaba, E.; Del Ser, J.; Herrera, F. A prescription of methodological guidelines for comparing bio-inspired optimization algorithms. *Swarm Evol. Comput.* **2021**, *67*, 100973. [[CrossRef](#)]
30. Tanabe, R.; Fukunaga, A.S. Improving the search performance of SHADE using linear population size reduction. In Proceedings of the 2014 IEEE Congress on Evolutionary Computation (CEC), Beijing, China, 6–11 July 2014; pp. 1658–1665. [[CrossRef](#)]
31. Chopra, N.; Ansari, M.M. Golden jackal optimization: A novel nature-inspired optimizer for engineering applications. *Expert Syst. Appl.* **2022**, *198*, 116924. [[CrossRef](#)]
32. Xue, J.; Shen, B. Dung beetle optimizer: A new meta-heuristic algorithm for global optimization. *J. Supercomput.* **2023**, *79*, 7305–7336. [[CrossRef](#)]
33. Mirjalili, S.; Lewis, A. The Whale Optimization Algorithm. *Adv. Eng. Softw.* **2016**, *95*, 51–67. [[CrossRef](#)]
34. Heidari, A.A.; Mirjalili, S.; Faris, H.; Aljarah, I.; Mafarja, M.; Chen, H. Harris hawks optimization: Algorithm and applications. *Future Gener. Comput. Syst.* **2019**, *97*, 849–872. [[CrossRef](#)]
35. Wilcoxon, F. Individual Comparisons by Ranking Methods. *Biom. Bull.* **1945**, *1*, 80–83. [[CrossRef](#)]
36. Derrac, J.; García, S.; Molina, D.; Herrera, F. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm Evol. Comput.* **2011**, *1*, 3–18. [[CrossRef](#)]
37. Cliff, N. Dominance Statistics: Ordinal Analyses to Answer Ordinal Questions. *Psychol. Bull.* **1993**, *114*, 494–509. [[CrossRef](#)]
38. Yang, X.S.; Huyck, C.; Karamanoglu, M.; Khan, N. True global optimality of the pressure vessel design problem: A benchmark for bio-inspired optimisation algorithms. *Int. J.-Bio-Inspired Comput.* **2013**, *5*, 329–335. [[CrossRef](#)]

39. Coello Coello, C.A. Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: A survey of the state of the art. *Comput. Methods Appl. Mech. Eng.* **2002**, *191*, 1245–1287. [[CrossRef](#)]
40. Deb, K. An efficient constraint handling method for genetic algorithms. *Comput. Methods Appl. Mech. Eng.* **2000**, *186*, 311–338. [[CrossRef](#)]
41. Coello Coello, C.A. Use of a self-adaptive penalty approach for engineering optimization problems. *Comput. Ind.* **2000**, *41*, 113–127. [[CrossRef](#)]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.