Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

CHESS: A Structured Jump and Directed Line-Search Metaheuristic for Global Optimization

Mohammad Amin Behzadi
Sharif university of technology
Iman Hosseini
iman.hosseini@shirazu.ac.ir
Shiraz university

Research Article

Keywords: Chess Algorithm, Optimization, convergence, robustness, multidimensional functions

Posted Date: November 18th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11597235/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

CHESS: A Structured Jump and Directed Line-Search Metaheuristic for Global Optimization

Mohammad Amin Behzadi[a], Iman Hosseini[b*]
a: Department of mechanical engineering, Sharif university of technology, Tehran, Iran
b- Hydro-Aeronautical research center, Shiraz university, Shiraz, Iran (iman.hosseini@shirazu.ac.ir)

Abstract—Metaheuristic optimization algorithms are powerful numerical tools for solving complex, high-dimensional problems. Population-based by design, they seek near-optimal solutions efficiently. Owing to their flexibility, metaheuristic algorithms are applicable across a wide range of optimization tasks. However, one of the key challenges they face is handling multi-dimensional functions. Numerous algorithms have been proposed to date, competing on speed, accuracy, and robustness. In this paper, several existing optimization algorithms are first reviewed. Then, a novel optimization algorithm inspired by the chess game is introduced, to predict the global optimum more accurately and efficiently. Various tuning parameters are proposed to adapt the algorithm to specific problems, enhancing both convergence speed and accuracy. A comparative evaluation against established methods shows that the proposed Chess Optimization (CHESS) algorithm is competitive in robustness, accuracy, and speed on the benchmark problems considered. These results are demonstrated on four representative benchmark functions with five independent runs per algorithm; extending the evaluation to larger-scale benchmark suites is discussed as future work.

Index Terms—Chess Algorithm, Optimization, convergence, robustness, multidimensional functions

1 Introduction

Global optimization has become a cornerstone of modern computational science, with applications spanning industries such as manufacturing, economics, and control systems. These problems encompass a diverse range of characteristics, including continuous or discontinuous, convex or nonconvex, and constrained or unconstrained. Addressing such complexity has driven the development of numerous solution strategies, which are broadly categorized into classical mathematical methods and heuristic approaches.

Classical mathematical methods, including techniques like differentiation, the Newton-Raphson method, and the Runge-Kutta method, rely on rigorous analytical formulations. While these methods are powerful under ideal conditions, they suffer from significant limitations in practical applications. For example, they require functions to be continuous, involve complex computations for derivatives, and often struggle with multidimensional or nonconvex problems. Additionally, their deterministic nature makes them prone to premature convergence and local optima, especially when faced with irregular solution landscapes.

In contrast, heuristic methods have emerged as a transformative approach to overcoming these challenges. These algorithms, inspired by natural phenomena and human decision-making processes, leverage intelligent search strategies to explore solution spaces effectively. Unlike classical methods, heuristics do not rely on derivatives, making them suitable for nonconvex and nonlinear problems. Despite their success, heuristic methods also face challenges, including high computational costs for large-scale problems and susceptibility to local optima, particularly in high-dimensional spaces with complex landscapes.

This study introduces a novel optimization algorithm inspired by chess movements, designed to address these limitations. The algorithm innovatively combines strategic search patterns with adaptable parameters, providing a versatile and efficient solution framework.

Its contributions are as follows:

- A novel chess-inspired optimization algorithm: This algorithm overcomes some limitations of classical and heuristic methods by integrating intelligent mechanisms to pursue global solutions specifically for multi-dimensional and highly fluctuating objective functions.
- Adjustable parameters for flexibility: The algorithm includes multiple tunable parameters, allowing users to fine-tune its performance for a wide range of optimization problems.
- Broad applicability: Its flexibility enables applications to multidimensional and nonconvex problems, especially those with high fluctuations.
- Minimized computational costs: The algorithm is designed to optimize resource usage, achieving faster convergence without sacrificing accuracy.
- Benchmark performance: Comparisons against established methods on the benchmark functions considered demonstrate competitive accuracy and speed (Section 6); broader validation on high-dimensional, large-scale suites is identified as future work (Section 7).
- Effectiveness in complex environments: The algorithm excels in addressing complex optimization challenges, particularly in environments characterized by significant fluctuations and high dimensionality.

To state the contribution precisely: it is not the chess

metaphor itself, but a formally defined combination of three operators and their operational coupling. (i) A Structured Jump operator generates anisotropic, possibly adaptive proposals concentrated on informative subspaces (Eq. 1–2). (ii) A Rank-Biased Probabilistic Target Selection (RPTS) policy probabilistically selects promising candidates from a pooled set using tunable rank weights (Eq. 7–8). (iii) A Directed Line-Search with Subsampling and Orthogonal Jitter (DLSS) performs low-cost exploitation along population-guided directions while avoiding narrow-ridge entrapment (Eq. 6). While each individual element has antecedents in the literature – random/subspace search, rank selection in evolutionary algorithms, and direct-search line probes – the claimed novelty is the formal integration of structured sampling, rank-biased target choice, and subsampled directional exploitation into a single coupled optimizer, rather than their use as independent components.

This claim needs to be weighed against two recent works that also use chess or board-game strategy as their organizing metaphor. Offensive Defensive Optimization (ODO) models offensive behavior (global exploration) and defensive behavior (local exploitation) as two competing update rules, validated on CEC2017/CEC2022 and two engineering design problems [1]. Chess-Inspired Multi-Algorithm Optimization (CheMO) instead assigns each type of chess piece to a distinct existing classical algorithm – pawns to simulated annealing, bishops to BFGS, knights to Nelder–Mead, rooks to a direct-search method, queens to a chained BFGS–Nelder-Mead–direct-search sequence – using the board purely as a dispatch mechanism between established solvers . CHESS differs from both in kind rather than degree: rather than modeling offense/defense as competing behaviors, or routing between existing algorithms by piece type, it defines two new operators – the Structured Jump and the Directed Line-Search with Subsampling and Orthogonal Jitter – coupled through a shared rank-based selection policy (RPTS) that neither prior work formalizes. The chess motif here names the geometry of the two search operators (an L-shaped structured jump and a combined linear/diagonal directed search), not a piece-to-algorithm mapping or a behavioral dichotomy. Metaphor-driven algorithm design of this kind has also drawn sustained methodological criticism in the field [2]; the response offered here is that the contribution is judged on the operator formalism above, independent of the chess framing.

Mathematical definitions of each operator are given in Sec. 4.2, 4.4, and 4.5; comparisons to established baselines are reported in Sec. 6. The results of this study reveal that the proposed algorithm offers a robust, adaptable, and computationally efficient alternative to existing methodologies for the multidimensional, nonconvex benchmark problems considered.

2 Notation

The main parameters of the Chess Optimization Algorithm are summarized in Table 1.

TABLE 1: Notation used in the Chess Optimization Algorithm

Symbol	Description
$\Omega \subset \mathbb{R}^n$	Search domain
n	Dimension of decision vector
$x \in \mathbb{R}^n$	Candidate solution
$f(x)$	Objective function value
m	Number of sampling agents (knights)
p	Number of directed agents (queens)
$X = \{x_j\}_{j=1}^{m}$	Sampling-agent positions
$Y = \{y_q\}_{q=1}^{p}$	Directed-agent positions
q_J	Jump samples per knight per iteration
θ_J	Parameters of Sampling Jump distribution
N	Number of top sampled candidates retained
$R > 0$	Placement range around top-N
$\alpha > 0$	Line-search step length
$k \in \mathbb{N}$	Subsampling factor along line-search
$\sigma_j \geq 0$	Orthogonal jitter magnitude (DLSS)
$t_{\max}$	Maximum iterations
$F_{\max}$	Maximum function evaluations
T_{noimp}	No-improvement stopping iterations
tol_{abs}	Absolute improvement tolerance
$\rho \in [0, 1]$	Fraction of knights to reseed
use_subspace	Enable random subspace restriction
s	Subspace size (active coordinates)
grid_mode	Enable domain discretization
$\Delta > 0$	Grid spacing

3 Related Works

Metaheuristic optimization has developed through several complementary families of population-based and direct-search methods. Genetic algorithms (GA) remain a foundational evolutionary approach, using selection, crossover, and mutation to explore complex objective landscapes. Modifications of GA have addressed global-search efficiency and convergence behavior, including alternative mutation and stopping strategies [3]. GA has also been applied to practical engineering optimization and controller design, illustrating its usefulness beyond synthetic benchmarks. In particular, [4] has demonstrated GA-based optimization for decentralized PI/PID controller tuning.

Particle swarm optimization (PSO) is another major population-based method, in which candidate solutions are updated using individual and collective search information [5], [6]. Several variants have been proposed to improve convergence and robustness, including modified PSO and Kalman-filter-assisted PSO [7], [8]. QUAPSO further introduced adaptive quantum-inspired search mechanisms for balancing exploration and exploitation [9]. The convergence properties of PSO have also been studied from a control-theoretic perspective: Wakasa et al. analyzed its decay rate and l^2 gain in the Asian Journal of Control [10]. PSO has additionally been used directly for controller tuning, including LQG tuning for a DC–DC converter [11].

Hybrid evolutionary methods combine complementary search mechanisms. PSO–GA hybrids have been investigated for combinatorial and continuous optimization, while genetic operators can be incorporated into PSO to improve population diversity and refinement [12], [13]. Hybrid PSO–GWO approaches have likewise been proposed [14]. These studies motivate the use of multiple coordinated search mechanisms rather than relying on a single update rule.

Grey Wolf Optimization (GWO) is a widely used population-based method inspired by hierarchical hunting behavior. Improved GWO variants have been applied to

structural design and engineering problems [15], while self-adaptive variants have been investigated for electromagnetic and 5G-related design problems [16]. Ant Colony Optimization (ACO) provides another established population-based paradigm, using collective pheromone information to guide the search [17]. Hybrid combinations of evolutionary methods with local/direct search, such as GA–PSO–Nelder–Mead schemes, have also been reported [18]. Comparative studies across ACO, PSO, ABC, FA, and GA further illustrate the trade-offs among exploration, exploitation, and computational cost [19].

Metaheuristic optimization has increasingly been coupled with surrogate, machine-learning, and derivative-free strategies. Gaussian-process/PSO approaches have been investigated for mixed-variable hyperparameter optimization [20]; cooperative population methods assisted by dimensionality reduction have been proposed for expensive optimization [21]; and derivative-free branch-and-model methods have been developed to reduce the number of expensive function evaluations [22]. These directions are particularly relevant to CHESS because its design explicitly separates candidate generation from a low-cost directional exploitation stage.

Recent work has also produced a large number of newly named nature- and behavior-inspired optimizers. Representative examples include the Snake Optimizer [23], Pelican Optimization Algorithm [24], Tangent Search Algorithm [25], Dwarf Mongoose Optimization [26], White Shark Optimizer [27], Starling Murmuration Optimizer [28], Termite Queen Algorithm [29], RIME [30], Nutcracker Optimization [31], and Gold Rush Optimization [32]. A systematic review has documented the rapid growth of this class of algorithms and the associated reproducibility and comparison challenges [33]. Rather than reproducing a long catalogue of such methods, this work focuses on the mechanisms that are most directly relevant to CHESS: structured sampling, rank-biased selection, directional exploitation, and the balance between exploration and exploitation.

The control literature also provides relevant examples of optimization integrated with estimation and constrained control. Work by Hosseini et al. on distributed model predictive control and by Mirzaei et al. on attitude determination illustrates the broader engineering context in which optimization and intelligent search can support complex control systems [34], [35]. More recent Asian Journal of Control studies similarly combine optimization with nonlinear control and system identification, including improved differential evolution for fractional-order system identification [36] and PSO-assisted design of nonlinear control systems [37]. Optimization-based controller tuning has also been formulated through computationally efficient objective minimization in recent Asian Journal of Control work [38].

A particularly important distinction for the present study concerns chess- and metaphor-based optimization. Offensive Defensive Optimization (ODO) uses offensive and defensive behaviors as two competing search mechanisms and has been evaluated on standard benchmark suites and engineering problems [1]. CHESS is different in its algorithmic construction: its chess metaphor corresponds to explicitly defined structured-jump and directed line-search operators, coupled through ranked probabilistic target selection, rather than merely assigning chess terminology to an existing optimizer. At the same time, the broader literature has raised methodological concerns about metaphor-driven algorithm design [2], [39], [40]. Accordingly, the intended contribution of CHESS is the formal coupling and empirical evaluation of its operators, not the chess metaphor itself.

4 Algorithm Approach

The algorithm is formalized using standard optimization terminology. Initialize m sampling agents $\{x_j\}_{j=1}^{m}$uniformly over the search domain. Each agent produces $q_J$candidate points by applying the structured jump operator $J\left(x_j;\theta_J\right)$. The objective $f$is evaluated at all generated candidates, and the best Npoints (across all agents) are retained for the next stage. For each directed-search agent q, a target point t_q is selected from the top-N candidate set using Ranked Probabilistic Target Selection (RPTS), which applies linear rank-based weights to bias the selection toward high-quality points while maintaining exploration. Let $y_q$be the current position of directed agent q.

The agent moves toward its selected target t_q along the normalized direction $v = \frac{t_q - y_q}{\|t_q - y_q\|}$. Points along this line are sampled at a fixed step size α, but only every k-th point is evaluated to reduce the number of function evaluations. After all evaluated line points are collected, the position y_q is updated to the best point encountered along its trajectory. The algorithm stops when either (i) the total number of function evaluations exceeds $F_{\max}$, or (ii) no further improvement in the best objective value is observed for T_{noimp} consecutive iterations.

To help the reader develop an intuitive understanding of the algorithm's metaphorical structure, the overall concept is explained. Initially, the algorithm is formulated for two-dimensional functions and subsequently extended to n-dimensional optimization problems. The process begins with the initialization of agents referred to as knights, which move in an "L"-shaped pattern within the search space, analogous to their movement in chess. At each step, the knights evaluate the objective function at their landing positions, gathering candidate solutions. Subsequently, a queen agent is activated, which explores the search space using combined linear and diagonal movements—mimicking the full range of queen movements in chess. The queen evaluates the objective function at each visited point and identifies the best-performing solutions. If multiple locations yield identical optimal values, convergence is declared. Otherwise, the solution set is updated based on the queen's exploration, and the algorithm repeats the loop until convergence criteria are met. An overview of the algorithm's structure is illustrated in Figure 1.

The algorithm expands step by step as follows:

4.1 Step 1: Discretizing the Domain

We optionally discretize the domain Ω into a finite grid $\mathcal{G}$ with spacing Δ. In discretized mode every candidate point is projected to the nearest grid node; in continuous mode no projection is applied. The grid spacing Δ (or equivalently number of discrete places per dimension) controls search accuracy vs computational cost. Grid: $\mathcal{G} = \{a_i + k\Delta : k \in \mathbb{Z}, a_i \text{ min bound in dimension } i\}$. In Figure . 2., a comparison between two levels of accuracy is illustrated.

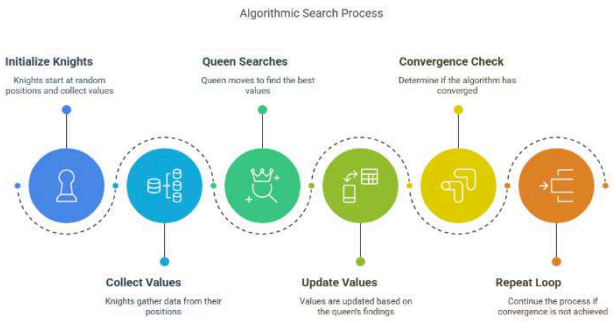


Fig. 1: Overall metaphorical view of chess algorithm.

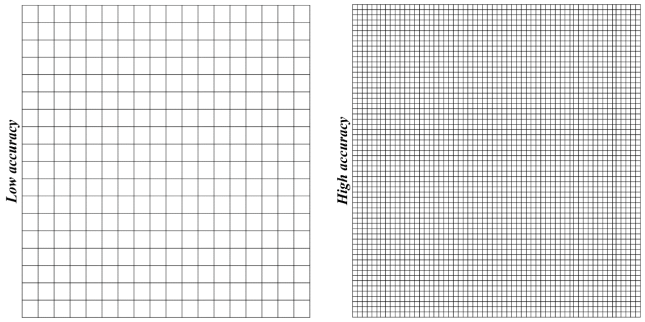


Fig. 2: Two accuracies are compared. (a) Domain is divided in lower accuracy. (b) Domain is divided in higher accuracy.

4.2 Step 2: Sampling Jump Operator J

Define Sampling Jump Operator $J\left(x;\theta_J\right)$: from current point x draw q_J perturbations $\delta_i \sim \mathcal{D}\left(\theta_J\right)$ and form candidates $x+\delta_i$. In discretized mode project candidates onto $\mathcal{G}$. Evaluate f at these candidates and keep the top Nglobally across all sampling agents for directed search.

$$J\left(x;\theta_J\right)=\left\{x+\delta_i:\delta_i\sim\mathcal{D}(\theta_J)\,,\ i=1,\ldots,q_J\right\}\cap\Omega \quad (1)$$

For example, if L-shape as a structured jump in 2D is considered, define $\mathcal{D}$ that draws offsets $(\pm a,\pm b)$and $(\pm b,\pm a)$.The algorithm then initializes several "knights" (key input 2), which are randomly placed within the discretized domain. The vertical and horizontal components of this L-shaped movement are defined by key inputs 3 and 4.

$$J\left(x;\theta_J\right)=\{x+o:o\in\mathcal{O}_L\}\cap\Omega,\mathcal{O}_L=\{(\pm a,\pm b),(\pm b,\pm a)\} \quad (2)$$

The algorithm calculates the function value at each position where the knights land. Finally, it selects the top N (key input 5) best positions from this initial set of search operator points. The placement of the directed-search agents around these top-N candidates is defined in Step 3 (Eq. 4–5).

Generalization to n dimensions. The 2D illustration above draws jump offsets from the coordinate-pair set $\mathcal{O}_L=\{(\pm a,\pm b),(\pm b,\pm a)\}$, i.e., exactly two coordinates are perturbed per candidate, mirroring a knight's move on a chessboard. For $n>2$, the same structured-jump principle is applied by selecting a random pair of coordinates $\{i,j\}\subseteq\{1,\ldots,n\}$ at each draw and applying the L-shaped offset to that pair while holding the remaining $n-2$ coordinates fixed:

$$J(x;\theta_J)=\Bigg\{x+o\,e_{\{i,j\}}:o\in\mathcal{O}_L,\ \{i,j\}\sim \text{Uniform}\Big(\binom{\{1,\ldots,n\}}{2}\Big)\Bigg\}\cap\Omega, \quad (3)$$

where $e_{\{i,j\}}$ places the two offset components on coordinates i,j and zeros elsewhere. This is the same mechanism used by the random-subspace restriction of Section 4.6 with subspace size $s=2$, and it is the definition actually exercised whenever $n>2$ in the experiments of Section 6.

4.3 Step 3: Choosing top-N & seeding searching agents

After identifying the top N candidates S_N from the sampling phase, we initialize the p directed-search agents (key input 6) inside a bounded region that covers all points in S_N. Let R be a user-specified placement range (key input 7). Given $S_N=\{s_1,\ldots,s_N\}$, define the component-wise vectors, $\underline{s}=\min_{s\in S_N} s, \bar{s}=\max_{s\in S_N} s$. Let the range (key input 7) be a positive scalar $R>0$.

The hyper-rectangle is defined as:

$$\mathcal{B}(S_N,R)=\left\{x\in\mathbb{R}^d:\underline{s}_i-R\le x_i\le\bar{s}_i+R,\ i=1,\ldots,d\right\} \quad (4)$$

Placing the p queens means:

$$y_q^0\sim\text{Uniform}(\mathcal{B}(S_N,R))\,,\qquad q=1,\ldots,p \quad (5)$$

The hyper-rectangle is defined around the top candidates. Placing the queens means each directed-search agent is placed uniformly at random inside this region. This ensures all p agents begin their directed search in the region surrounding the best candidates discovered so far.

4.4 Step 4: Directed Line-Search Operator

Define directed line search $\mathcal{L}(y,t;\alpha,k,\sigma)$produce points on the line from y to t with step α, evaluate only every k-th point (key input 11). Optionally add orthogonal jitter σ to avoid strictly 1D paths.

$$\mathcal{L}(y,t;\alpha,k,\sigma)=\{y+n\alpha v+\mathbf{1}_{\{\sigma>0\}}\xi_n \ :\ n=0,k,2k,\ldots,n_{\max}\} \quad (6)$$

where:
$\xi_n\sim\mathcal{N}(0,\sigma^2P_v^{\perp})$,
$v=\frac{t-y}{\|t-y\|}, P_v^{\perp}=I-vv^{\top},\ \ n_{\max}=\left\lfloor\frac{\|t-y\|}{\alpha}\right\rfloor$.

The upper bound $n_{\max}$ ensures that the directed line-search generates points only up to the target $t$and never overshoots it, since $n\alpha$represents the cumulative step distance along the direction v The set enumerates only every k-th point, not all steps. The jitter term is a small perturbation added to prevent the search from stagnating on a strictly linear path. When $\sigma=0$, jitter disappears automatically and the operator becomes purely deterministic. $P_v^{\perp}$ensures jitter is orthogonal to the search direction (avoids 1-D stagnation). It perturbs only in directions perpendicular to the search direction, so it does not change the step distance toward the target, only widens the search tube in surrounding space.

4.5 Ranked Probabilistic Target Selection (RPTS)

RPTS converts a pooled candidate set into a discrete probability distribution over targets using rank weights.

Given a pooled candidate set $\mathcal{C} = \{c_1, \ldots, c_N\}$ and their objective values $\{f(c_i)\}$, sort the candidates so that

$$f(c_{(1)}) \leq f(c_{(2)}) \leq \cdots \leq f(c_{(N)}) \tag{7}$$

where $c_{(r)}$ denotes the r-th best candidate (rank r). The selection probability for rank r is defined as

$$\begin{aligned} w_r &= N + 1 - r + \varepsilon, & r &= 1, \ldots, N, \\ Z &= \sum_{s=1}^{N} w_s = \frac{N(N+1)}{2} + N\varepsilon, & & \\ P_r &= \frac{w_r}{Z}, & \sum_{r=1}^{N} P_r &= 1. \end{aligned} \tag{8}$$

To avoid degeneracy in cumulative probability construction and to break ties reliably, a small positive constant ε is added.

4.6 Convergence Check

Let $b^{(t)}$ be the best objective value found at iteration t, and let $b^{(t-1)}$ be the best value from the previous iteration. The algorithm checks improvement by computing

$$\Delta_t = \left| b^{(t)} - b^{(t-1)} \right|. \tag{9}$$

The algorithm stops (converges) when $\Delta_t < \text{tol}_{\text{abs}}$, where tol_{abs} is a small tolerance value (e.g., 10^{-5}). If the condition is not satisfied, the algorithm continues to the next iteration and updates $b^{(t-1)} \leftarrow b^{(t)}$. To prevent infinite loops, the algorithm also stops when the maximum iteration count or maximum function-evaluation budget is reached. The algorithm terminates when any of the following holds:

$$\Delta_t < \text{tol}_{\text{abs}} \quad \text{or} \quad t \geq t_{\max} \quad \text{or} \quad FE \geq FE_{\max}. \tag{10}$$

FE stands for Function Evaluations, the number of times the objective function $f(x)$ has been evaluated during the algorithm.

For multidimensional problems, Step 3 uses a random selection process to determine which N dimensions (key input 10) receive focus at each iteration, ensuring comprehensive coverage of all dimensions over the course of the run.. Optionally a random subspace operator is defined as at iteration r sample a subset of dimensions $D_r \subseteq \{1, \ldots, d\}$ of size s. Operators J and L are applied only to coordinates in D_r (others kept fixed). This reduces per-iteration cost and can improve scalability. The algorithm is explained in Table 2.

TABLE 2: Summary of the CHESS optimization algorithm.

Stage	Description
Initialization	Initialize sampling agents X uniformly in Ω, directed agents Y, iteration counter $t = 0$, function evaluations $FE = 0$, best objective value $b_0 = +\infty$, and the no-improvement counter.
Main Loop	Repeat while $t < t_{\max}$, $FE < FE_{\max}$, and the no-improvement limit has not been exceeded.
Exploration	Generate candidate solutions around each sampling agent using random perturbations. Evaluate feasible samples and collect them.
Elite Selection	Select the best N samples from S_{all} to form the elite set S_N.
Search Region Construction	Construct the bounding box $\mathcal{B}(S_N, R)$ around S_N, expanded by radius R.
Exploitation (DLSS)	For each directed agent, perform a discretized line search toward a selected elite target with optional jitter and subsampling, then update the agent position.
Pooling and Update	Pool elite and directed-search samples, identify the iteration best solution b_t, and update the global best solution.
Convergence Check	Compute $\Delta_t = \lvert b_t - b_{t-1} \rvert$.
Optional Mechanisms	Reseed sampling agents to maintain diversity and optionally restrict updates to a randomly selected subspace of dimensions.
Termination	Terminate when convergence is achieved or when $t \geq t_{\max}$ or $FE \geq FE_{\max}$.
Output	Return the best solution found, its objective value, and the total number of function evaluations.

5 Convergence Behavior of Search Operators: Illustrative Examples

The search behavior of the proposed CHESS algorithm is illustrated using two representative benchmark functions. These examples provide an intuitive visualization of how the search operators balance exploration and exploitation while progressing toward the global optimum.

The first example considers the simple paraboloid function

$$f(x, y) = x^2 + y^2, \tag{11}$$

whose unique global minimum is located at $(0, 0)$. A single queen is employed to explore the search space. Figure 3 shows the surface of the objective function, while Figure 4 illustrates the trajectory of the queen during the optimization process. As the search progresses, the queen gradually approaches the global optimum through successive exploratory and exploitative movements.

The second example considers the Eggholder function, which is well known for its highly multimodal landscape containing numerous local minima. This function presents a considerably more challenging optimization problem than the paraboloid function. The global minimum is located approximately at $(512, 404.2319)$.

Figure 5 presents the surface of the Eggholder function. To illustrate the search dynamics, two queens are initialized at different locations. Their trajectories over four representative iterations are shown in Fig. 6. Initially, both queens explore different regions of the search space. As the optimization proceeds, the search trajectories become increasingly concentrated around promising regions. In the final iteration, one queen reaches the vicinity of the global optimum, while the second queen continues exploring neighboring regions, demonstrating the balance between exploitation and exploration achieved by the proposed algorithm.

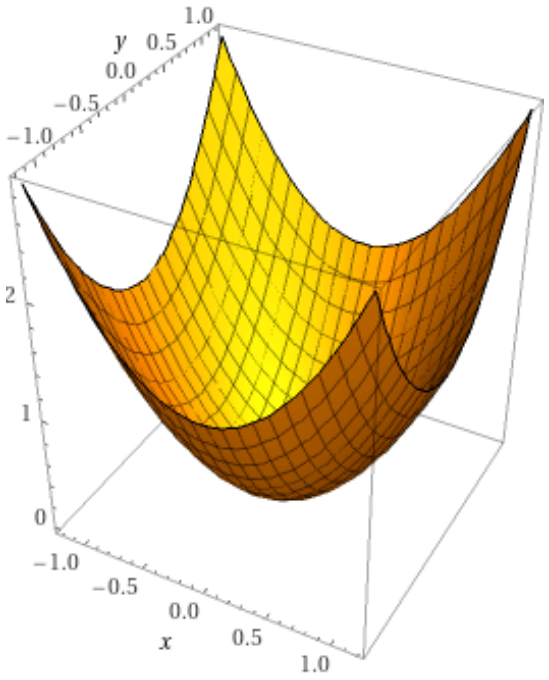


Fig. 3: Three-dimensional surface of the paraboloid function, $f(x, y) = x^2 + y^2$.

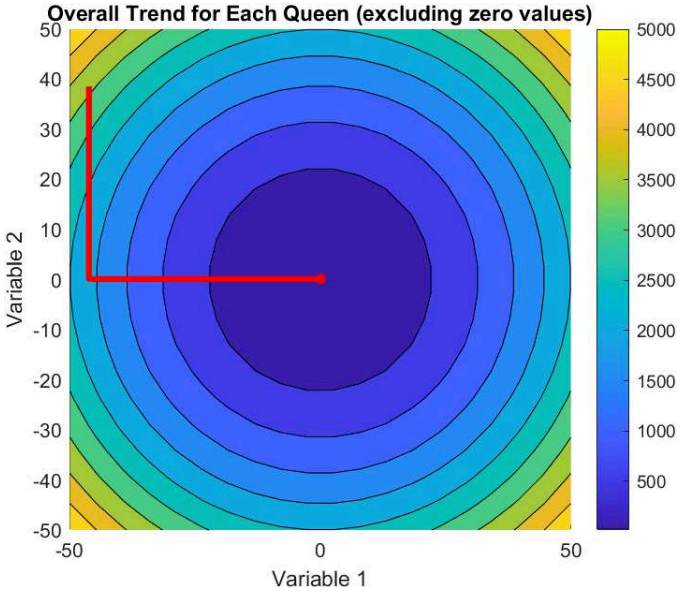


Fig. 4: Trajectory of a single queen on the paraboloid function. The queen progressively moves toward the global optimum at $(0, 0)$.

It is important to emphasize that the search operators are not restricted to selecting only the best solution obtained in the previous iteration. Such a greedy strategy could easily lead to premature convergence in highly multimodal search spaces. Instead, the proposed Rank-Weighted Probabilistic Target Selection (RPTS) mechanism assigns selection probabilities according to the ranking of candidate solutions. Better solutions receive higher probabilities, while lower-ranked solutions retain a non-zero probability of being selected. Consequently, the search is naturally biased toward promising regions while

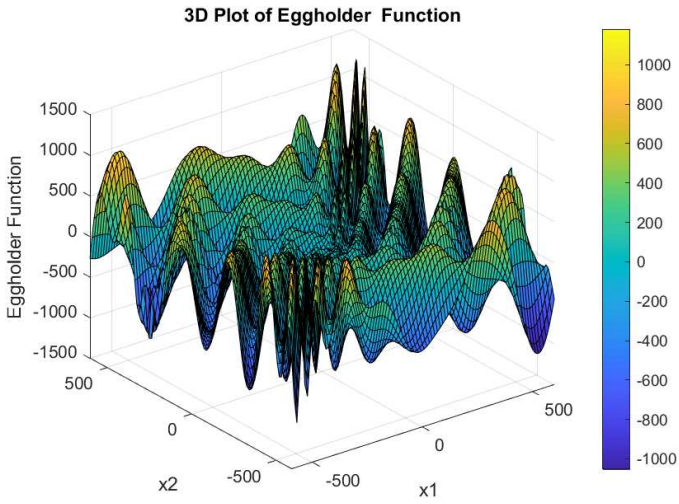


Fig. 5: Three-dimensional surface of the Eggholder function.

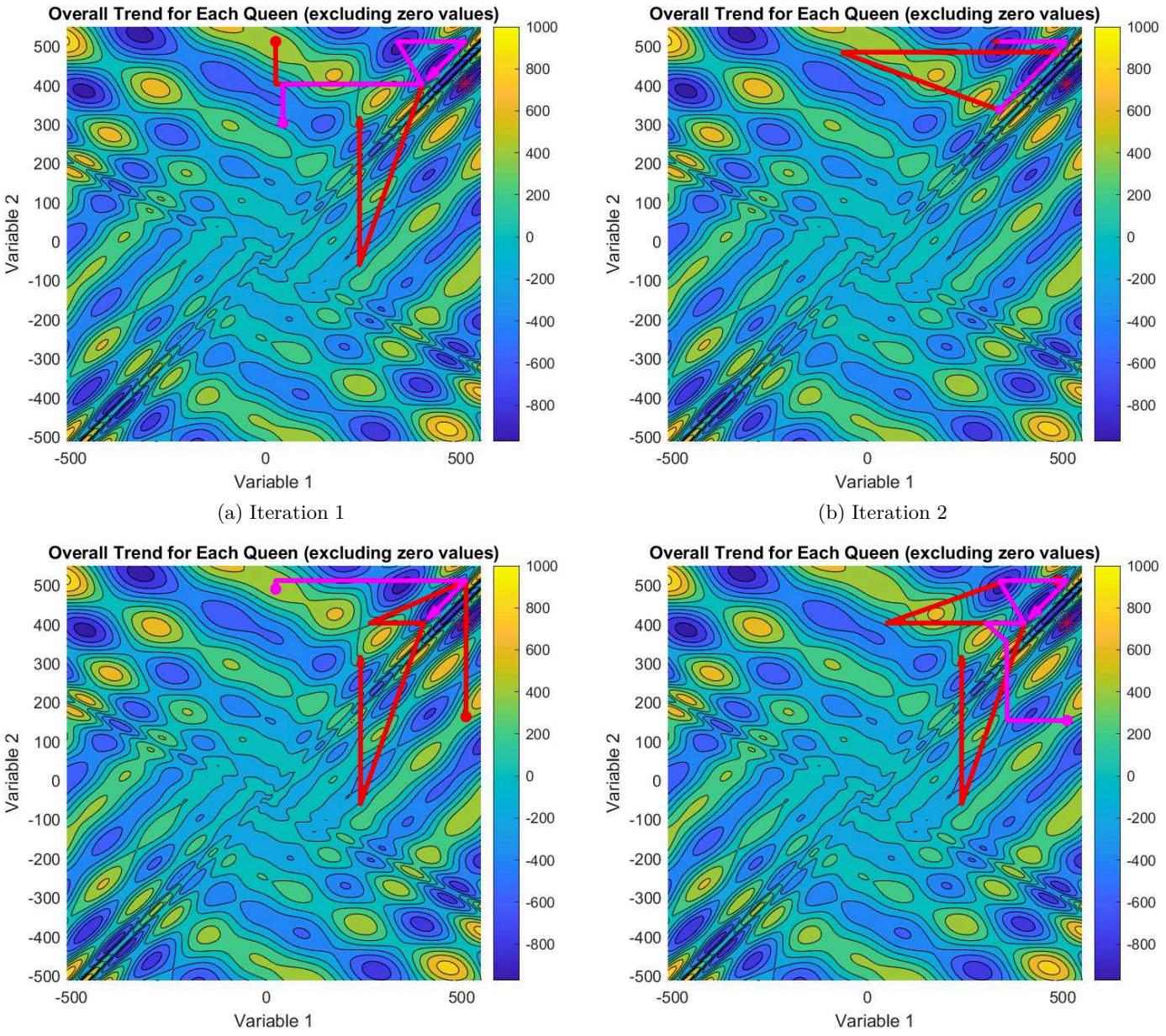
preserving sufficient diversity to escape local optima. This probabilistic selection strategy contributes significantly to the robustness and global search capability of the proposed CHESS algorithm.

6 Performance Evaluation

This section evaluates the performance of Chess Optimization against state-of-the-art optimization algorithms, including Genetic Algorithm (GA), Grey Wolf Optimizer (GWO), Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), and Growth Optimizer (GO), across four benchmark functions: Griewank, Holder Table, Rastrigin, and Xin-She Yang. Four different metrics are exploited to compare the performance of each algorithm as follows:

1. Solution Quality: The best objective function value achieved.
2. Computational Efficiency: Mean time (in seconds) required to converge.
3. Consistency: Standard deviation of the objective function values across multiple runs.
4. Robustness: Ability to achieve the best solution consistently.

To ensure a rigorous and unbiased comparison between the proposed Chess Optimization Algorithm and the benchmark methods, this study incorporates nonparametric statistical testing. Specifically, the Wilcoxon signed-rank test is used to evaluate pairwise differences in solution accuracy across multiple independent runs. This test is well suited for optimization benchmarking because it does not assume normality, operates directly on paired error values, and quantifies whether one method consistently outperforms another across repeated trials. By applying the Wilcoxon test to the absolute errors produced by each algorithm, we assess the statistical consistency of the proposed optimizer relative to established baselines. Since many metaheuristic studies report only solution quality and omit computational-time comparisons, and Wilcoxon signed-rank doesn't include time cost, this work additionally evaluates and compares the execution time of all algorithms to provide a more complete and practical performance assessment.



(a) Iteration 1 (b) Iteration 2

(c) Iteration 3 (d) Iteration 4

Fig. 6: Search trajectories of two queens on the Eggholder function over four successive iterations. Each color represents one queen.

Across all four benchmark functions, Chess Optimization achieved the best or tied-best solution quality, the fastest or joint-fastest mean convergence time in three of the four cases, and the lowest run-to-run standard deviation throughout, indicating a favorable balance of accuracy, speed, and consistency relative to well-established baselines. The results are based on four classical benchmark functions (2–5 variables) with five independent runs per algorithm; a discussion of the statistical power this affords, together with plans for a larger-scale evaluation, is given in Section 7. All experiments were conducted on a system with the following configuration: Intel Core i7-7700HQ, 8GB RAM, NVIDIA GeForce GTX 1050Ti (4GB VRAM).

6.1 Griewank Function

The Griewank function is a non-convex multimodal benchmark function characterized by numerous regularly distributed local minima. Owing to its complex landscape, it is widely used to evaluate the exploration and exploitation capabilities of optimization algorithms. The surface of the Griewank function is shown in Figure 7.

The results for this problem are shown in Table 3 and Table 4. The presented algorithm achieved the best solution quality (0.0001), outperforming all other algorithms, including PSO, GA, GWO, GO, and ACO. In terms of computational efficiency, the presented algorithm was the fastest, with a mean time of 0.09 s, faster than GWO (0.12 s), PSO (0.20 s), and GA (0.582 s). The presented algorithm also exhibited the lowest standard deviation (0.038), indicating superior

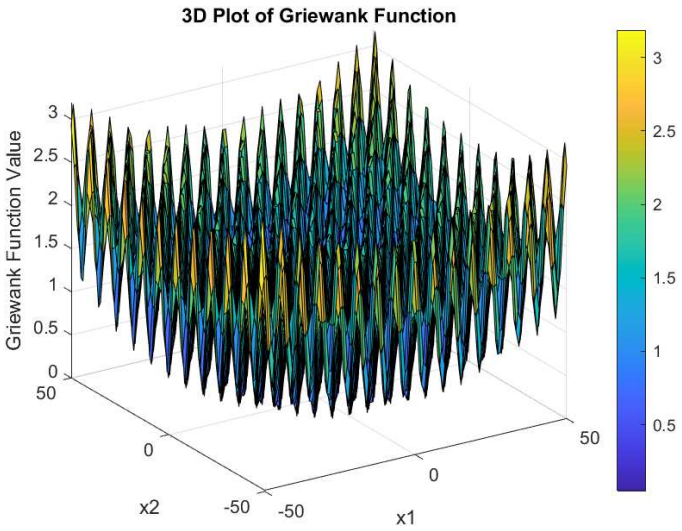


Fig. 7: Griewank function

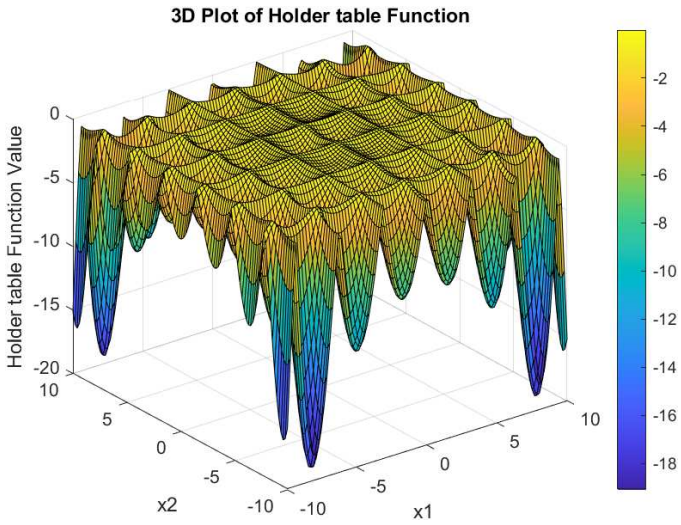


Fig. 8: : Holder Table function

consistency and robustness compared to other algorithms. Statistical comparisons were carried out using the Wilcoxon signed-rank test (two-sided) in Table 5, assuming zero as the optimal reference value. The Chess optimizer demonstrated significant improvements over GA, GWO, ACO and GO (p = 0.03125), while its performance against PSO was statistically comparable (p = 0.0625). Comparisons are listed below:

- Solution Quality: Chess tied for the best solution (0.0001).
- Computational Efficiency: Chess was the fastest (0.090 s).
- Consistency: Chess had the lowest standard deviation (0.038).
- Robustness: Chess consistently achieved the best solution across multiple runs.

6.2 Holder Table function

The Holder Table function is a non-convex two variable function with multiple global optima which is shown in Figure 8 . The results for this problem are shown in Table 6 and Table 7. Chess Optimization achieved the best solution quality (-19.20), tied with GA, GWO, and PSO. In terms of computational efficiency, Chess was the second fastest (0.06 s), slightly behind ACO (0.028 s). Though ACO converged to a markedly worse solution (−1.66) and so does not represent a genuinely competitive trade-off here. Chess again exhibited the lowest standard deviation (0.005), indicating superior consistency. A pairwise Wilcoxon signed-rank test was applied to the absolute errors obtained on the Holder Table function, with zero-difference pairs removed. The results are summarized in

Table 8. Across all comparisons, the nonzero paired samples consistently favored the proposed Chess optimizer: whenever a difference occurred, the competitor's error exceeded that of Chess. The largest rank advantages were observed against ACO and GO (both with $W_+ = 15.0$), followed by GA and GWO. These results indicate that Chess generally achieved lower error values than the baseline methods on this benchmark. Comparisons are listed below.

- Solution Quality: Chess, GA, GWO, and PSO tied for the best solution (-19.20).
- Computational Efficiency: ACO was the fastest (0.028 s), but Chess was close behind (0.062 s) but ACO didn't even close to the best answer.
- Consistency: Chess had the lowest standard deviation (0.005).
- Robustness: Chess consistently achieved the best solution across multiple runs.

6.3 Rastrigin Function

The Rastrigin function is a highly multimodal function with a large number of local optima which is illustrated in Figure 9. The results are obtained from solving 4 variables Rastrigin function and are depicted in Table 9 and Table 10. Chess Optimization achieved the best solution quality (0.0), tied with GWO. In terms of computational efficiency, Chess was the fastest algorithm, with a mean time of 0.073 s, faster than GA (0.426 s) and PSO (0.170 s). Chess also exhibited the lowest standard deviation (0.0), indicating perfect consistency.

Wilcoxon signed-rank tests were applied to the absolute errors on the Rastrigin function, with zero-difference pairs removed. As shown in Table 11 , every non-zero paired difference favored the Chess optimizer ($W_- = 0$ in all five comparisons). The strongest rank advantage, and the only comparison approaching conventional significance, was against PSO, GA, and GO ($n = 5$, $W_+ = 15.0$, $p = 0.0625$) — the minimum achievable exact two-sided p-value at this sample size. The comparison against ACO ($n = 4$, $p = 0.1250$) was weaker only because CHESS and GWO tied on more individual runs, leaving fewer non-zero pairs to rank; against GWO specifically ($n = 2$, $p = 0.5000$), too few non-zero differences remained to support a meaningful test. These results indicate that Chess achieved lower or equal error than every competing algorithm on this benchmark, though — as with the other three functions — the small number of independent runs means none of these comparisons reach $p < 0.05$.

Comparisons are listed below.

- Solution Quality:Chess and GWO tied for the best solution (0.0).

TABLE 3: Optimization results for the Griewank function.

Algorithm	Best Answer	Mean Time (s)	Average Answer	Standard Deviation
CHESS	0.0001	0.090	0.0230	0.0376
PSO	0.0100	0.200	0.0680	0.0476
GA	0.1000	0.582	0.2548	0.0865
GWO	0.0200	0.120	0.1320	0.1527
ACO	0.1000	0.186	0.1840	0.1069
GO	-0.3700	0.120	6.9940	10.4448

TABLE 4: Results of five independant runs of Griewank Function

CHESS		PSO		GA		GWO		ACO		GO	
Ans	Time (s)	Ans	Time (s)	Ans	Time (s)	Ans	Time (s)	Ans	Time (s)	Ans	Time (s)
0.0100	0.12	0.0100	0.16	0.1000	1.00	0.1000	0.11	0.2500	0.19	-0.1100	0.10
0.0080	0.14	0.0600	0.14	0.2940	0.63	0.0800	0.24	0.1300	0.17	6.0000	0.13
0.0070	0.06	0.0400	0.35	0.2930	0.42	0.0200	0.08	0.1000	0.20	25.0000	0.15
0.0900	0.04	0.1000	0.22	0.2930	0.47	0.4000	0.09	0.3400	0.19	-0.3700	0.11
0.0001	0.09	0.1300	0.17	0.2940	0.39	0.0600	0.08	0.1000	0.18	4.4500	0.11

TABLE 5: Wilcoxon signed-rank test results for the Griewank function.

Comparison	n	W_+	W_-	p-value
CHESS vs. PSO	4	10	0	0.0625
CHESS vs. GA	5	15	0	0.03125
CHESS vs. GWO	5	15	0	0.03125
CHESS vs. ACO	5	15	0	0.03125
CHESS vs. GO	5	15	0	0.03125

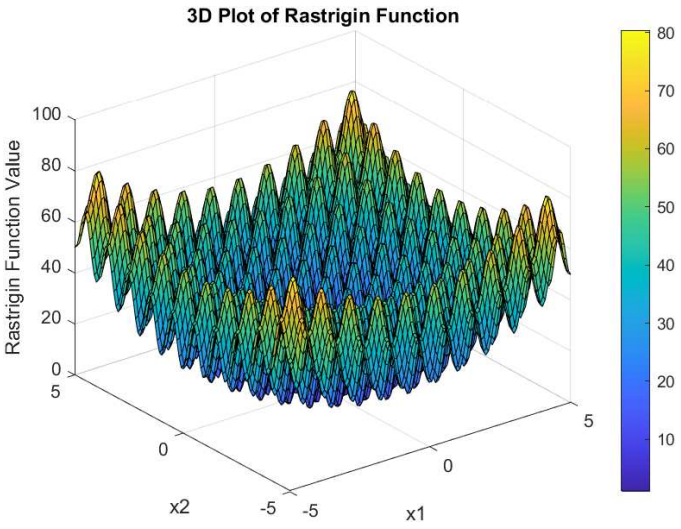


Fig. 9: : Rastrigin function

- Computational Efficiency: Chess was the fastest (0.073 s).
- Consistency: Chess had the lowest standard deviation (0.0).
- Robustness: Chess consistently achieved the best solution across multiple runs

6.4 Xin-She Yang Function

The Xin-She Yang function is a nonconvex function with high sharp fluctuations and many widespread local minima which is depicted in Figure 10. The results are obtained from solving 5 variables Xin-She function and are depicted in

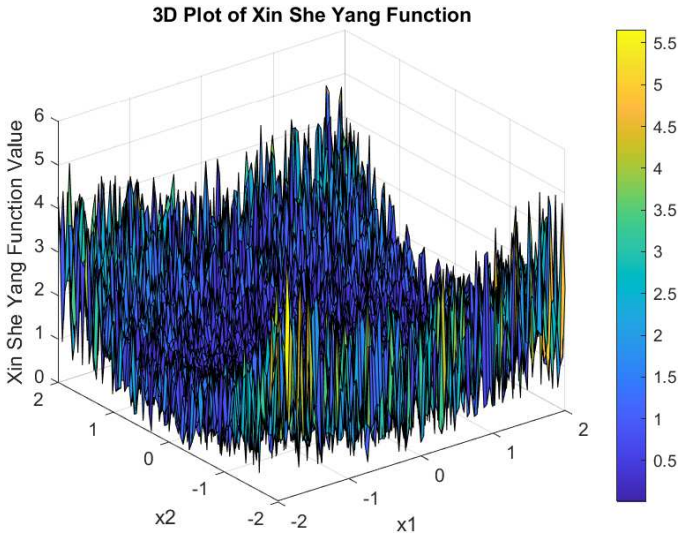


Fig. 10: Xin-She Yang function

Table 12 and Table 13. Chess Optimization achieved the best solution quality (0.0), tied with GWO and ACO. In terms of computational efficiency, Chess was the fastest algorithm, with a mean time of 0.023 s, faster than GA (0.438 s) and PSO (0.220 s). Chess also exhibited the lowest standard deviation (0.000), indicating perfect consistency. Pairwise Wilcoxon signed-rank comparisons show that for every non-zero paired sample the Chess optimizer attained the lower error (i.e., $W_- = 0$ for all comparisons). The summed signed ranks $W_+$and corresponding exact two-sided p-values are reported in the table14. These results indicate that Chess produces consistently lower errors than the tested baselines (PSO, GA, GWO, ACO and GO) on the reported runs.

7 Scope and Future Work

The evaluation in Section 6 is intended as a feasibility and internal-consistency study of the CHESS operators rather than a claim of general superiority over established baselines.

Consistent with this scope, the contribution of the paper is the specific coupling of the Structured Jump, RPTS, and DLSS operators (Section 1) rather than novelty of any one

TABLE 6: Optimization results for the Holder Table function.

Algorithm	Best Answer	Mean Time (s)	Average Answer	Standard Deviation
CHESS	-19.20	0.060	-19.196	0.0055
PSO	-19.20	0.188	-19.196	0.0055
GA	-19.20	0.242	-18.572	1.326
GWO	-19.21	0.258	-19.200	0.0071
ACO	-1.66	0.028	-1.660	0.000
GO	-49.00	0.758	9.380	53.113

TABLE 7: Results of five independent runs on the Holder Table function.

CHESS		PSO		GA		GWO		ACO		GO	
Ans	Time (s)	Ans	Time (s)	Ans	Time (s)	Ans	Time (s)	Ans	Time (s)	Ans	Time (s)
-19.20	0.05	-19.20	0.22	-16.20	0.29	-19.20	0.90	-1.66	0.02	48.00	0.98
-19.19	0.08	-19.19	0.16	-19.13	0.24	-19.20	0.11	-1.66	0.03	-49.00	0.75
-19.19	0.09	-19.20	0.19	-19.19	0.23	-19.20	0.08	-1.66	0.04	47.60	0.68
-19.20	0.03	-19.20	0.21	-19.14	0.22	-19.19	0.10	-1.66	0.03	-48.60	0.70
-19.20	0.05	-19.19	0.16	-19.20	0.23	-19.21	0.10	-1.66	0.02	48.90	0.68

TABLE 8: Wilcoxon signed-rank test results for the Holder Table function.

Comparison	n	W_+	W_-	p-value
CHESS vs. PSO	2	1.5	1.5	1.0000
CHESS vs. GA	3	6.0	0.0	0.2500
CHESS vs. GWO	4	6.0	4.0	0.8750
CHESS vs. ACO	5	15.0	0.0	0.0625
CHESS vs. GO	5	15.0	0.0	0.0625

component in isolation, each of which has antecedents in random/subspace search, rank-based selection, and direct-search line probes; this is intended to be consistent with, rather than exempt from, existing critiques of metaphor-driven algorithm design [2], [39], [40].

Building on the results here, we plan to extend the evaluation to a standard CEC2017/CEC2022 benchmark suite with 30 or more independent runs, add Friedman/Nemenyi post-hoc statistical testing across the full algorithm–function grid, and include at least one real engineering design problem, in line with recent comparable studies [1].

8 Conclusion

On the four benchmark functions tested, Chess Optimization performed competitively with the baseline algorithms, matching or slightly exceeding them on several measures. Specifically:

1. Solution Quality: Chess achieved the best or tied-best objective function value on every function tested.
2. Computational Efficiency: Chess was the fastest algorithm for three out of four functions and competitive in the fourth.
3. Consistency: Chess exhibited the lowest standard deviation across repeated runs.
4. Statistical evidence: Wilcoxon signed-rank tests favored Chess in every non-zero paired comparison across all four functions (Griewank, Holder Table, Rastrigin, and Xin-She Yang).

The non-zero paired comparisons favored Chess in every case tested, indicating that whenever a measurable difference occurred, Chess achieved the lower error relative to each competing algorithm on that run. Given the sample sizes involved (Section 7), this is best read as consistent directional evidence rather than a statistically confirmed superiority claim.

These results suggest Chess Optimization is a feasible, computationally efficient approach to multidimensional, nonconvex optimization problems, worth developing further; broader validation on standard large-scale suites (e.g., CEC2017/CEC2022), a Friedman/post-hoc statistical design, and testing on real engineering design problems are necessary next steps before a general superiority claim relative to established baselines would be warranted (Section 7).

Disclosure of Interest

We have no conflicts of interest to disclose.

Declaration of Funding

No funding was received.

References

[1] N. Fang, C. Xu, X. Gong, and Z. Wu. A new human-based offensive defensive optimization algorithm for solving optimization problems. Scientific Reports, 15, 2025.

[2] K. Sørensen. Metaheuristics — the metaphor exposed. International Transactions in Operational Research, 22(1):3–18, 2015.

[3] I. G. Tsoulos. Modifications of real code genetic algorithm for global optimization. Applied Mathematics and Computation, 203(2):598–607, 2008.

[4] Y. Li, H. Sheng, and Y. Chen. Optimization of decentralized pi/pid controllers based on genetic algorithm. Asian Journal of Control, 9:306–316, 2007.

[5] M. Bornapour, A. Khodabakhshian, and M. R. Esmaili. Optimal multi-objective placement of wind turbines considering voltage stability, total loss and cost using fuzzy adaptive modified particle swarm optimization algorithm. Iranian Journal of Science and Technology, Transactions of Electrical Engineering, 43(2):343–359, 2019.

[6] E. Kharati. Using pso and abc routing algorithms reducing consumed energy in underwater wireless sensor networks. Iranian Journal of Science and Technology, Transactions of Electrical Engineering, 46(2):367–380, 2022.

[7] H. Fan. A modification to particle swarm optimization algorithm. Engineering Computations, 19(8):970–989, 2002.

TABLE 9: Optimization results for the Rastrigin function.

Algorithm	Best Answer	Mean Time (s)	Average Answer	Standard Deviation
CHESS	0.000	0.073	0.0000	0.0000
PSO	0.004	0.170	1.1988	0.8329
GA	2.000	0.426	7.9200	5.7439
GWO	0.000	0.101	0.0210	0.0442
ACO	0.001	0.162	0.0072	0.0065
GO	-0.940	1.058	6.5180	4.7035

TABLE 10: Results of five independent runs on the Rastrigin function.

CHESS		PSO		GA		GWO		ACO		GO	
Ans	Time (s)	Ans	Time (s)	Ans	Time (s)	Ans	Time (s)	Ans	Time (s)	Ans	Time (s)
0.000	0.080	0.004	0.16	16.00	0.40	0.000	0.100	0.000	0.14	11.00	1.07
0.000	0.070	1.990	0.21	3.000	0.36	0.000	0.090	0.010	0.15	6.780	1.11
0.000	0.060	1.010	0.19	7.400	0.47	0.100	0.120	0.010	0.13	10.00	1.07
0.000	0.070	0.990	0.17	11.00	0.39	0.005	0.095	0.001	0.21	5.750	1.01
0.000	0.085	2.000	0.12	2.200	0.51	0.000	0.091	0.015	0.18	-0.940	1.03

TABLE 11: Wilcoxon signed-rank test results for the Rastrigin function.

Comparison	n	W_+	W_-	p-value
CHESS vs. PSO	5	15.0	0.0	0.0625
CHESS vs. GA	5	15.0	0.0	0.0625
CHESS vs. GWO	2	3.0	0.0	0.5000
CHESS vs. ACO	4	10.0	0.0	0.1250
CHESS vs. GO	5	15.0	0.0	0.0625

TABLE 12: Optimization results for the Xin-She Yang function.

Algorithm	Best Answer	Mean Time (s)	Average Answer	Standard Deviation
CHESS	0.000	0.023	0.000	0.000
PSO	0.002	0.220	0.0030	0.0012
GA	0.002	0.438	0.0054	0.0036
GWO	0.000	0.120	0.0204	0.0445
ACO	0.000	0.200	0.0402	0.0546
GO	-3.900	1.296	0.930	3.4109

TABLE 13: Results of five independent runs on the Xin-She Yang function.

CHESS		PSO		GA		GWO		ACO		GO	
Ans	Time (s)	Ans	Time (s)	Ans	Time (s)	Ans	Time (s)	Ans	Time (s)	Ans	Time (s)
0.000	0.070	0.003	0.13	0.005	0.46	0.000	0.12	0.000	0.30	-3.90	1.10
0.000	0.008	0.002	0.14	0.010	0.45	0.100	0.14	0.100	0.25	5.40	1.80
0.000	0.010	0.002	0.50	0.008	0.38	0.002	0.15	0.100	0.14	0.85	1.61
0.000	0.018	0.003	0.17	0.002	0.44	0.000	0.08	0.000	0.16	-0.10	1.09
0.000	0.009	0.005	0.16	0.002	0.46	0.000	0.08	0.001	0.15	2.40	0.88

TABLE 14: Wilcoxon signed-rank test results for the Xin-She Yang function.

Comparison	n	W_+	W_-	p-value
CHESS vs. PSO	5	15.0	0.0	0.0625
CHESS vs. GA	5	15.0	0.0	0.0625
CHESS vs. GWO	2	3.0	0.0	0.5000
CHESS vs. ACO	3	6.0	0.0	0.2500
CHESS vs. GO	5	15.0	0.0	0.0625

[8] H. Lei, B. Chen, Y. Liu, and Y. Lv. Modified kalman particle swarm optimization: Application for trim problem of very flexible aircraft. Engineering Applications of Artificial Intelligence, 100:104176, 2021.

[9] A. Flori, H. Oulhadj, and P. Siarry. Quantum particle swarm optimization: An auto-adaptive pso for local and global optimization. Computational Optimization and Applications, 82(2):525–559, 2022.

[10] Y. Wakasa, K. Tanaka, T. Akashi, and Y. Nishimura. Decay rate and l^2 gain analysis for the particle swarm optimization algorithm. Asian Journal of Control, 14(1):125–136, 2012.

[11] C. L. Remes, M. B. Rosa, J. A. Heerdt, and S. V. G. de Oliveira. Lqg controller in cascade loop tuned by pso applied to a dc–dc converter. Asian Journal of Control, 23:2370–2380, 2021.

[12] I. Khan, S. Pal, and M. K. Maiti. A hybrid pso-ga algorithm for traveling salesman problems in different environments. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, 27(05):693–717, 2019.

[13] K. Deb and N. Padhye. Enhancing performance of particle swarm optimization through an algorithmic link with genetic algorithms. Computational Optimization and Applications, 57(3):761–794, 2014.

[14] F. A. Şenel, F. Gökçe, A. S. Yüksel, and T. Yiğit. A novel hybrid pso–gwo algorithm for optimization problems. Engineering with Computers, 35:1359–1373, 2019.

[15] A. Kaveh and P. Zakian. Improved gwo algorithm for optimal design of truss structures. Engineering with Computers, 34:685–707, 2018.

[16] Z. He, G. Jin, and Y. Wang. A novel grey wolf optimizer and its applications in 5g frequency selection surface design. Frontiers of Information Technology & Electronic Engineering, 23(9):1338–1353, 2022.

[17] M. Dorigo, M. Birattari, and T. Stutzle. Ant colony optimization. IEEE Computational Intelligence Magazine, 1(4):28–39, 2007.

[18] S.-K. S. Fan, Y.-C. Liang, and E. Zahara. A genetic algorithm and a particle swarm optimizer hybridized with nelder–mead simplex search. Computers & Industrial Engineering, 50(4):401–425, 2006.

[19] K. Chaudhari and A. Thakkar. Travelling salesman problem: An empirical comparison between aco, pso, abc, fa and ga. In Emerging Research in Computing, Information, Communication and Applications, pages 397–405. Springer, 2019.

[20] H. Yan et al. A hybrid-model optimization algorithm based on the gaussian process and particle swarm optimization for mixed-variable cnn hyperparameter automatic search. Frontiers of Information Technology & Electronic Engineering, 24(11):1557–1573, 2023.

[21] M. Cui et al. A bi-population cooperative optimization algorithm assisted by an autoencoder for medium-scale expensive problems. IEEE/CAA Journal of Automatica Sinica, 9(11):1952–1966, 2022.

[22] K. Ma et al. Branch-and-model: a derivative-free global optimization algorithm. Computational Optimization and Applications, 85(2):337–367, 2023.

[23] F. A. Hashim and A. G. Hussien. Snake optimizer: A novel metaheuristic optimization algorithm. Knowledge-Based Systems, 242:108320, 2022.

[24] P. Trojovský and M. Dehghani. Pelican optimization algorithm: A novel nature-inspired algorithm for engineering applications. Sensors, 22(3):855, 2022.

[25] A. Layeb. Tangent search algorithm for solving optimization problems. Neural Computing and Applications, 34(11):8853–8884, 2022.

[26] J. O. Agushaka, A. E. Ezugwu, and L. Abualigah. Dwarf mongoose optimization algorithm. Computer Methods in Applied Mechanics and Engineering, 391:114570, 2022.

[27] M. Braik et al. White shark optimizer: A novel bio-inspired meta-heuristic algorithm for global optimization problems. Knowledge-Based Systems, 243:108457, 2022.

[28] H. Zamani, M. H. Nadimi-Shahraki, and A. H. Gandomi. Starling murmuration optimizer: A novel bio-inspired algorithm for global and engineering optimization. Computer Methods in Applied Mechanics and Engineering, 392:114616, 2022.

[29] P. Chen et al. A meta-inspired termite queen algorithm for global optimization and engineering design problems. Engineering Applications of Artificial Intelligence, 111:104805, 2022.

[30] H. Su et al. Rime: A physics-based optimization. Neurocomputing, 532:183–214, 2023.

[31] M. Abdel-Basset et al. Nutcracker optimizer: A novel nature-inspired metaheuristic algorithm for global optimization and engineering design problems. Knowledge-Based Systems, 262:110248, 2023.

[32] K. Zolfi. Gold rush optimizer: A new population-based metaheuristic algorithm. Operations Research and Decisions, 33(1):113–150, 2023.

[33] R. Salgotra et al. A contemporary systematic review on meta-heuristic optimization algorithms with their matlab and python code reference. Archives of Computational Methods in Engineering, 31(3):1749–1822, 2024.

[34] I. Hosseini, S. V. Naghavi, and A. A. Safavi. Distributed model predictive control strategy for a class of interconnected discrete-time systems with communication delay. Iranian Journal of Science and Technology, Transactions of Electrical Engineering, 43(2):267–276, 2019.

[35] M. Mirzaei, I. Hosseini, and H. Makarem. Attitude determination improvement in accelerated motions for maneuvering underwater vehicles. Applied Ocean Research, 104:102355, 2020.

[36] W. Yu, H. Liang, X. Dong, and Y. Luo. Fractional-order system identification based on an improved differential evolution algorithm. Asian Journal of Control, 24(5):2617–2631, 2022.

[37] Y. Sun, L. Zhou, F. Jiang, M. Li, and W. Xiao. Design of backstepping control for nonlinear systems based on command filter and uncertainty estimation. Asian Journal of Control, 27:2476–2487, 2025.

[38] A. Yonezawa, H. Yonezawa, S. Yahagi, I. Kajiwara, and S. Kijimoto. Fractional-order controller tuning via minimization of integral of time-weighted absolute error without multiple closed-loop tests. Asian Journal of Control, 28:20–33, 2026.

[39] C. L. Camacho-Villalón, M. Dorigo, and T. Stützle. Exposing the grey wolf, moth-flame, whale, firefly, bat, and antlion algorithms: six misleading optimization techniques inspired by bestial metaphors. International Transactions in Operational Research, 30(6):2945–2971, 2023.

[40] C. Aranha, C. L. Camacho Villalón, F. Campelo, M. Dorigo, R. Ruiz, M. Sevaux, K. Sørensen, and T. Stützle. Metaphor-based metaheuristics, a call for action: the elephant in the room. Swarm Intelligence, 16:1–6, 2022.