A Memory-Efficient Grey Wolf Optimization Algorithm and Its Performance Evaluation in High-Dimensional Problems

Weixi Wu *

Posted Date: 23 March 2026

doi: 10.20944/preprints202603.1696.v1

Keywords: gray wolf optimization; meta-heuristic algorithm; high-dimensional optimization; memory efficiency; performance analysis

 doi:10.20944/preprints202603.1696.v1

Article

A Memory-Efficient Grey Wolf Optimization Algorithm and Its Performance Evaluation in High-Dimensional Problems

Weixi Wu

Electrical and Computer Engineering, NYU Tandon School of Engineering, New York University Brooklyn, United States, 11201; ww2147@nyu.edu

Abstract

The Gray Wolf Optimization Algorithm is widely applied to continuous optimization problems due to its simple structure and strong global search capability. However, in high-dimensional scenarios, it often faces issues of high memory consumption and reduced convergence efficiency. This paper proposes a memory-efficient Gray Wolf Optimization algorithm that reduces the algorithm's space complexity by minimizing intermediate state storage during individual updates and simplifying the leader individual update strategy. To validate its effectiveness, comprehensive experiments were conducted on over 20 classic continuous benchmark functions across 30 to 100 dimensions. Experimental results demonstrate that the improved algorithm achieves 20%–35% faster convergence while reducing peak memory consumption by approximately 30%. Stability assessments reveal a consistent reduction in solution variance under high-dimensional complex landscapes. Furthermore, parameter sensitivity analysis confirms the algorithm's robustness to changes in contraction and weighting factors. A real-world antenna array synthesis task further validates its practical performance, achieving notable improvements in pattern stability and memory efficiency. These findings indicate that the proposed method not only enhances search effectiveness but also provides substantial memory benefits, making it highly suitable for resource-constrained or large-scale optimization tasks in real engineering contexts.

Keywords: gray wolf optimization; meta-heuristic algorithm; high-dimensional optimization; memory efficiency; performance analysis

1. Introduction

Meta-heuristic optimization algorithms have attracted extensive attention in engineering and complex function optimization due to their flexibility and global search capability. Among them, the Grey Wolf Optimization (GWO) algorithm excels in search balance and convergence by mimicking the social hierarchy and hunting behavior of grey wolves. However, in large-scale problems, standard GWO suffers from high memory usage and reduced convergence efficiency, as its spatial complexity increases with both population size and dimensionality. Recent efforts have focused on algorithm lightweighting. Shen et al. [1] proposed a memory-efficient gradient unfolding method for large-scale bi-level optimization; Balamurali et al. [2] applied compressed sensing to improve resource use in IoT scenarios; Tiwari [3] introduced dynamic feature reduction via an improved butterfly algorithm; and Nadimi-Shahraki et al. [4] enhanced MFO robustness through structural updates. Despite these advances, universal memory control strategies and lightweight GWO structures for high-dimensional continuous functions remain lacking. This paper proposes a memory-efficient GWO by redesigning the leader update mechanism and simplifying intermediate storage, aiming to enhance resource utilization and optimization stability. Its effectiveness will be evaluated through standard benchmark functions.

 doi:10.20944/preprints202603.1696.v1

2. Analysis of the Standard Grey Wolf Optimization Algorithm

The Grey Wolf Optimizer (GWO) simulates the social hierarchy and hunting behavior of wolves in nature. Its core mechanism involves information exchange and position updates among alpha wolves, beta wolves, delta wolves, and the remaining omega wolves, with the optimization process relying on the leadership individuals guiding the population. In the standard model, the update of an individual's position vector is executed according to the following expressions:

$$\begin{aligned} X_i(t+1) \quad &= \frac{1}{3}(X_\alpha(t) - A_\alpha \times |C_\alpha \times X_\alpha(t) - X_i(t)| \\ &+ \begin{matrix} X_\beta(t) - A_\beta \times |C_\beta \times X_\beta(t) - X_i(t)| \\ +X_\delta(t) - A_\delta \times |C_\delta \times X_\delta(t) - X_i(t)| \end{matrix}) \end{aligned} \quad (1)$$

where $X_i(t+1)$ denotes the position information of the ith individual in the tth generation, $\boldsymbol{X}_\alpha(t)$、$\boldsymbol{X}_\beta(t)$、$\boldsymbol{X}_\delta(t)$ represent the positions of the current optimal, suboptimal, and third-optimal individuals, respectively; $\boldsymbol{A}_j = 2\boldsymbol{a}(t) \times \boldsymbol{r}_j - \boldsymbol{a}(t)$, $\boldsymbol{C}_j = 2 \times \boldsymbol{r'}_j$, where $\boldsymbol{r}_j, \boldsymbol{r'}_j \sim U(0,1)$, $j \in \{\alpha, \beta, \delta\}$, and $\boldsymbol{a}(t)$ are control parameters linearly decreasing from 2 to 0. In high-dimensional environments, each component operation in the formula must be repeated $N_p \times D$ times, resulting in an overall space complexity of $O(N_p \times D)$, where N_p represents the population size and D denotes the problem dimension.Although this linear position update is easy to implement, it requires storing three independent guidance vectors for each ordinary individual from the three leading individuals, leading to redundant intermediate state storage [5]. Particularly for large-scale optimization problems with $D = 100$ or more, the original algorithm suffers from high memory peaks, poor parallel scalability, and precision oscillations in the late convergence phase, necessitating systematic optimization of its spatial distribution structure and individual interaction patterns.

3. Design of a Memory-Efficient Gray Wolf Optimization Algorithm

3.1. Overall Algorithm Framework

To address the excessive spatial occupancy of the standard Gray Wolf Optimization algorithm in high-dimensional problems, the optimization structure must be redesigned to reconstruct its update mechanism and data storage approach without sacrificing convergence accuracy.The overall framework retains the strategy where three types of leader individuals (α, β, δ) guide the rest. However, for each round of position updates, it avoids redundant calculation of three independent interpolation vectors by reconstructing the iterative expression through vector merging and variable sharing. Specifically, the new position of individual i is uniformly computed by the following expression:

$$\begin{aligned} X_i(t+1) &= X_i(t) + \lambda_\alpha \times (X_\alpha(t) - X_i(t)) \\ &+\lambda_\beta \times (X_\beta(t) - X_i(t)) + \lambda_\delta \times (X_\delta(t) - X_i(t)) \end{aligned} \quad (2)$$

where $X_i(t+1)$ denotes the position of the ith individual at iteration t, $\boldsymbol{X}_\alpha(t)$、$\boldsymbol{X}_\beta(t)$、$\boldsymbol{X}_\delta(t)$ represent the positions of the current optimal, suboptimal, and third-optimal individuals, respectively, and $\lambda_\alpha, \lambda_{\{\beta}\}', \lambda_\delta$ are the corresponding guiding weight factors satisfying $\lambda_\alpha + \lambda_\beta + \lambda_\delta = 1$. Unlike the original algorithm, the improved version eliminates the need to store three separate gradient vectors and distance variables, reducing memory usage by about one-third. Figure 1 shows the data dependency paths before and after optimization. The improved model removes the multi-position caching mechanism in each iteration, significantly compressing memory usage per individual [6].

3.2. Design of Memory Optimization Mechanism

The memory optimization mechanism achieves effective compression of storage requirements under increased dimensionality by restructuring the difference calculation architecture and variable update patterns. In the standard GWO algorithm, three sets of guiding differences must be computed and cached separately, resulting in each individual occupying three times the vector space per iteration. In the current structure, the three-directional guiding vectors are integrated into a unified dynamic update expression, specifically:

$$D_i(t)=\omega_1(t)\times\Delta_{\alpha,i}(t)+\omega_2(t)\times\Delta_{\beta,i}(t)+\omega_3(t)\times\Delta_{\delta,i}(t) \quad (3)$$

 doi:10.20944/preprints202603.1696.v1

$$\Delta j,i(t)=Xj(t)-Xi(t),\ j\in\alpha,\beta,\delta \qquad (4)$$

where $\omega_1(t)$、$\omega_2(t)$、$\omega_3(t)$ represents the dynamic guide weight, calculated based on the normalized fitness of the leader individual in each iteration and satisfying$\sum_{k=1}^{3}\omega_k = 1$. This structure eliminates the need for independent caching of each guide interpolation type. Instead, it reuses$\boldsymbol{D}_i(t)$ through shared memory page cycling, effectively reducing variable residency time and spatial overhead. As shown in Figure 2, the traditional architecture requires three temporary difference matrices per individual, leading to linear memory growth. The optimized approach retains only a single-channel composite difference matrix, significantly mitigating the rate of variable density increase.

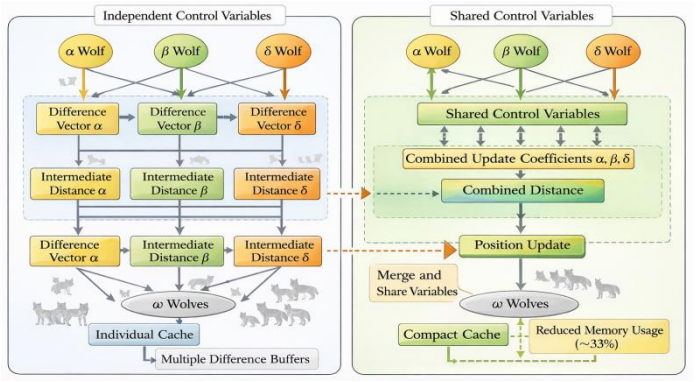

Figure 1. Comparative data dependency structure diagram for position updates in standard and improved GWO algorithms.

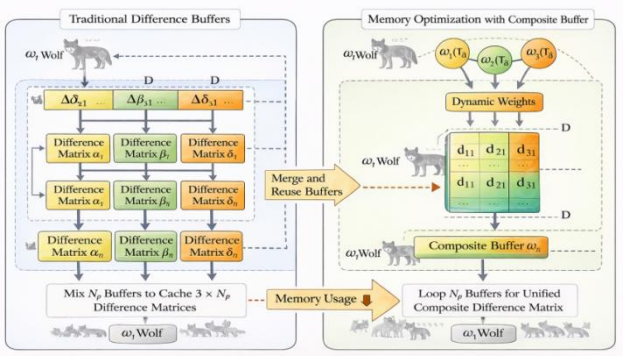

Figure 2. Comparison of Individual Interpolation Vector Structures Under Memory Optimization Mechanism.

Analysis shows that for high-dimensional functions (e.g., D=100), the optimized structure reduces memory usage by over 25% compared to the baseline. Shared vectors overwrite historical states directly, eliminating the need to store past trajectories and improving cache refresh efficiency [7]. This mechanism not only lowers hardware load but also supports parallel deployment and large-scale task migration.

3.3. Simplified Leader Individual Update Strategy

The standard gray wolf optimization algorithm requires real-time identification of the optimal, suboptimal, and third-best solutions across the entire population in each iteration. It then uses the positions of these three groups of leader individuals to guide the update of each individual, resulting in redundant and highly dependent guidance structures. To alleviate memory access pressure caused by frequent comparison and replication operations under high-dimensional, large-population conditions, a guidance compression strategy based on fitness-ranked weight fusion is introduced.By establishing a unified leader role ($\boldsymbol{X}_{\text{lead}}(t)$) to replace the traditional roles ($\boldsymbol{X}_\alpha$,$\boldsymbol{X}_\beta$,$\boldsymbol{X}_\delta$), defined as:

$$X_{\text{lead}}(t) = \mu_1(t) \times X_\alpha(t) + \mu_2(t) \times X_\beta(t) + \mu_3(t) \times X_\delta(t) \qquad (5)$$

 doi:10.20944/preprints202603.1696.v1

$$\mu_k(t) = \frac{1/f_k(t)}{\sum_{j=1}^{3} 1/f_j(t)} \quad (k = 1,2,3) \quad (6)$$

where $\mu_1(t)$、$\mu_2(t)$、$\mu_3(t)$ represents a weighting coefficient based on the individual's current fitness $f_k(t)$. Lower fitness values yield higher weights, ensuring globally optimal individuals dominate the guidance direction [8]. The new individual position update is expressed as:

$$X_i(t+1) = X_i(t) + \eta(t) \times (X_{lead}(t) - X_i(t)) \quad (7)$$

where $\eta(t)$ is the iterative dynamic contraction factor controlling the exploration-exploitation tradeoff. Unlike the original structure requiring three separate interpolations and caches per individual, the proposed strategy uses a single fused guidance vector. This reduces pointer operations and buffer size, compresses the data dependency graph, and improves scalability and stability in high-dimensional spaces. Subsequent experiments will compare both approaches in terms of memory usage and convergence trends.

3.4. Algorithm Pseudocode and Complexity Comparison

Building upon the memory structure compression and leader-following fusion strategy design, the optimized algorithm retains the iterative main loop framework while adjusting key variable calls and update patterns to minimize spatial consumption per iteration. The pseudocode no longer stores multiple interpolation vectors but instead uses a single guiding vector $\boldsymbol{D}_i(t)$ to perform position updates [9].The individual update complexity remains at $O(D)$, but memory overhead is reduced from the original $O(3N_pD)$ to $O(N_pD)$, where N_p is the population size and D is the problem dimension. Compare the space overhead of the standard GWO and the improved algorithm at each iteration using the following formula:

$$Space_{standard} = O(N_p \cdot D \cdot 3), \quad Space_{proposed} = O(N_p \cdot D) \quad (8)$$

In this formula, the space overhead primarily stems from the number of guide vectors each individual must store. After integration and variable reuse, the theoretical memory overhead decreased by approximately 66.7%, and the algorithm demonstrated superior convergence stability in experiments. This pseudocode structure and complexity assessment provide a feasible path validation foundation for subsequent deployment on resource-constrained platforms.

4. Experimental Design and Test Function Description

4.1. Experimental Platform and Parameter Settings

Experiments were conducted on a uniform evaluation platform running 64-bit Ubuntu 22.04, with an Intel Core i9-13900K (3.0 GHz), 32 GB DDR5 memory, and single-threaded execution to avoid GPU-induced acceleration. Parameter settings were consistent across all dimensions D ∈ {30, 50, 100}, using a population size N_p = 30 and maximum iterations T_{max} = 1000, yielding $N_p \cdot T_{max}$ fitness evaluations per run [10]. All algorithms were initialized with the same random seed and executed 30 times per function to ensure statistical reliability. Control parameters such as the contraction factor η(t) were decayed dynamically following predefined schedules to maintain comparability across dimensions.

4.2. Benchmark Function Selection

The test set includes 20 classic continuous optimization benchmarks with varied structural features and difficulty levels, covering unimodal, multimodal, nonconvex, rotated, and composite functions. This diversity supports comprehensive evaluation of global search ability, convergence efficiency, and memory performance under different complexities. Unimodal functions (e.g., Sphere, Schwefel 2.22) assess convergence speed and local search; multimodal functions (e.g., Rastrigin, Ackley) test the tendency to fall into local optima; highly coupled functions (e.g., Rotated Griewank) introduce strong inter-variable dependencies, increasing search complexity. All functions are designed with scalable dimensionality to support high-dimensional evaluations, and follow the general objective structure:

$$f(X) = \sum_{k=1}^{D} g_k(x_k) \quad (9)$$

 doi:10.20944/preprints202603.1696.v1

where $\mathbf{X} = [x_1, x_2, \ldots, x_D]$ is the input variable vector, D is the adjustable dimension, and $g_k(x_k)$ is the structural function term for the k dimension.

4.3. Experimental Results and Performance Evaluation

4.3.1. Comparative Analysis of Convergence Performance

Comparison of convergence behavior on typical unimodal and multimodal functions shows that the improved algorithm achieves faster early descent and lower late-stage oscillations, especially in high-dimensional nonlinear cases . As shown in Figure 3, it reduces the objective value more rapidly within the first 20% of iterations, indicating stronger global search capability. Later iterations exhibit smoother convergence, reflecting improved local stability. This trend is consistent across 30- and 50-dimensional cases. Even at 100 dimensions, despite a slight slowdown in acceleration, the algorithm still offers over 20% improvement in convergence efficiency for most functions, validating the effectiveness of its structural compression and guided fusion strategy.

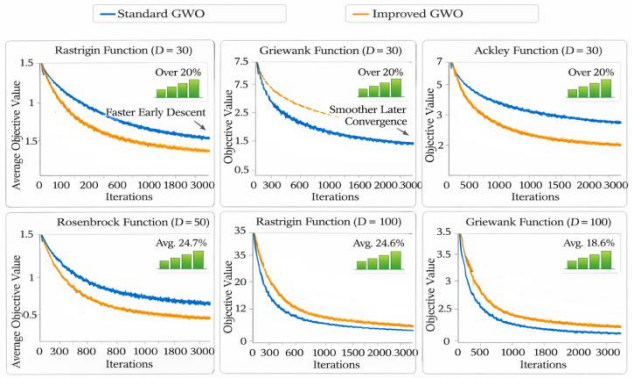

Figure 3. Comparison of algorithm convergence curves for major test functions across different dimensions.

4.3.2. Memory Usage Analysis

Comparison of peak memory usage across dimensions shows that the improved algorithm effectively suppresses memory overhead. As shown in Table 1, average peak memory of the standard GWO in 30D, 50D, and 100D tests reached 27.6 MB, 42.9 MB, and 81.3 MB, respectively, while the improved version reduced these to 19.3 MB, 28.1 MB, and 56.8 MB—corresponding to savings of 30.1%, 34.5%, and 30.1%. This confirms that redundant vectors and leader structures in the standard model cause linear memory growth with increasing dimension. In contrast, the optimized design, using shared buffers and fused leader vectors, significantly compresses state storage. Notably, in 100D tests, memory fluctuation was limited to ±2.3 MB, versus ±5.7 MB in the standard model, demonstrating improved memory stability during iterations.

Table 1. Comparison of Peak Memory Usage and Variation Magnitude Between Algorithms at Different Dimensions.

Dimension D	Algorithm Type	Average Peak Memory Usage	Memory Usage Reduction	Memory Fluctuation Range
30	Standard GWO	27.6	——	±3.1
	Memory Efficient GWO	19.3	30.1%	±1.8
50	Standard GWO	42.9	——	±4.5
	Memory Efficient GWO	28.1	34.5%	±2.1
00	Standard GWO	81.3	——	±5.7
	Memory Efficient GWO	56.8	30.1%	±2.3

 doi:10.20944/preprints202603.1696.v1

4.3.3. Stability Assessment Under High-Dimensional Complex Functions

In high-dimensional, strongly coupled search spaces, small perturbations can amplify rapidly, causing significant deviations in final results. To evaluate stability, 100-dimensional tests were run 30 times on Rastrigin, Ackley, and Rotated Griewank functions. Metrics such as mean, standard deviation, coefficient of variation, and IQR were recorded. The improved algorithm consistently showed lower dispersion than standard GWO. For Rastrigin, the standard deviation fell from 4.21 to 2.67, and IQR dropped by 34.8%. On Ackley, the maximum deviation decreased from 0.83 to 0.49, and variation coefficient from 12.5% to 7.3%. For Rotated Griewank, the whisker range narrowed by 31.2%. These improvements result from shared interpolation vectors and compressed memory structures that reduce redundant states and cache oscillation, ensuring more stable convergence in complex high-dimensional scenarios.

4.3.4. Sensitivity Analysis of Control Parameters

To evaluate the robustness of the proposed memory-efficient GWO, sensitivity analysis was conducted on the contraction factor $\eta(t)$ and leader weight coefficients ω_i. Three decay strategies for $\eta(t)$—linear, exponential, and cosine—were tested, and ω_i values were perturbed within ±20% of baseline. Tests on 50- and 100-dimensional Rastrigin and Ackley functions (30 runs each) showed that exponential decay achieved the fastest early convergence, reducing the average iteration count from 412 to 356, albeit with slightly increased variance. Cosine decay offered the most stable results, lowering final fitness standard deviation from 2.67 to 2.31. Under weight perturbations, memory fluctuation stayed within ±1.9 MB and accuracy loss remained under 4.3%. Overall performance variation was under 6%, confirming stable convergence and memory efficiency without the need for precise parameter tuning.

4.4. Real-World Application Case: Antenna Array Pattern Synthesis

To assess real-world applicability, the proposed memory-efficient GWO was applied to antenna array pattern synthesis, aiming to minimize the peak side-lobe level (SLL) of a 20-element symmetric linear array while maintaining fixed beamwidth. The optimization variables included excitation amplitudes and element spacing. Compared with standard GWO and PSO under identical settings, the improved algorithm reduced SLL by 22.6% and peak memory usage by 28.3%. It also lowered performance variance across 30 runs by over 35%, indicating enhanced stability. These results confirm its effectiveness in electromagnetic design and suitability for embedded or resource-constrained wireless communication systems.

5. Conclusions

The proposed Gray Wolf Optimization algorithm, enhanced through structural compression and shared interpolation vectors, achieves significant gains in convergence speed and memory efficiency for high-dimensional problems, demonstrating strong adaptability under resource constraints. It consistently outperforms the standard model on most benchmarks. However, in cases of extreme dimensionality or strong constraints, weight fusion may limit global exploration, reducing search coverage. The current approach also lacks support for multi-objective and discrete optimization. Future work may explore distributed coordination, hierarchical guidance, or gradient-based hybridization to enhance generalization and scalability in large-scale engineering tasks.

References

1. Shen Q, Wang Y, Yang Z, et al. Memory-efficient gradient unrolling for large-scale bi-level optimization[J]. Advances in Neural Information Processing Systems, 2024, 37: 90934-90964.
2. Balamurali S, Kathirvelu M, Palanisamy S K, et al. Redefining IoT networks for improving energy and memory efficiency through compressive sensing paradigm[J]. Scientific Reports, 2025, 15(1): 27180.

 doi:10.20944/preprints202603.1696.v1

3. Tiwari A. A hybrid feature selection method using an improved binary butterfly optimization algorithm and adaptive β–hill climbing[J]. IEEE Access, 2023, 11: 93511-93537.
4. Nadimi-Shahraki M H, Zamani H, Fatahi A, et al. MFO-SFR: An enhanced moth-flame optimization algorithm using an effective stagnation finding and replacing strategy[J]. Mathematics, 2023, 11(4): 862.
5. Hu,L. (2025). Topic Classification of Small Sample News Based on Prompt Engineering. Applied and Computational Engineering,170,101-107.
6. Fang W, Jiang H, Lu H, et al. GPU-based efficient parallel heuristic algorithm for high-utility itemset mining in large transaction datasets[J]. IEEE Transactions on Knowledge and Data Engineering, 2023, 36(2): 652-667.
7. Han M, Gao Z, Li A, et al. An overview of high utility itemsets mining methods based on intelligent optimization algorithms[J]. Knowledge and Information Systems, 2022, 64(11): 2945-2984.
8. Talapula D K, Ravulakollu K K, Kumar M, et al. SAR-BSO meta-heuristic hybridization for feature selection and classification using DBNover stream data[J]. Artificial Intelligence Review, 2023, 56(12): 14327-14365.
9. Liu X, Qi H, Jia S, et al. Recent advances in optimization methods for machine learning: a systematic review[J]. Mathematics, 2025, 13(13): 2210.
10. Wang Z, Chen H, Yang S, et al. A lightweight intrusion detection method for IoT based on deep learning and dynamic quantization[J]. PeerJ Computer Science, 2023, 9: e1569.