Genetic Programming-Based Evolutionary Strategy Generation for Complex Optimization

1st Jeng-Shyang Pan
School of Information Engineering, Yango University, Fuzhou 350015, China
College of Computer Science and Engineering, Shandong University of Science and Technology, Qingdao 266590, China
Department of Information Management, Chaoyang University of Technology, Taichung 41349, Taiwan, China
Email: jengshyangpan@gmail.com

2nd Wenda Li
College of Computer Science and Engineering,
Shandong University of Science and Technology, Qingdao 266590, China
Email: liwenda@sdust.edu.cn

3rd Shu-Chuan Chu
School of Artificial Intelligence, Nanjing University of Information Science and Technology, Nanjing 210044, China
Email: scchu0803@gmail.com

4th Zhi-Gang Du
School of Transportation and Logistics, Southwest Jiaotong University, Chengdu 611756, China
Email: 1461779873@qq.com

5th Hongmei Yang
College of Computer Science and Engineering, Shandong University of Science and Technology, Qingdao 266590, China
Email: Yhm1998@163.com

6th Lingping Kong
Faculty of Electrical Engineering and Computer Science, VSB-Technical University of Ostrava, Ostrava, Czech Republic
Email: lingping.kong@vsb.cz

***Abstract*—Designing effective update strategies is vital to the success of metaheuristic algorithms. Traditional methods often depend on manually designed rules and empirical tuning, which restrict their adaptability and scalability across different optimization problems. To overcome this limitation, a novel framework named GP-MAs is proposed, which integrates Genetic Programming (GP) into metaheuristics to automatically evolve and optimize their update equations. In this framework, GP dynamically constructs learning rules within the search process, allowing the algorithm to adapt to varying problem landscapes. The framework is implemented on the Growth Optimizer (GO), forming a hybrid variant called GPbasedGO. Experimental evaluations on the CEC2022 benchmark suite demonstrate that GPbasedGO achieves superior convergence speed, robustness, and generalization ability compared with several state-of-the-art algorithms. The proposed GP-MAs framework offers a flexible and automated paradigm for metaheuristic design, enabling the generation of adaptive update strategies suitable for complex optimization tasks and real-world applications.**

***Index Terms*—Metaheuristic optimization, Genetic Programming, Adaptive update strategy**

“

I. Introduction

With the rapid advancement of artificial intelligence (AI), applications in healthcare, transportation, and manufacturing have grown substantially. However, challenges such as technical bottlenecks, data heterogeneity, and scenario-specific constraints persist. Real-world problems often exhibit nonlinear, non-differentiable, and complex characteristics. Metaheuristic algorithms (MAs) have emerged as powerful tools for global optimization, providing flexible, structure-independent solutions [1], [2]. Unlike traditional optimization, MAs explore complex search spaces probabilistically, often simulating natural, biological, or physical processes, and exploit problem-specific knowledge to find high-quality solutions. Recent applications include wireless sensor networks (WSNs) [3], [4], cloud resource allocation [5], communication security [6], intelligent manufacturing [7], machine learning [8], and AI tasks [9], [10].

MAs are taxonomically classified based on inspiration mechanisms [11], [12], as summarized in Fig. 1.

Classification: 1) *Evolutionary-based*: Genetic Algorithm

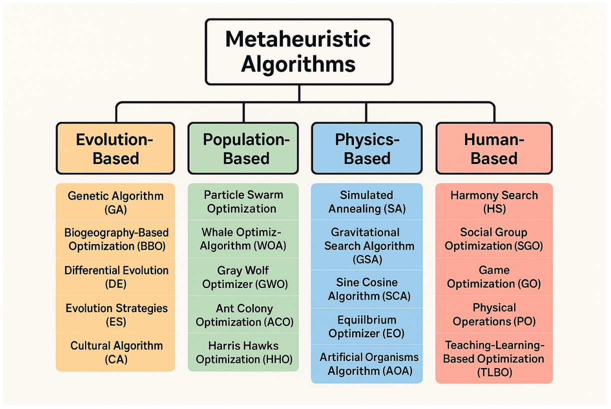

Metaheuristic Algorithms
Evolution-Based
Population-Based
Physics-Based
Human-Based
Genetic Algorithm (GA)
Biogeography-Based Optimization (BBO)
Differential Evolution (DE)
Evolution Strategies (ES)
Cultural Algorithm (CA)
Particle Swarm Optimization
Whale Optimiz-Algorithm (WOA)
Gray Wolf Optimizer (GWO)
Ant Colony Optimization (ACO)
Harris Hawks Optimization (HHO)
Simulated Annealing (SA)
Gravitational Search Algorithm (GSA)
Sine Cosine Algorithm (SCA)
Equilibrium Optimizer (EO)
Artificial Organisms Algorithm (AOA)
Harmony Search (HS)
Social Group Optimization (SGO)
Game Optimization (GO)
Physical Operations (PO)
Teaching-Learning-Based Optimization (TLBO)

Fig. 1: A brief classification of metaheuristic algorithms.

(GA) [13], Differential Evolution (DE) [14], Evolution Strategy (ES) [15], Biogeography-Based Optimization (BBO) [16], Immunization Algorithm (IA) [17]; 2) *Swarm intelligence*: PSO [18], ACO [19], ABC [20], GWO [21], WOA [22], MGO [23]; 3) *Physics-based*: Simulated Annealing (SA) [24], Gravitational Search Algorithm (GSA) [25], Sine Cosine Algorithm (SCA) [26], Multi-Verse Optimization (MVO) [27]; 4) *Human behavior-inspired*: TLBO [28], Harmony Search (HS) [29], Social Group Optimization (SGO) [30], Special Forces Algorithm (SFA) [31].

Research in MAs follows three main directions: (i) proposing new algorithms, e.g., Gannet Optimization Algorithm (GOA) [32] and Status-based Optimization (SBO) [33]; (ii) hybridization of algorithms, e.g., IMPAEO [34] and HADEFP [35]; (iii) enhancing performance through new strategies or parameter tuning [36].

Despite their diversity, MAs rely heavily on manually designed update rules, often requiring empirical tuning. Limitations include:

- **Empirical dependence**: strategies are problem-specific, leading to performance fluctuations;
- **Lack of adaptability**: static rules cannot dynamically adjust to changing landscapes;
- **High resource consumption**: redesigning or hybridizing formulas requires substantial time and computation.

To overcome these issues, Genetic Programming (GP) provides a hyper-heuristic approach to automatically generate update strategies [37], [38]. GP evolves formulas from basic components to match problem characteristics, reducing manual effort and enhancing generalization.

This study proposes the GP-MAs framework, using GP to automate MA update rules. Candidate formulas are encoded via syntax trees and evaluated with respect to an objective function based on performance metrics. The framework enables offline evolution of high-performance update rules, minimizing human intervention and accelerating algorithm design.

The Growth Optimizer (GO) algorithm [39], inspired by human learning and reflection, is adopted as a test case. Integrating GP-MAs into GO's learning phase yields GPbasedGO, which dynamically discovers strategies and balances exploration and convergence. Benchmark evaluations on the CEC2022 suite verify its improved accuracy and robustness.

The remainder of the paper is structured as follows: Section II reviews GO and GP methodologies; Section III introduces the GP-MAs framework and GPbasedGO; Section IV presents experimental results; Section V concludes and discusses future directions.

II. Related Work

This chapter systematically reviews the theoretical foundations and techniques closely related to this research, mainly including Growth Optimizer (GO), Genetic Programming (GP), and key techniques for multispectral and panchromatic image fusion. These provide theoretical support for the subsequent methodology and experiments.

A. Growth Optimizer

The GO algorithm is an emerging intelligent optimization algorithm simulating the human "learning-reflection-self-improvement" process. It features strong global search capability and adaptability. This section introduces its basic structure, including initialization, learning mechanism, and reflection phase.

1) Initialization: The initial population consists of 0^{th} generation individuals. For the i-th individual $X_i(0)$, the j-th component $x_{i,j}(0)$ lies within $[x_{i,j}^{\min}, x_{i,j}^{\max}]$, where $x_{i,j}^{\min}$ and $x_{i,j}^{\max}$ are the lower and upper bounds. Population size is N_p and dimension number is D. Initialization uses uniform distribution:

$$x_{i,j}(0) = x_{i,j}^{\min} + \text{rand}(0,1) \cdot (x_{i,j}^{\max} - x_{i,j}^{\min}) \tag{1}$$

2) Learning Phase: Individuals learn from gaps between themselves and others. Four gap types are defined:

$$\begin{cases} \text{Gap}_1 = \vec{x}_{\text{best}} - \vec{x}_{\text{better}} \\ \text{Gap}_2 = \vec{x}_{\text{best}} - \vec{x}_{\text{worse}} \\ \text{Gap}_3 = \vec{x}_{\text{better}} - \vec{x}_{\text{worse}} \\ \text{Gap}_4 = \vec{x}_{\text{L1}} - \vec{x}_{\text{L2}} \end{cases} \tag{2}$$

The learning factor (LF_k) measures the influence of each gap:

$$LF_k = \frac{\|\text{Gap}_k\|}{\sum_{k=1}^{4} \|\text{Gap}_k\|}, \quad k = 1,2,3,4 \tag{3}$$

Self-perception factor (SF) evaluates an individual's state:

$$SF = \frac{GR}{GR_{\max}} \tag{4}$$

Knowledge acquisition from each gap:

$$\vec{KA}_k = SF \cdot LF_k \cdot \vec{Gap}_k, \quad k = 1,2,3,4 \tag{5}$$

Position update:

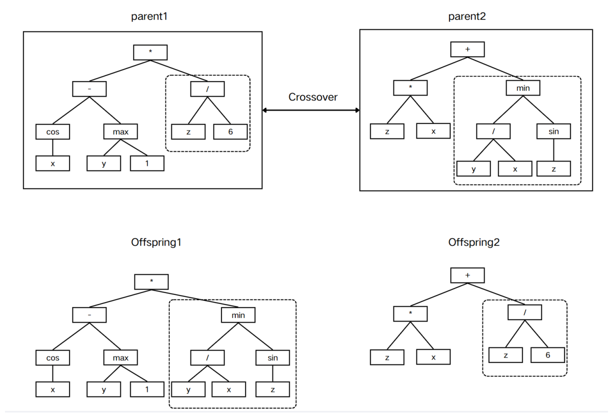


Fig. 2: Sample diagram of crossover operation.

$$\vec{x}_i^{\text{It}+1} = \vec{x}_i^{\text{It}} + \sum_{k=1}^{4} \vec{KA}_k \tag{6}$$

Update verification ensures genuine improvement:

$$\vec{x}_i^{\text{it}+1} = \begin{cases} \vec{x}_i^{\text{it}+1} & \text{if } f(\vec{x}_i^{\text{it}+1}) < f(\vec{x}_i^{\text{it}}) \\ \begin{cases} \vec{x}_i^{\text{it}+1} & r_1 < P_2 \\ \vec{x}_i^{\text{it}} & \text{else} \end{cases} & \text{otherwise} \end{cases} \tag{7}$$

3) Reflection Phase: Reflection updates each dimension as:

$$\vec{x}_{i,j}^{\text{it}+1} = \begin{cases} lb + r_4 \cdot (ub - lb) & r_3 < AF \\ \vec{x}_{i,j}^{\text{it}} + r_5 \cdot (R_j - \vec{x}_{i,j}^{\text{it}}) & r_2 < P_3 \\ \vec{x}_{i,j}^{\text{it}} & \text{otherwise} \end{cases} \tag{8}$$

$$AF = 0.01 + 0.99 \cdot \left(1 - \frac{FEs}{MaxFEs}\right) \tag{9}$$

B. Genetic Programming

Genetic programming (GP) automatically generates program structures via evolution. Individuals are represented as trees with function nodes $F = \{f_1, ..., f_n\}$ and terminal nodes $T = \{t_1, ..., t_m\}$. Initialization methods include:

Full Method: Functions fill all non-leaf nodes; terminals occupy leaves. **Grow Method:** Functions or terminals are randomly assigned to nodes. **Ramped Half-and-Half:** Combines both methods to ensure structural diversity.

Basic GP operations include:

(1) **Selection:** Chooses individuals for reproduction. Strategies include Roulette Wheel, Tournament, and Rank selection.

(2) **Crossover:** Swaps randomly selected subtrees between two parents to produce offspring (Fig. 2).

(3) **Mutation:** Introduces structural changes. Subtree mutation replaces a subtree with a random one; point mutation randomly alters a node (Fig. 3).

III. GP-MAs Framework and GO Algorithm Based on the GP-MAs Framework

This chapter introduces the GP-MAs framework and its integration into the Growth Optimizer (GO). It outlines the framework's architecture, parameter design, and the specific update mechanisms developed for GO.

A. Overall Structure

Fig. 4 shows the architecture of the GP-MAs-GO framework, with red lines indicating data flow. The original GO serves as the base algorithm, while the GP-MAs module is embedded only in the learning phase to dynamically evolve learning strategies. The reflection phase and overall GO structure remain unchanged.

Step 1: Define the function set, terminal set, and objective function for evaluating evolved learning formulas, then initialize a GP population encoding candidate formulas. Through selection, crossover, and mutation, superior strategies are retained and inferior ones discarded.

Each GP individual is evaluated by integrating its formula into GO's learning phase, replacing the original update rule. The modified GO is run on benchmark functions, and the resulting optimization performance serves as the individual's fitness.

Step 2: Replace GO's original manual formula with the best-evolved formula from GP-MAs, yielding GPbasedGO. Unlike conventional GO, GPbasedGO dynamically adapts its learning strategy to different problem landscapes and evolutionary stages. Since GP-MAs evolution is offline, the best formula can be reused across tasks, reducing manual design effort and improving adaptability.

B. The GP-MAs Framework

The GP-MAs (Genetic Programming-based Metaheuristic Algorithms) framework uses genetic programming to automatically construct update strategies for metaheuristic algorithms. By encoding search behaviors as evolvable mathematical expressions, the framework iteratively refines these strategies to improve optimization performance. Table I summarizes the GP-MAs parameter settings.

TABLE I: Parameter settings of the GP-EAs framework

Parameter	**Value**
Population size (PS)	100
Initialization method	Half depth-first, half breadth-first
Maximum iterations (MAXIT)	200
Tree depth (MAXD / MIND)	7 / 9
Elitism rate	20%
Crossover / Mutation probability	0.8 / 0.2

The detailed steps of the GP-MAs framework are as follows:

1) **Initialization:** A population of individuals is randomly initialized using a predefined set of functions and terminals. To enhance structural diversity, half of the individuals are generated using a depth-first growth method, while the remaining half use breadth-first construction.

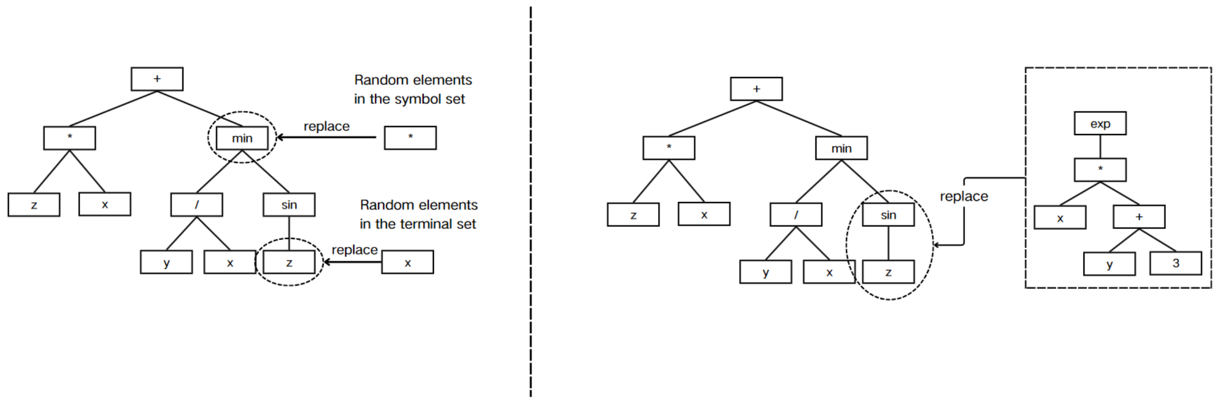

+
*
min
z
x
/
sin
y
x
z
Random elements in the symbol set
replace
*
Random elements in the terminal set
replace
x
+
*
min
z
x
/
sin
y
x
z
replace
exp
*
x
+
y
3

Fig. 3: Sample diagram of mutation operation.

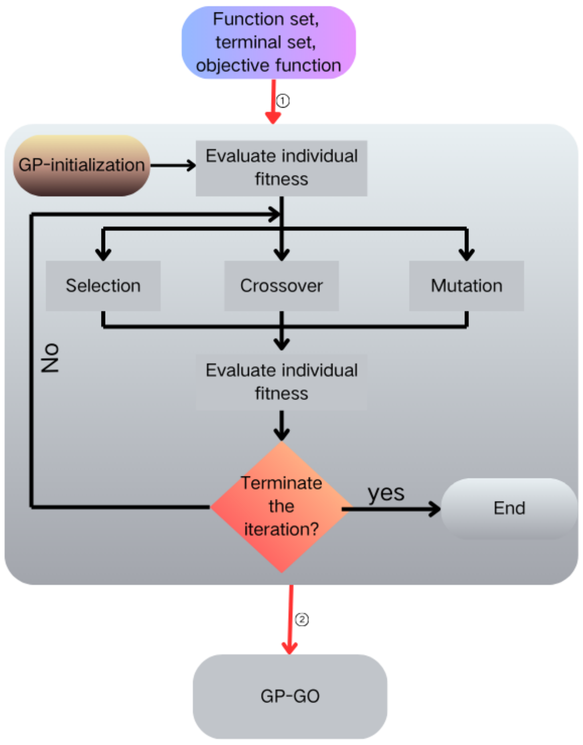

Function set, terminal set, objective function
①
GP-initialization
Evaluate individual fitness
Selection
Crossover
Mutation
No
Evaluate individual fitness
Terminate the iteration?
yes
End
②
GP-GO

Fig. 4: System framework diagram.

2) **Expression Construction and Evaluation:** Each individual encodes a candidate metaheuristic component (e.g., an update rule or control policy) as a tree structure. These individuals are converted into executable expressions via in-order traversal and then embedded into a target metaheuristic algorithm, where their performance is evaluated using a predefined objective function.
3) **Elitism:** The top-performing individuals, based on fitness scores, are preserved in the next generation to ensure the retention of high-quality solutions.
4) **Variation Operators:** The remaining individuals undergo variation through crossover and mutation, with parent selection guided by a roulette-wheel selection mechanism. Structural constraints are applied to maintain the syntactic and semantic validity of the expressions.
5) **Fitness Evaluation:** The fitness of the new individuals is recalculated by integrating them into the algorithm and re-evaluating their performance.
6) **Termination:** Steps 2 to 5 are repeated until the stopping criterion is met, such as reaching the maximum number of generations or achieving a satisfactory performance level.

Algorithm 1 outlines the workflow of the GP-MAs framework. Inspired by prior studies on evolutionary formula generation in metaheuristics, this approach ensures structural diversity at initialization via a balanced tree construction strategy. Each individual, represented as a candidate metaheuristic rule, is transformed into an executable formula and evaluated within a host algorithm. Top individuals are retained via elitism, while the remaining ones undergo crossover and mutation under structural constraints. The evolution proceeds iteratively until convergence or reaching the iteration limit.

To quantitatively assess the performance of evolved formulas, a customized evaluation function F is designed for the GP-MAs framework. Considering the heterogeneity of real-world optimization problems, four representative test functions are selected from the CEC2022 benchmark suite: F1 (unimodal), F3 (basic multimodal), F7 (hybrid), and F10 (composition). Each evolved formula is integrated into a metaheuristic algorithm variant, referred to as *new-MAs*, and tested on these functions.

Let fit_j be the mean result of *new-MAs* on function j over five independent runs, and fit_j^* denote the known global optimum. The overall fitness score is computed as:

$$\text{Fitness} = \sum_{j\in\{1,3,7,10\}} \frac{\text{fit}_j}{\text{fit}_j^*} \tag{10}$$

This aggregated metric reflects the generalization capacity of evolved strategies across a diverse problem set, thus pro-

Algorithm 1 GP-MAs: Genetic Programming-based Metaheuristic Automation

```
1: Input: $PS$, $MAXD$, $MAXIT$, Evaluation function $F$, Function set, Terminal set
2: Output: Optimal evolved update rule
3: Initialization:
4:   -population $POP^0$ of size $PS$:
5:     - 50% via depth-first tree generation
6:     - 50% via breadth-first tree generation
7:     - Nodes drawn from function and terminal sets
8: Parse individuals (in-order) $\rightarrow$ executable expressions
9: Evaluate fitness of each individual using $F$
10: $iter \leftarrow 1$
11: while $iter \leq MAXIT$ do
12:     Preserve top 20% of $POP^{iter}$ into $POP^{iter+1}$ (elitism)
13:     Select parents via hybrid (roulette + tournament) selection
14:     crossover: subtree swapping
15:     mutation:
16:         - Node mutation
17:         - Subtree mutation
18:     evaluate offspring using $F$
19:     Add valid offspring to $POP^{iter+1}$ until $|POP| = PS$
20:     $iter \leftarrow iter + 1$
21: end while
22: Return: Best-performing individual in final generation
```

viding an indirect yet robust assessment of their effectiveness and adaptability.

C. The GP-GO Algorithm

This paper proposes **GPbasedGO**, an enhanced variant of the Growth Optimizer (GO) algorithm optimized through the GP-MAs framework. Within this framework, GP-MAs autonomously evolve the learning rule. The evaluation function F is configured with a dimension of 10. Consistent with established methodologies, four representative functions (F1, F3, F7, F10) from the CEC2022 benchmark are selected for evaluation.lm

The terminal set $\mathcal{T}$ and function set $\mathcal{F}$ critically determine GP performance. For GO optimization, these sets are designed as:

$$\begin{cases} \mathcal{T} = \{Gap_1, Gap_2, Gap_3, Gap_4, SF\} \\ \mathcal{F} = \{+, -, \times, \div, \text{norm}, \sin, \cos, \tan\} \end{cases}$$

$\mathcal{T}$ encodes social hierarchy dynamics: Gap_{1-4} quantify rank disparities (e.g., leader-elite, elite-lower, random pairs) to guide knowledge acquisition through hierarchical learning mechanisms. SF (self-perception factor) dynamically scales learning intensity via growth resistance (GR), balancing local refinement ($SF \ll 1$) and global exploration ($SF \gg 1$). This adaptive scaling mirrors parameter self-adaptation strategies in differential evolution.

$\mathcal{F}$ integrates arithmetic operators for evolutionary recombination and $norm$ for numerical stability. Trigonometric functions inject nonlinear perturbations to escape local optima while maintaining diversity. This hybrid design balances deterministic hierarchical learning with stochastic exploration, accelerating cnvergence while preserving solution diversity. Embedded self-adaptive components reduce parameter sensitivity while maintaining evolutionary pressure toward optimal regions, aligning with the ”learning-reflection-self-improvement” philosophy of human-inspired optimization.

The update rule for this process can be expressed as follows:

$$\begin{aligned} \text{Term}_1 &= \left\|\text{Gap}_1\right\| \cdot \text{Gap}_1 + \left\|\text{Gap}_2\right\| \cdot \text{Gap}_2, \\ \text{Term}_2 &= \left\|\text{Gap}_3\right\| \cdot \text{Gap}_3 \\ &\quad + \left\| (\text{Gap}_1 - \text{Gap}_1) - (SF \cdot SF) \right\| \cdot \text{Gap}_4 \end{aligned}$$

The numerator of the update formula can then be written as:

$$\text{Numerator} = SF \cdot (\text{Term}_1 + \text{Term}_2)\,. \tag{11}$$

The denominator is given by:

$$\begin{aligned} \text{Denominator}_1 &= \left\| \cos\left(\tan\left(\cos(\|\text{Gap}_1\|) \right) \right) \right\| + \left\|\text{Gap}_2\right\|, \\ \text{Denominator}_2 &= \left\|\text{Gap}_3\right\| + \left\| \sin\left(\| \cos(\text{Gap}_3 - \text{Gap}_2) \| \right) \right\|, \end{aligned} \tag{12}$$

Thus, the complete update equation becomes:

$$\text{Update} = \frac{\text{Numerator}}{\text{Denominator}_1 + \text{Denominator}_2}. \tag{13}$$

Compared with the original formulation, the new equation is evolved automatically through Genetic Programming (GP), thereby exhibiting notable advantages in terms of structural complexity, adaptability, and information integration. First, the new equation incorporates multiple nonlinear operators (e.g., trigonometric functions, nested norms, and difference operations), which substantially enhance its representational capacity to capture intricate relationships among different gaps. Second, the GP-evolved structure autonomously determines the selection and combination of operators, enabling dynamic adaptability to various optimization tasks without manual redesign. Moreover, the formulation integrates interaction information among gaps, not only exploiting individual gap measures but also leveraging higher-order features such as the differences between gaps, thereby enriching the learning information sources. Finally, the inclusion of nonlinear transformations increases the diversity of search directions and step sizes, which helps to avoid premature convergence and better balance exploration and exploitation. Overall, the new equation demonstrates superior convergence performance, enhanced global search ability, and reduced human design bias compared to the original formulation, offering a more flexible and efficient learning mechanism for complex optimization problems.

IV. Experimental Results

This section details the experimental framework for evaluating the proposed GPbasedGO algorithm. Representative benchmark functions from the CEC2022 test suite are employed to assess optimization performance in complex scenarios.

A. Experimental Setup

Ten comparative algorithms are evaluated, including PSO, GA, ASO, BOA, SA, CSO, GWO, DE, CRO, and GO. To ensure a fair comparison, all algorithms are executed with identical population sizes and fixed fitness evaluations (FES). Each algorithm is independently run 30 times on every benchmark function to reduce the impact of initialization randomness on solution accuracy. The performance is assessed in terms of the mean and standard deviation of the solutions, as well as Wilson?s rank sum test.

To evaluate the performance of the proposed GPbasedGO algorithm, the standard CEC2022 benchmark suite was utilized. This test set comprises four categories of functions: unimodal, basic multimodal, hybrid, and composition functions. The unimodal function (F1) is specifically designed to assess the algorithm?s local search capability and convergence accuracy, as it contains only a single global optimum.

The basic multimodal functions (F2–F5) present a large number of local optima, which are intended to evaluate the algorithm?s global exploration ability and its competence in avoiding premature convergence to local minima. These functions pose a significant challenge by requiring an effective balance between exploration and exploitation.

The hybrid functions (F6–F8) are constructed by combining multiple basic functions with diverse characteristics, resulting in more intricate and deceptive landscapes. These functions are used to examine the algorithm?s robustness and adaptability in dynamic and heterogeneous optimization environments.

Finally, the composition functions (F9–F12) represent the most complex category. They are formulated by blending several hybrid or basic functions with various biases, rotations, and scaling transformations. These functions introduce significant levels of nonlinearity, modality, and search space irregularity, thereby providing a rigorous testbed for assessing the algorithm?s ability to locate and transition between multiple distinct regions in the search space.

Through comprehensive experimentation across this diverse set of benchmark functions, the effectiveness and generalization capability of the GPbasedGO algorithm can be thoroughly evaluated in optimization problems of varying difficulty and landscape complexity.

B. Analysis of experimental results

1) Analysis of 10-dimensional results: Tables II and III present comparative results on 10-dimensional problems between the GPbasedGO algorithm and other optimization algorithms. The comparison of mean values reveals the superior overall performance of GPbasedGO. The data indicate that GPbasedGO generally achieves better results, with its mean metric showing statistically significant improvements over other algorithms on specific test functions such as F1, F4, F6, and F11. However, on certain functions such as F3 and F7, other algorithms may perform comparably to or slightly better than GPbasedGO.

When comparing GPbasedGO with various algorithms (e.g., PSO, GA, DE), its unique advantages become evident. As shown in the tables, GPbasedGO achieves superior results in several test cases. Compared to PSO, GPbasedGO exhibits smaller mean values across all functions, demonstrating enhanced performance in optimizing high-dimensional complex functions. When compared to GA, the standard deviation of GPbasedGO is significantly smaller, indicating greater stability and reduced sensitivity to initial conditions and random factors. Against DE and similar algorithms, GPbasedGO shows stronger search capabilities on multimodal functions. Furthermore, its advantages over algorithms like GWO and CSO primarily lie in balancing unimodal and multimodal optimization, highlighting its robust problem-solving ability for complex tasks.

Fig. 5 displays the convergence curves of various algorithms in 10-dimensional space. The figure illustrates representative segments of convergence behavior, with subplot titles indicating the corresponding benchmark functions. It is evident that GPbasedGO exhibits strong optimization capabilities on most functions, ultimately outperforming nearly all other algorithms. In the first 500 evaluations, all algorithms converge rapidly except on functions like F12 and F18, where algorithms such as CSO and CRO stagnate. GPbasedGO, however, effectively balances local exploitation and global exploration?a critical feature for optimizing complex functions. This equilibrium prevents premature convergence to local optima while efficiently exploring the entire search space.

2) Analysis of 20-dimensional results: Tables IV and V present the 20-dimensional comparison results of the GPbasedGO algorithm against other algorithms. The average values in the tables indicate the overall superiority of GPbasedGO. Specifically, GPbasedGO demonstrates significantly improved average value metrics compared to other algorithms on certain test functions like F1, F2, F8, and F12. However, on some functions such as F10, other algorithms may perform comparably or slightly better than GPbasedGO.

When compared to other algorithms, including PSO, GA, DE, GWO, and GO, GPbasedGO exhibits unique advantages. Among the listed algorithms, GPbasedGO achieves better results in several test cases. Compared to PSO, GPbasedGO has lower average values on most functions, and in terms of convergence speed, GPbasedGO also shows faster convergence in the initial stage. In high-dimensional complex function optimization, GPbasedGO shows better average values. Relative to GA, GPbasedGO has a smaller standard deviation, indicating more stable results with less influence from initial conditions and random factors. Compared to DE, GPbasedGO exhibits stronger search abilities for multimodal functions. Moreover, GPbasedGO outperforms other algorithms like GWO and GO, particularly in the combination of unimodal and multimodal functions, demonstrating its strong ability to solve complex problems.

Fig. 6 illustrates the convergence curves of various algo-

TABLE II: 10-Dimensional comparison results(Part 1).

Func_Num	Mean_PSO	R	Mean_GA	R	Mean_DE	R	Mean_GWO	R	Mean_GO	R	Mean_GPbasedGO
1	8828.213	+	7178.107	+	4653.567	+	5883.246	+	2070.409	+	761.238
2	481.4768	+	891.4384	+	421.6076	+	443.5214	+	408.641	+	403.9103
3	622.0307	+	641.7936	+	608.4225	+	602.7519	+	601.6679	=	601.4074
4	856.9073	+	841.992	+	850.7588	+	822.5716	=	838.2599	+	821.5802
5	1274.636	+	1237.221	+	938.7871	+	946.3514	=	926.8552	+	914.3811
6	4344257	+	56128305	+	17187.74	+	21769.89	+	28638.76	+	4487.426
7	2070.12	+	2078.786	+	2052.423	+	2049.472	+	2035.152	+	2023.685
8	2237.394	+	2248.042	+	2231.638	+	2263.761	+	2228.674	+	2224.31
9	2587.122	+	2704.782	+	2506.232	=	2593.41	+	2529.35	+	2501.057
10	2624.657	+	2547.583	=	2546.931	=	2591.171	=	2547.91	=	2557.738
11	2995.133	+	3080.935	+	2828.305	+	2828.417	=	2689.166	=	2749.009
12	2875.162	+	3001.373	+	2862.248	+	2869.924	+	2863.979	+	2860.399
Win		12		11		11		8		9	

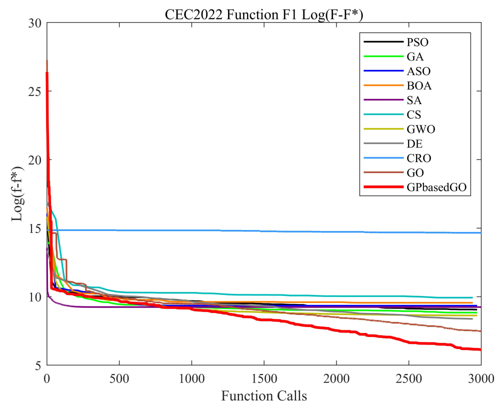
TABLE III: 10-Dimensional comparison results(Part 2).

Func_Num	Mean_ASO	R	Mean_BOA	R	Mean_SA	R	Mean_CS	R	Mean_CRO	R	Mean_GPbasedGO
1	11718.16	+	14424.11	+	10635.12	+	20780.83	+	2331248	+	761.238
2	489.1466	+	2221.291	+	526.9067	+	1651.528	+	4592.578	+	403.9103
3	620.5264	+	668.0617	+	649.047	+	668.4357	+	699.9431	+	601.4074
4	838.7514	+	875.0021	+	874.2788	+	890.413	+	928.6312	+	821.5802
5	1041.59	+	1787.522	+	1528.244	+	2680.351	+	4934.607	+	914.3811
6	1004955	+	2.58E+08	+	4323.045	=	1.56E+08	+	1.62E+09	+	4487.426
7	2087.336	+	2123.092	+	2106.302	+	2144.093	+	2274.834	+	2023.685
8	2240.496	+	2317.531	+	2249.618	+	2313.266	+	3270.592	+	2224.31
9	2652.427	+	2823.84	+	2724.674	+	2819.166	+	3176.525	+	2501.057
10	2517.251	=	2859.451	+	2756.895	+	2738.914	+	4144.509	+	2557.738
11	2796.109	+	3811.853	+	3111.818	+	3743.182	+	5064.099	+	2749.009
12	2927.294	+	3085.632	+	2963.107	+	2977.527	+	3238.392	+	2860.399
Win		11		12		11		12		12	

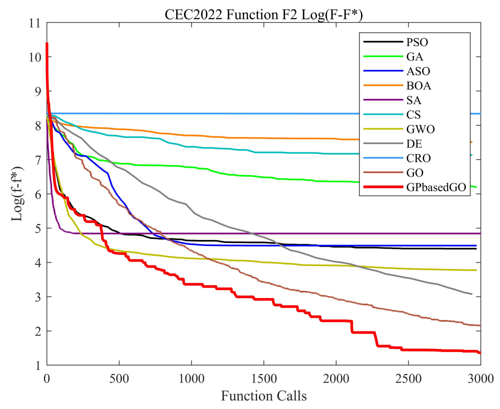
rithms in 10-dimensional space. We show only representative parts of the convergence graphs here, with the subplot titles indicating the corresponding benchmark functions. It is evident that GPbasedGO shows strong optimization ability on most functions and outperforms almost all other algorithms in the final results. During the first 500 evaluations, all algorithms converge rapidly, except for some, like CS and CRO, for certain functions. GPbasedGO alternates between two phases with a growth cycle during iteration, allowing it to balance local and global exploration effectively. This helps prevent premature convergence to local optima and enables better search space exploration. In the seedling growth phase, the population update method slows down and stabilizes the search space exploration process. This approach maintains a certain degree of exploration of new regions while focusing more on the in-depth development of the surrounding environment.

V. Conclusion

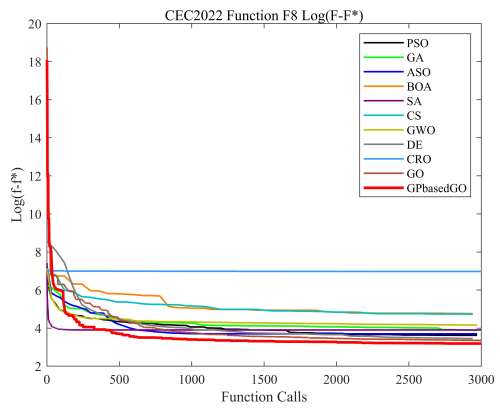
This paper introduces a novel computational framework, GP-MAs, which harnesses the power of Genetic Programming (GP) to autonomously design and optimize update strategies within metaheuristic algorithms. By decoupling the update rule construction from manual design and embedding it within an evolutionary paradigm, the proposed approach addresses key limitations of traditional metaheuristics—namely, empirical dependency, lack of adaptivity, and poor generalizability. The GP-MAs framework not only reduces human intervention and design overhead but also enhances algorithmic robustness by evolving dynamic and problem-specific update mechanisms. To validate the effectiveness of this framework, the Growth Optimizer (GO) algorithm was selected as a case study and enhanced through the integration of GP-driven strategy generation, resulting in the GPbasedGO algorithm. Extensive experiments were conducted on the CEC2022 benchmark suite, where GPbasedGO demonstrated superior performance in terms of convergence accuracy and robustness, particularly in complex and multimodal landscapes. Overall, the proposed GP-MAs framework provides a promising pathway for achieving automated, adaptive, and generalizable optimization in metaheuristic algorithm design.



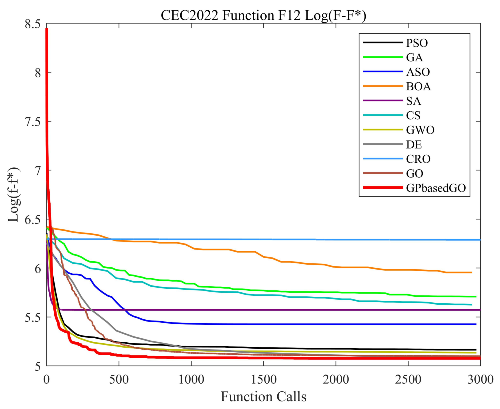
(**a**) F1



(**b**) F2



(**c**) F8



(**d**) F12

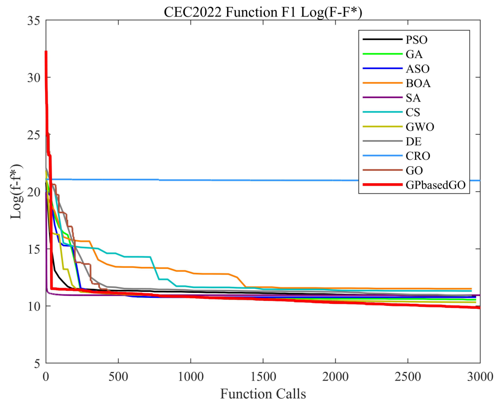
Fig. 5: 10-dimensional convergence plots of the algorithm.

TABLE IV: 20-Dimensional comparison results(Part 1).

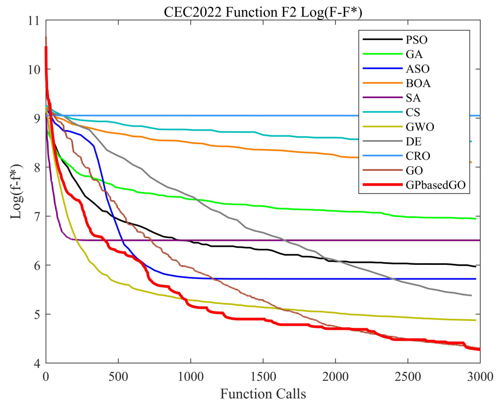
Func_Num	Mean_PSO	R	Mean_GA	R	Mean_DE	R	Mean_GWO	R	Mean_GO	R	Mean_GPbasedGO
1	52904.89	+	38427.22	+	54598.73	+	30783.65	+	19378.57	=	18987.99
2	791.1697	+	1435.08	+	616.9638	+	531.2367	+	473.6368	=	471.8706
3	636.9382	+	675.4038	+	630.5062	+	614.387	=	609.4576	=	611.4177
4	968.9801	+	949.5482	+	957.3419	+	864.6567	=	932.2348	+	868.1742
5	3389.725	+	3130.209	+	2040.769	+	1514.343	=	1447.686	=	1371.278
6	3.16E+08	+	7.93E+08	+	25570941	+	3841266	+	2683454	+	146310.6
7	2196.214	+	2180.967	+	2225.742	+	2118.254	=	2134.449	+	2095.174
8	2334.213	+	2429.743	+	2317.251	+	2275.405	+	2251.454	+	2244.28
9	2620.435	+	3059.033	+	2519.274	+	2561.716	+	2484.558	+	2474.834
10	5862.466	+	3977.858	=	5745.673	+	4394.218	+	3732.7	=	3577.976
11	4637.618	+	7291.082	+	4381.296	+	4010.602	+	3239.342	=	3267.323
12	3097.133	+	3753.875	+	2900.005	-	3027.249	+	2961.031	+	2915.146
Win		12		11		11		8		6	

TABLE V: 20-Dimensional comparison results(Part 2).

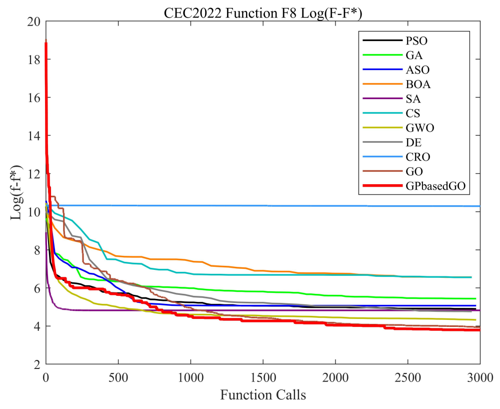
Func_Num	Mean_ASO	R	Mean_BOA	R	Mean_SA	R	Mean_CS	R	Mean_CRO	R	Mean_GPbasedGO
1	47506.21	+	99751.94	+	56751.55	+	82307.3	+	1.28E+09	+	18987.99
2	704.7262	+	3692.529	+	1069.681	+	5435.199	+	8923.236	+	471.8706
3	644.7519	+	697.827	+	692.8078	+	708.7662	+	738.0695	+	611.4177
4	927.8916	+	1020.596	+	1025.24	+	1064.253	+	1108.797	+	868.1742
5	2416.787	+	5299.146	+	3482.977	+	9737.955	+	14244.68	+	1371.278
6	11609227	+	2.75E+09	+	7888.752	-	3.44E+09	+	8.79E+09	+	146310.6
7	2242.773	+	2301.811	+	2227.635	+	2320.135	+	2535.034	+	2095.174
8	2359.021	+	2909.087	+	2324.389	+	2903.279	+	31644.07	+	2244.28
9	2688.301	+	3675.949	+	2892.854	+	3409.789	+	4881.742	+	2474.834
10	4580.649	=	6886.978	+	4959.543	+	6572.561	+	8394.181	+	3577.976
11	4633.79	+	9374.592	+	7549.17	+	11243.98	+	15367.25	+	3267.323
12	3479.162	+	4029.731	+	3527.145	+	3676.27	+	4493.905	+	2915.146
Win		11		12		11		12		12	



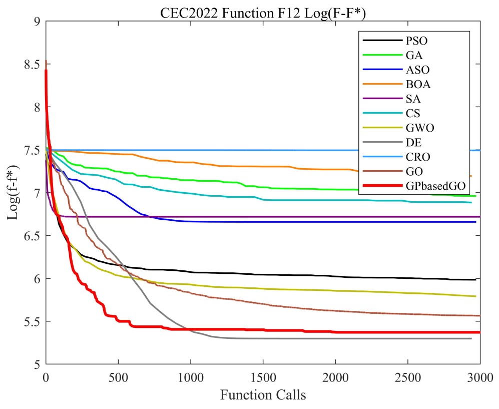
(**a**) F1



(**b**) F2



(**c**) F8



(**d**) F12

Fig. 6: 20-dimensional convergence plots of the algorithm.

REFERENCES

[1] T. Dokeroglu, E. Sevinc, T. Kucukyilmaz, and A. Cosar, "A survey on new generation metaheuristic algorithms," *Computers & Industrial Engineering*, vol. 137, p. 106040, 2019.

[2] K. Rajwar, K. Deep, and S. Das, "An exhaustive review of the metaheuristic algorithms for search and optimization: taxonomy, applications, and open challenges," *Artificial Intelligence Review*, vol. 56, no. 11, pp. 13 187–13 257, 2023.

[3] T.-T. Nguyen, J.-S. Pan, and T.-K. Dao, "An improved flower pollination algorithm for optimizing layouts of nodes in wireless sensor network," *IEEE Access*, vol. 7, pp. 75 985–75 998, 2019.

[4] Z.-G. Du, J.-S. Pan, S.-C. Chu, H.-J. Luo, and P. Hu, "Quasi-affine transformation evolutionary algorithm with communication schemes for application of rssi in wireless sensor networks," *IEEE Access*, vol. 8, pp. 8583–8594, 2020.

[5] G. Zhou, Y. Xie, H. Lan, W. Tian, R. Buyya, and K. Wu, "Information interaction and partial growth-based multi-population growable genetic algorithm for multi-dimensional resources utilization optimization of cloud computing," *Swarm and Evolutionary Computation*, vol. 87, p. 101575, 2024.

[6] A. K. Abasi, M. Aloqaily, M. Guizani, and B. Ouni, "Metaheuristic algorithms for 6g wireless communications: Recent advances and applications," *Ad Hoc Networks*, p. 103474, 2024.

[7] L. Wang, Z. Pan, and J. Wang, "A review of reinforcement learning based intelligent optimization for manufacturing scheduling," *Complex System Modeling and Simulation*, vol. 1, no. 4, pp. 257–270, 2021.

[8] E.-G. Talbi, "Machine learning into metaheuristics: A survey and taxonomy," *ACM Computing Surveys (CSUR)*, vol. 54, no. 6, pp. 1–32, 2021.

[9] X. Wang and Y. Wu, "An improved deep spiking neural network model based on bionic optimization algorithm," 2024.

[10] H. A. Elsayed, M. F. Hameed, and M. M. El Sherbiny, "Image classification using decision tree classifier and features extraction using hough transform and genetic algorithm," 2025.

[11] G. Hu, Y. Guo, J. Zhong, and G. Wei, "Iydse: Ameliorated young's double-slit experiment optimizer for applied mechanics and engineering," *Computer Methods in Applied Mechanics and Engineering*, vol. 412, p. 116062, 2023.

[12] Z. Liang and Z. Wang, "An enhanced kepler optimization algorithm with global attraction model and dynamic neighborhood search for global optimization and engineering problems," *Mathematics and Computers in Simulation*, vol. 237, pp. 107–144, 2025.

[13] J. A. López-Campos, A. Segade, E. Casarejos, J. R. Fernández, and G. R. Días, "Hyperelastic characterization oriented to finite element applications using genetic algorithms," *Advances in Engineering Software*, vol. 133, pp. 52–59, 2019.

[14] A. K. Qin, V. L. Huang, and P. N. Suganthan, "Differential evolution algorithm with strategy adaptation for global numerical optimization," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 2, pp. 398–417, 2009.

[15] H.-G. Beyer and H.-P. Schwefel, "Evolution strategies–a comprehensive introduction," *Natural computing*, vol. 1, pp. 3–52, 2002.

[16] D. Simon, "Biogeography-based optimization," *IEEE transactions on evolutionary computation*, vol. 12, no. 6, pp. 702–713, 2008.

[17] B. Shams, "Using network properties to evaluate targeted immunization algorithms," *Network Biology*, vol. 4, no. 3, p. 74, 2014.

[18] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95-international conference on neural networks*, vol. 4. ieee, 1995, pp. 1942–1948.

[19] C. Blum, "Ant colony optimization: Introduction and recent trends," *Physics of Life reviews*, vol. 2, no. 4, pp. 353–373, 2005.

[20] D. Karaboga and B. Akay, "A comparative study of artificial bee colony algorithm," *Applied mathematics and computation*, vol. 214, no. 1, pp. 108–132, 2009.

[21] S. Mirjalili, S. M. Mirjalili, and A. Lewis, "Grey wolf optimizer," *Advances in engineering software*, vol. 69, pp. 46–61, 2014.

[22] S. Mirjalili and A. Lewis, "The whale optimization algorithm," *Advances in engineering software*, vol. 95, pp. 51–67, 2016.

[23] B. Abdollahzadeh, F. S. Gharehchopogh, N. Khodadadi, and S. Mirjalili, "Mountain gazelle optimizer: a new nature-inspired metaheuristic algorithm for global optimization problems," *Advances in Engineering Software*, vol. 174, p. 103282, 2022.

[24] D. Bertsimas and J. Tsitsiklis, "Simulated annealing," *Statistical science*, vol. 8, no. 1, pp. 10–15, 1993.

[25] E. Rashedi, H. Nezamabadi-Pour, and S. Saryazdi, "Gsa: a gravitational search algorithm," *Information sciences*, vol. 179, no. 13, pp. 2232–2248, 2009.

[26] S. Mirjalili, "Sca: a sine cosine algorithm for solving optimization problems," *Knowledge-based systems*, vol. 96, pp. 120–133, 2016.

[27] S. Mirjalili, S. M. Mirjalili, and A. Hatamlou, "Multi-verse optimizer: a nature-inspired algorithm for global optimization," *Neural Computing and Applications*, vol. 27, pp. 495–513, 2016.

[28] R. V. Rao, V. J. Savsani, and D. P. Vakharia, "Teaching–learning-based optimization: a novel method for constrained mechanical design optimization problems," *Computer-aided design*, vol. 43, no. 3, pp. 303–315, 2011.

[29] Z. W. Geem, J. H. Kim, and G. V. Loganathan, "A new heuristic optimization algorithm: harmony search," *simulation*, vol. 76, no. 2, pp. 60–68, 2001.

[30] S. Satapathy and A. Naik, "Social group optimization (sgo): a new population evolutionary optimization technique," *Complex & Intelligent Systems*, vol. 2, no. 3, pp. 173–203, 2016.

[31] W. Zhang, K. Pan, S. Li, and Y. Wang, "Special forces algorithm: A novel meta-heuristic method for global optimization," *Mathematics and Computers in Simulation*, vol. 213, pp. 394–417, 2023.

[32] J.-S. Pan, L.-G. Zhang, R.-B. Wang, V. Snášel, and S.-C. Chu, "Gannet optimization algorithm: A new metaheuristic algorithm for solving engineering optimization problems," *Mathematics and Computers in Simulation*, vol. 202, pp. 343–373, 2022.

[33] J. Wang, Y. Chen, C. Lu, A. A. Heidari, Z. Wu, and H. Chen, "The status-based optimization: Algorithm and comprehensive performance analysis," *Neurocomputing*, p. 130603, 2025.

[34] Z. Liang, Z. Wang, and A. W. Mohamed, "A novel hybrid algorithm based on improved marine predators algorithm and equilibrium optimizer for parameter extraction of solar photovoltaic models," *Heliyon*, vol. 10, no. 19, 2024.

[35] H. Song, J. Bei, H. Zhang, J. Wang, and P. Zhang, "Hybrid algorithm of differential evolution and flower pollination for global optimization problems," *Expert Systems with Applications*, vol. 237, p. 121402, 2024.

[36] J. Zhao, D. Chen, R. Xiao, J. Chen, J. S. Pan, Z. H. Cui, and H. Wang, "Multi-objective firefly algorithm with adaptive region division," *Applied Soft Computing*, p. 147, 2023.

[37] J. R. Koza, "Genetic programming as a means for programming computers by natural selection," *Statistics and computing*, vol. 4, pp. 87–112, 1994.

[38] X. Xue and J. C.-W. Lin, "Heterogeneous sensor integration through co-evolutionary genetic programming with multiple individual representations," *Swarm and Evolutionary Computation*, vol. 98, p. 102098, 2025.

[39] Q. Zhang, H. Gao, Z.-H. Zhan, J. Li, and H. Zhang, "Growth optimizer: A powerful metaheuristic algorithm for solving continuous and discrete global optimization problems," *Knowledge-Based Systems*, vol. 261, p. 110206, 2023.