



ARTICLE

Genetic Programming-Based Search Strategy Generation Applied to Emergency Material Transportation Scheduling

Jeng-Shyang Pan^{1,2,3}, Cuijing Cao⁴, Shu-Chuan Chu^{2,*}, Lingping Kong⁵, Xingsi Xue⁶ and Jia Zhao⁷

¹School of Information Engineering, Yango University, Fuzhou, China

²School of Artificial Intelligence/School of Future Technology, Nanjing University of Information Science and Technology, Nanjing, China

³Department of Information Management, Chaoyang University of Technology, Taichung, Taiwan

⁴College of Computer Science and Engineering, Shandong University of Science and Technology, Qingdao, China

⁵Faculty of Electrical Engineering and Computer Science, VŠB-Technical University of Ostrava, Ostrava, Czech Republic

⁶Fujian Provincial Key Laboratory of Big Data Mining and Applications, Fujian University of Technology, Fuzhou, China

⁷School of Information Engineering, Jiangxi University of Water Resources and Electric Power, Nanchang, China

*Corresponding Author: Shu-Chuan Chu. Email: scchu0803@gmail.com

Received: 18 May 2026; Accepted: 28 July 2026; Published: 15 September 2026

ABSTRACT: This study proposes a Genetic Programming-based Search Strategy Generation Framework (GP-SSGF) and a novel variant of the tumbleweed algorithm, the genetic programming-based tumbleweed algorithm (GPTA). The framework automates the evolution of search formulas within metaheuristic algorithms, reducing reliance on manually designed update rules and enhancing adaptability. The GPTA algorithm, developed within this framework, employs evolved position-update formulas to improve search efficiency and convergence. Through extensive experiments on the CEC2017 benchmark suite across multiple dimensions, GPTA demonstrates superior solution quality and stability compared with other metaheuristic algorithms. Its practical effectiveness is further validated in emergency material transportation scheduling, where it optimizes resource dispatching under multiple constraints, significantly reducing costs. This research highlights GPTA's potential to solve complex optimization problems and its broad applicability across fields such as emergency management.

KEYWORDS: Genetic programming; tumbleweed algorithm; search strategy generation; emergency scheduling

1 Introduction

As real-world challenges get more intricate, meta-heuristic algorithms have emerged as essential instruments for navigating high-dimensional search spaces and circumventing local optima [1,2]. Meta-heuristic algorithms provide numerous advantages compared to traditional mathematical optimization methods [3,4]. Primarily, they function without need gradient information. Furthermore, they have substantial global exploration capabilities and resilience to local optima [5,6]. Due to these characteristics, meta-heuristic algorithms can achieve near-optimal answers in intricate search spaces. Owing to these advantages, meta-heuristic algorithms are extensively employed across several domains. These domains encompass engineering design, data mining, path planning, and intelligent manufacturing [7]. Common meta-heuristic algorithms encompass the Genetic Algorithm (GA) [8,9], Particle Swarm Optimization (PSO) [10,11], Differential Evolution (DE) [12,13], and Artificial Bee Colony (ABC) [14–16], along with recently introduced nature-inspired algorithms such as the Whale Optimization Algorithm (WOA) [17,18] and the Grey Wolf

Optimizer (GWO) [19–21], among others. The Tumbleweed Algorithm (TA) [22] has garnered interest due to its distinctive search patterns among new approaches. Through the modeling of tumbleweed seed dispersal and seedling growth, TA demonstrates robust global search capabilities and versatility, excelling in various test functions and engineering applications. Despite the commendable performance of these algorithms in practical applications, some significant issues persist. On one hand, various algorithms typically employ fixed or manually crafted position update formulas, rendering this “static update mechanism” challenging to adaptively modify based on the problem’s characteristics or varying search phases; on the other hand, these fixed formulas frequently fail to adequately balance global exploration and local exploitation, leading to algorithms becoming trapped in local optima or converging at a sluggish pace [23]. Moreover, manually crafted formulas are significantly reliant on the expertise of practitioners and heuristic guidelines, resulting in a deficiency of systematicity and adaptability [24]. Consequently, eliminating dependence on manual design methods has emerged as a significant focus in intelligent optimization research.

To address the aforementioned issues, researchers have recently commenced the integration of machine learning techniques into metaheuristic algorithms to augment their intelligence and adaptability. Genetic Programming (GP) offers a viable avenue for improving optimization algorithms owing to its symbolic regression ability and adaptive representation framework [25,26]. In contrast to conventional genetic algorithms that modify parameter values, genetic programming (GP) utilizes a tree-like chromosomal architecture that facilitates the dynamic evolution of computational operators and search techniques, producing well specified mathematical expressions [27]. This property allows it to demonstrate distinct advantages in optimizing meta-heuristic algorithms. Meta-heuristic algorithms typically depend on predetermined search methods to direct the search process. Nevertheless, rigid artificial formulas frequently lack the adaptability to respond to intricate and evolving problem attributes. Conversely, GP can autonomously produce more adaptable search strategies throughout the evolutionary process, hence enhancing the search efficiency and convergence performance of the algorithm. Despite the extensive application of genetic programming in symbolic regression and automatic formula generation, there is a paucity of research focused on improving position update formulae inside meta-heuristic algorithms, and a comprehensive methodological framework has yet to be established. By incorporating GP, optimization algorithms can adaptively formulate update methods informed by the feedback from the search environment, so dynamically modifying the search trajectory and augmenting both global and local search efficacy. In contrast to the static formulas manually crafted by human professionals, the expressions produced by GP exhibit considerable advantages in flexibility and adaptability to problems. Consequently, it offers novel concepts for developing more efficient and intelligent optimization algorithms.

The scheduling of emergency material transportation is a fundamental task within contemporary emergency management systems, with the primary objective of swiftly and judiciously distributing essential materials from various storage locations to impacted areas following an emergency, thereby facilitating prompt rescue efforts and optimal resource allocation [28]. Common application scenarios encompass the delivery and distribution of food, medication, tents, water purification apparatus, and other emergency supplies during significant public emergencies, including earthquakes, floods, typhoons, fires, and outbreaks of infectious illnesses [29]. The rapid urbanization and the increasing frequency of natural catastrophes have rendered the sophistication and intelligence of the emergency logistics system a crucial metric for assessing the emergency management capabilities of a country or region. The optimization level of the dispatching system is directly correlated with the speed of rescue response, the assurance of survival in the disaster zone, the efficiency of resource utilization, and the capacity to manage secondary catastrophes [30]. Consequently, optimizing the dispatch of emergency supplies amongst many restrictions, including time, resources, routes, and diverse demand types, has emerged as a prominent research focus in operations

research optimization and artificial intelligence. The scheduling of emergency material transportation is fundamentally an NP-hard problem, characterized by its multidimensional complexity [31]. The objectives of scheduling frequently encompass minimizing response time, reducing transportation costs, alleviating warehouse strain, and maximizing material demand fulfillment, which are often in conflict and need trade-offs. The impacted regions may be extensively dispersed, the roadway capacity remains ambiguous, and the selection of routes is challenging [32]. This prompts the investigation of hybrid computational intelligence algorithms to synergistically integrate the benefits of several optimization approaches.

As a result, this paper presents the Genetic Programming-based Search Strategy Generation Framework (GP-SSGF). This framework is designed to automatically evolve search formulas within algorithms. In doing so, it lowers the need for manually defined update rules. The primary contributions of this study are as follows:

- (1) A new framework, GP-SSGF, is developed to automatically generate search strategies for metaheuristic algorithms.
- (2) Using the GP-SSGF framework, a novel variant of the tumbleweed algorithm, the genetic programming-based tumbleweed algorithm (GPTA), is created by developing the position update formulas.
- (3) Extensive tests using the CEC2017 benchmark suite are carried out to assess the performance of GPTA in 10, 30, and 50-dimensional issue situations.
- (4) The suggested GPTA is then applied to a real-world situation of emergency material transportation scheduling, demonstrating its practical utility.

The subsequent sections are organized as follows: [Section 2](#) discusses the underlying ideas of the Tumbleweed Algorithm and Genetic Programming, as well as an overview of related research. [Section 3](#) introduces the suggested GP-SSGF and describes the implementation of the enhanced algorithm, GPTA. [Section 4](#) gives a comparative analysis based on experimental data from the CEC2017 benchmark function, followed by a statistical significance analysis. [Section 5](#) explains how to use GPTA to schedule emergency material transportation. [Section 6](#) summarizes the findings and suggests some directions for future research.

2 Fundamentals and Related Works

This section provides a brief overview of the tumbleweed algorithm and genetic programming, with the goal of laying the theoretical groundwork for the development of future techniques. Meanwhile, it analyzes new research on enhancing the tumbleweed algorithm and optimizing the meta-heuristic method with genetic programming.

2.1 Fundamentals

2.1.1 Genetic Programming

Genetic Programming is an extension of standard Genetic Algorithms, first presented by John Koza in the early 1990s. Genetic Programming, like GA, adheres to natural rules such as “survival of the fittest,” genetic recombination, and mutation. This domain-independent bioheuristic technique develops computer code based on a high-level description of the problem. By simulating the process of natural selection, the approach allows those candidate solutions that perform well to have children, hence optimizing the program’s performance until a solution that fits the requirements is discovered. This method is unusual in that it eliminates the need to define each step in detail, allowing the program to learn and improve as it grows.

Because each individual is tree-structured, genes are classified as functions or terminals. All functions contain child nodes. The collection of functions is represented by $F = \{f_1, f_2, \dots, f_n\}$. Terminals, on the

other hand, are evaluated directly, with return values that can be calculated and returned. The terminals are denoted by $T = \{t_1, t_2, \dots, t_n\}$.

This paper provides an example subroutine that is represented by a structure tree and uses the functions and terminals listed above. Fig. 1 depicts a specific schematic.

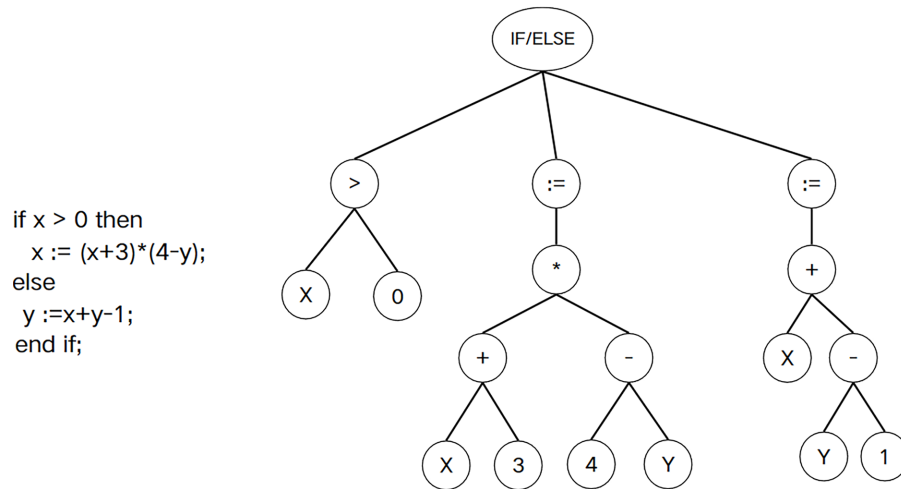


Figure 1: A subroutine and a structure tree.

Selection, crossover, and mutation are essential processes in genetic programming that are often performed randomized based on an individual's fitness. Individuals with superior fitness have a better probability of getting chosen for these operations.

In genetic programming, selection refers to the process of selecting people based on their fitness to form a new generation. It is a key operation in Genetic Programming that seeks to replicate the idea of “survival of the fittest” in natural selection, so that individuals with higher fitness have a better chance of passing on their “genes” to the next generation. Common selection methods include roulette and tournaments.

In contrast, the crossover operation randomly selects two individuals from the current population as parents based on fitness and then randomly selects two spots for crossover in each of these two parents. As a result, two additional offspring are formed. This way, the parents' advantages can be combined to produce higher-performing children. Fig. 2 illustrates an example of crossover.

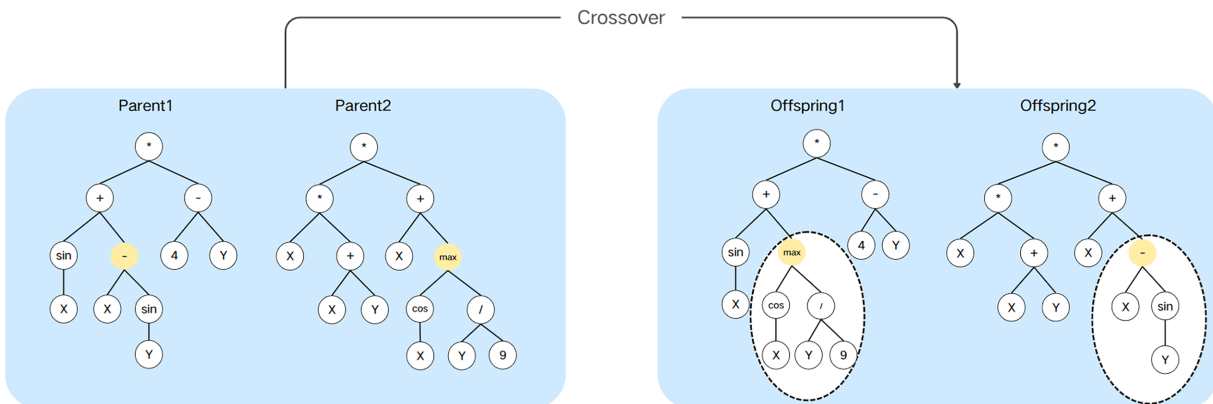


Figure 2: Individual crossover.

The mutation procedure first randomly selects a parent individual based on their fitness, and then randomly selects a mutation site on that individual. The subtree at that mutation point is then removed, and a procedure similar to that used to produce the initial individual is employed to generate a new subtree to be placed in its place. This method introduces novel genetic variants, which contribute to increased population diversity and the exploration of additional solution areas. Fig. 3 illustrates the process of individual mutation.

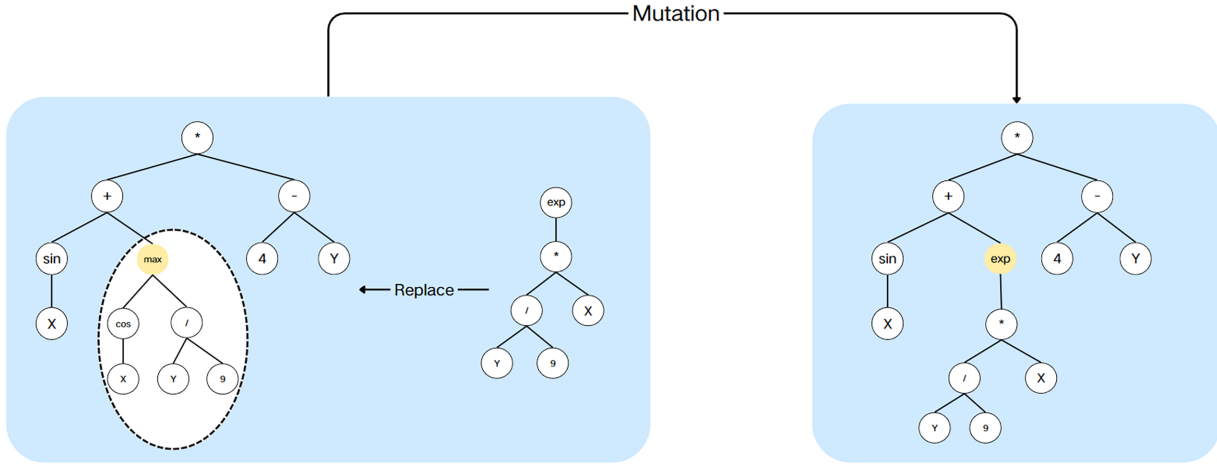


Figure 3: Individual mutation.

2.1.2 Tumbleweed Algorithm

Tumbleweed is a plant that grows in arid and semi-arid parts of the world. It is most typically seen in North American grasslands and deserts. As the plant ages, the roots dry out and the stem structure degrades. When the wind picks up, the plant detaches from its roots and rolls with it. This rolling action aids in the spreading of many seeds. This rolling not only aids seed dissemination, but also enables the plant to reproduce quickly in proper settings. The qualities listed above served as the foundation for the development of the Tumbleweed Algorithm. It is divided into two stages: seedling growth and seed distribution.

Seedling growth: Tumbleweed seeds are normally disseminated during the autumn or winter months, and given the right conditions, the seeds will sprout in the spring or early spring. Despite their adaption to arid conditions, tumbleweeds require appropriate moisture for seed germination, and the ideal temperature range is critical for successful germination. When the seeds sprout, the plant enters the seedling growth stage. The major root system grows first, helping the plant to quickly absorb water and nutrients from the soil. In dry environments where water is scarce, the plant's ability to grow deep roots is critical for identifying water sources deeper in the soil. The effect of these variables on seedlings is referred to as E1. Furthermore, adjacent seedlings compete for resources, hence E2 represents the effect of individual competition.

$$E1 = \frac{r_1}{\frac{fit(X_i^{old})}{sum(fit(X))} + 1} \quad (1)$$

$$E2 = r_2 * (X_{best} - X_i^{old}) \quad (2)$$

where r_1 is a random number expressing the degree of influence by the environment. As r_2 decreases to 0, it represents the degree of effect of the nearby strong seedlings. The variation formula is shown in Eq. (3).

$$r_2 = t * \left(1 - \frac{gen}{Max_gen}\right) \quad (3)$$

From this, the formula for the first stage can be derived, as expressed in Eq. (4).

$$X_i^{new} = X_i^{old} + E1 + E2 \quad (4)$$

Seed Dispersal: As tumbleweeds develop, their branches stiffen to roll and disperse seeds as the wind takes hold. Once a tumbleweed's roots have broken off, it will begin to roll on the ground, propelled by the wind, and eventually release seeds. The process of seed release is random and slow, therefore tumbleweeds are likely to drop some seeds at each stage of rolling. This slow release process guarantees that seeds scatter at multiple locations. According to the literature [33], the seed dispersal distance can be determined using Eq. (5).

$$d = \frac{\mu * H}{F} \quad (5)$$

An iterative process of optimizing the algorithm with this formula resulted in an improved formula for the seed dispersal step, represented by Eq. (6).

$$X_i^{new} = X_{gbest} + V_i * \frac{gen}{Max_gen} \quad (6)$$

2.2 Related Works

2.2.1 Optimization of Metaheuristic Algorithms by Genetic Programming

Genetic programming has emerged as an effective method for optimizing metaheuristics and solving complicated optimization issues across multiple domains. Runka's [34] work improved the Ant System algorithm by evolving decision formulae through GP. They introduced two mechanisms—a roulette wheel selection and a greedy selection—which demonstrated effectiveness in bigger TSP instances. Tavares and pereira [35] developed enhanced ant communication mechanisms using GP's pheromone update techniques, which improved the generalization of ACO in optimization challenges. Poli et al. [24] used GP to optimize particle mobility and search efficiency. Bartashevich et al. [23] enhanced PSO's capabilities in dynamic situations by integrating GP and the Evolutionary Demes Despeciation Algorithm (EDDA), resulting in collective search behavior similar to birds' flying patterns.

Li et al. [36] created a two-stage framework for battery health prediction, combining GP with Binary MOGWO for precise and cost-effective SOH estimate. Burke et al. [37] emphasized GP's significance in automating heuristic design and its inventive potential to solve complicated computational challenges.

Overall, these investigations demonstrate GP's adaptability in optimizing metaheuristics. From improving Ant Colony Optimization and PSO to developing hyperheuristic frameworks, GP has constantly improved algorithmic efficiency while broadening applicability. By automating design and utilizing evolutionary techniques, GP is pushing innovation in metaheuristic development, providing solid answers to challenging issues.

2.2.2 Variants of the Tumbleweed Algorithm

In recent years, the Tumbleweed Algorithm (TA), a revolutionary swarm intelligence optimization algorithm, has received attention due to its superior performance in continuous optimization tasks. However, its limits in binary optimization and sensitivity to local convergence have prompted academics to suggest

a variety of enhancements. Pan et al. [33] adapted TA to handle binary optimization by introducing V-shaped and S-shaped transfer functions, resulting in the Binary Tumbleweed Algorithm (BTA). Their trials on CEC2013 benchmark functions revealed BTA's higher performance on unimodal, multimodal, and composite functions. Yuan et al. [38] introduced another BTA variation employing transfer functions for continuous-to-binary conversion. They validated its competitiveness on 23 benchmark functions and successfully used it to feature selection in intrusion detection datasets. Wang et al. [39] proposed the Parallel Binary Tumbleweed Algorithm (PBTA), which includes a time-varying transfer function, parallel mechanisms, and communication strategies. This strategy improved population diversity and outperformed CEC2013 benchmarks, particularly in feature selection for intrusion detection systems. Minke [40] introduced the Parallel Tumbleweed Algorithm (PTA) to address TA's local convergence difficulty. PTA uses two communication techniques for parallel population processing. PTA demonstrated improved global search capabilities through tests using CEC2017 test functions and aluminum electrolysis parameter optimization. Overall, these studies have significantly improved the applicability of TA to binary optimization and complex real-world problems by introducing transfer functions, parallel mechanisms, and communication strategies, providing valuable insights for the Tumbleweed Algorithm's future development and application.

3 GP-SSGF Framework and GPTA

This section introduces the GP-SSGF framework, which addresses the limits of static update techniques in classic metaheuristic algorithms. This framework allows the algorithm to automatically evolve the search formula, which improves the search process's adaptability and intelligence. On this basis, this study proposes the GPTA algorithm. This section describes in detail how this framework and algorithm function.

Unlike existing genetic programming-based optimization approaches, which typically evolve either heuristic rules or operator-level components in a fixed algorithmic pipeline, the proposed GP-SSGF framework introduces a general search-strategy generation paradigm that directly evolves complete position-update expressions using a unified symbolic structure. In particular, GP-SSGF decouples the search strategy generation process from the underlying metaheuristic, allowing for easy integration with other population-based optimizers.

3.1 GP-SSGF Framework

This section goes into detail about how the GP-SSGF framework was implemented. The system use genetic programming to automatically generate mathematical equations that direct the search process. By substituting manually constructed update formulae, the algorithm's flexibility and intelligence are increased. Table 1 shows the framework's relevant parameters.

Table 1: Parameters of the GP-SSGF framework.

Parameter	Setting
Population size	50
Maximum number of generations	50
Initialization method	The ramped half and half method
Maximum tree depth	3
Selection method	Roulette wheel selection
Crossover probability	0.8
Mutation probability	0.2

The proposed GP-SSGF framework is intended to be a general-purpose search strategy generator that is not dependent on a particular metaheuristic algorithm. Formally, GP-SSGF accepts as input the state information of an optimization process. The framework produces an evolved update function, which is used to replace the initial position update equation of a metaheuristic algorithm. The framework is not limited to the Tumbleweed algorithm. Any population-based optimization approach that uses explicit update rules, such as Particle Swarm Optimization, Differential Evolution, or Grey Wolf Optimizer, can easily incorporate GP-SSGF by replacing their update operators with evolved expressions.

To balance formula diversity and computing complexity, the depth of the tree is set to 3 in this research, allowing for more complicated formulae to be generated without requiring excessive processing. Furthermore, in order to increase population diversity, investigate potential solutions of varying complexity, and ultimately find a more concise and effective solution strategy, this paper employs the ramped half and half method to generate initial individuals with more diverse and adaptable structures.

In this study, the roulette wheel selection approach is used to effectively accelerate the algorithm's convergence. Specifically, the fitness value of each member in the population is first determined. Then, the selection probability P_i and the cumulative selection probability Q_i of each individual are determined, and their formulas are:

$$P_i = \frac{f_i}{\sum_{j=1}^N f_j} \quad (7)$$

$$Q_i = \sum_{k=1}^i P_k \quad (8)$$

where f_i represents the fitness value of individual i , and N specifies the population size. During the selection process, a random number r is created that is uniformly distributed in the interval $[0, 1]$, and the chosen individuals are defined by the cumulative selection probability interval in which the random number falls. For example, if the random number r comes between the interval $[Q_{i-1}, Q_i)$, the i th individual is chosen as the parent. This approach duplicates the evolutionary principle of survival of the fittest in nature, such that people with higher adaption have a better chance of passing on their own excellent genes to their kids, progressively boosting the population's total adaptation.

Crossover and mutation operations use the previously specified subtree crossover and subtree mutation methods.

In this study, the traveler's issue is used to evaluate the performance of the developed formulas. As a result, the Bier127 benchmark instance from TSPLIB was selected for testing. This dataset covers 127 cities, making the challenge large enough to capture the performance differences amongst the techniques without becoming computationally expensive. For medium-sized TSP problems, this dataset strikes a fair compromise between computing economy and algorithm efficacy. It combines the interference of local optimum solutions with the requirement for the algorithm to have a strong global search capability in order to identify the best path. This provides an appropriate challenge for evaluating algorithm performance. Eq. (9) describes the unique fitness function:

$$TotalDistance = \sum_{i=1}^{n-1} D(p_{r_i}, p_{r_{i+1}}) + D(p_{r_n}, p_{r_1}) \quad (9)$$

where $D(p_i, p_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$, $P = \{p_1, p_2, \dots, p_n\}$ denotes the coordinates of the city, $R = \{r_1, r_2, \dots, r_n\}$ denotes the order of the city visits, and $D(p_i, p_j)$ denotes the Euclidean distance from city i to j .

In this study, the Bier127 TSP instance is used as a realistic and challenging fitness evaluation setting to guide the generation of general-purpose update expressions. Although the evaluation is performed on a discrete TSP instance, the evolved expressions are defined over fundamental optimization variables such as position vectors, fitness values, and global best information, which are shared by both continuous and discrete population-based metaheuristics. As a result, the updated formulas are intrinsically problem-independent. CEC2017 functions are not employed directly during the GP evolution stage to avoid overfitting to a given benchmark suite and to guarantee that the evolved search strategy retains generalization capabilities across other issue domains.

Algorithm 1 shows the pseudo-code for the GP evolution formula.

Algorithm 1: GP-SSGF framework

- 1: **Input:** Function set, Terminal set, *MaxDepth*, population size, Maximum Iterations, The evaluation function *F*
 - 2: **Output:** Evolved optimal formula
 - 3: **Initialization:** Use the growth method to initialize the population based on *MaxDepth*. The nodes in the tree are selected from the given set according to the rules of the growth method.
 - 4: The fitness value for each individual is evaluated using *F*.
 - 5: **while** *iter* ≤ *Max_iter* **do**
 - 6: Selection of individuals using the roulette wheel method.
 - 7: Perform crossover and mutation operations.
 - 8: *iter* = *iter* + 1
 - 9: Evaluating fitness values for new individuals
 - 10: **end while**
-

3.2 GPTA Algorithm Based on GP-SSGF

This section proposes the GPTA algorithm using the GP-SSGF architecture. The technique substitutes the standard tumbleweed algorithm's location updating mechanism with a formula that the GP automatically evolves to achieve a better balance of global exploration and local exploitation. GPTA attempts to improve search efficiency, accelerate convergence speed, and increase the algorithm's adaptability and robustness to various challenges.

Based on the position update strategy in the TA algorithm (Eq. (4)), the function and terminal sets required by the GP-SSGF framework to evolve the GPTA update formulation are as follows.

$$\begin{cases} \text{Terminal} : \{X_i, X_{gbest}, fit(X), sum(fit(X)), r1, r2\} \\ \text{Function} : \{+, -, \times, \div\} \end{cases}$$

In the tumbleweed algorithm, X_i represents an individual's position variable, whereas X_{gbest} represents the current global optimal solution. $fit(X)$ and $sum(fit(X))$ represent the current individual's and total fitness values, respectively. The algorithm requires one random number r_1 and a linearly decreasing parameter r_2 . The formulas developed by the GP-SSGF framework are given in Eq. (10).

$$X_i^{new} = \left(\frac{r_2 - X_i^{old}}{sum(fit(X)) \cdot fit(X)} \right) / \left(\frac{X_{gbest}}{r_2} - (X_{gbest} - fit(X)) \right) \quad (10)$$

Eq. (10) shows that the suggested update rule creates an adaptive perturbation mechanism that balances exploration and exploitation through coupled stochastic and nonlinear components. The numerator term

$r_2 - X_i^{old}$ creates a stochastic differential perturbation that increases population heterogeneity by continuously introducing individual-level variation. This method prevents premature convergence by ensuring that the search space remains diverse.

The denominator term $\text{sum}(\text{fit}(X)) \cdot \text{fit}(X)$ creates a self-adaptive normalizing effect that adjusts the step size based on the fitness landscape. Individuals with better fitness values are given smaller update magnitudes to fine-tune local exploitation, whilst weaker individuals are given greater exploratory steps to improve global search capabilities. Furthermore, the coupling of global best information and individual fitness creates a nonlinear feedback loop that allows for adaptive search direction adjustment across different fitness distributions.

Although Eq. (10) lacks a precise analytical convergence proof, it does exhibit certain structural properties. First, reciprocal fitness-based scaling incorporates an implicit limited control mechanism to reduce excessive divergence during late-stage iterations. Second, the nonlinear coupling of global-best guiding and fitness disparities improves directional sensitivity in heterogeneous population distributions. Third, the stochastic perturbation term promotes continuous exploration, lowering the likelihood of stagnation in local optima.

Compared to the previous linearly weighted update structure, the new formula incorporates various fractional structures and nonlinear coupling effects, considerably increasing the expressive capacity and adaptive control capabilities of the search dynamics. When the population reaches a local optimum or the fitness distribution becomes highly concentrated, the stochastic perturbation is effectively amplified in comparison to the adaptive scaling terms, allowing for better escape from local basins and promoting diverse search behavior.

The pseudo-code for GPTA is shown in Algorithm 2.

Algorithm 2: GPTA (Revised for Reproducibility)

```

1: Input: Maximum iterations ( $Max\_iter$ ), population size ( $ps$ ), dimension ( $dim$ ), fitness function ( $F$ ),
   growth cycle ( $cycle$ ), bounds  $[LB, UB]$ 
2: Output: Best solution  $X_{gbest}$  and  $F(X_{gbest})$ 
3: Step 1: Chaotic initialization
4: Generate initial population using Logistic chaotic map:
5:  $x_{k+1} = \mu x_k(1 - x_k)$ ,  $\mu = 4$ 
6: Map chaotic sequence to  $[LB, UB]$ 
7: Evaluate fitness of all individuals
8: Initialize  $X_{gbest}$ 
9: while  $iter < Max\_iter$  do
10:  for  $i = 1$  to  $ps$  do
11:    Step 2: Define search direction
12:     $V_i = X_i^{old} - X_{gbest}$ 
13:    if  $\text{mod}(iter, cycle) < cycle/2$  then
14:      Update using Eq. (10):
15:       $X_i^{new} \leftarrow$  GP-evolved update formula
16:    else
17:      Update using Eq. (6):
18:       $X_i^{new} \leftarrow$  TA seed dispersal formula
19:    end if

```

(Continued)

Algorithm 2 (continued)

```

20:   Step 3: Boundary handling
21:   If  $X_i^{new} \notin [LB, UB]$ , apply reflection:
22:    $X_i^{new} = LB + rand(UB - LB)$  or  $UB - rand(UB - LB)$ 
23:   Evaluate fitness  $F(X_i^{new})$ 
24:   Step 4: Global best update
25:   if  $F(X_i^{new}) < F(X_{gbest})$  then
26:      $X_{gbest} = X_i^{new}$ 
27:   end if
28: end for
29:  $iter = iter + 1$ 
30: end while
31: Return  $X_{gbest}, F(X_{gbest})$ 

```

Fig. 4 depicts the general structure of the GP-SSGF framework and GPTA algorithm, as described before.

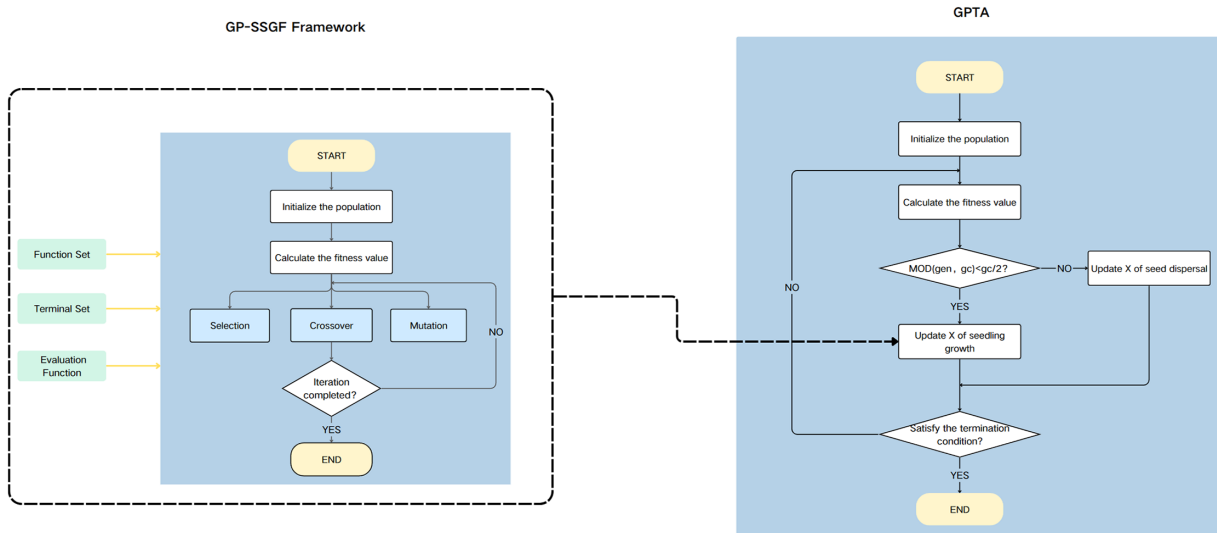


Figure 4: Overall structure diagram.

4 Experiments and Results

This section focuses on evaluating the performance of the proposed GPTA algorithm. To assess the algorithm's optimization capability, a series of comparative experiments are performed on benchmark functions of varying dimensions. The performance of GPTA is compared to numerous conventional methods. To confirm the statistical significance of the results, the Wilcoxon rank-sum test is used to compare the differences between algorithms.

4.1 Experimental Setup

The performance of the GPTA algorithm was evaluated using the CEC2017 standard test function set. The function set includes four categories: unimodal, simple multimodal, hybrid, and composite functions. The unimodal function (F1) is primarily used to assess the algorithm's local search capability and convergence.

speed, as it contains only one global optimal solution. The simple multimodal functions (F2–F10), on the other hand, have numerous local optimal solutions alongside a single global optimum, and are adopted to evaluate the algorithm's global exploration capacity and its capability to escape local optima. Hybrid functions (F11–F20) integrate the characteristics of multiple basic test functions (e.g., shifted and rotated variants). They generally consist of three or more weighted sub-functions, constructing intricate local landscapes to verify an algorithm's adaptability to mixed optimization scenarios. Composite functions (F21–F30) merge features from various hybrid and basic functions. Equipped with diverse function structures and abundant local optima, they can quantify the algorithm's capacity to identify and distinguish different types of optimal solutions within complicated search domains. Comprehensive tests on all four function categories enable a thorough investigation into the GPTA algorithm's performance over optimization problems with different complexity levels.

To confirm that the evaluation findings were reliable and stable, the GPTA algorithm was performed independently 30 times in 10, 30, and 50 dimensions, with a population size of 50 and 5000 function calls. Table 2 shows full parameter settings for the compared methods. To guarantee a fair and unbiased comparison, all benchmark algorithms utilized in this study were constructed with their standard parameter settings as published in the original papers or regularly used configurations in current literature. No extra parameter adjustment was done for the proposed GPTA approach or any competing solution in the test problems. In addition, all experiments were carried out using the MATLAB programming language on a machine with 24 GB RAM and an Intel Core i7 processor running Windows 11.

Table 2: Parameter settings of all compared algorithms.

Algorithm	Parameter Settings
GPTA	cycle = 50, $r_1 \in [0, 1]$, $r_2 \in [0, 2]: 2 \rightarrow 0$
TA	cycle = 50, $r_1 \in [0, 1]$, $r_2 \in [0, 2]: 2 \rightarrow 0$
WOA	$a_1 \in [0, 2]: 2 \rightarrow 0$, $a_2 \in [-2, -1]: -2 \rightarrow -1$, $r_1 \in [0, 1]$, $r_2 \in [0, 1]$
HHO	$E_1 \in [0, 2]: 2 \rightarrow 0$, $E_2 \in [-2, -1]: -2 \rightarrow -1$, $r \in [0, 1]$
PSO	$c = 2.0$, $\omega = 0.8$
BOA	$p = 0.8$, power_exponent = 0.1, sensory_modality = 0.01
SCA	$a = 2$, $r_1 \in [0, a]: a \rightarrow 0$, $r_2 \in [0, 2\pi]$, $r_3 \in [0, 2]$, $r_4 \in [0, 1]$
SOA	$Fc \in [0, 2]: 2 \rightarrow 0$, $r_1 \in [0, 1]$, $r_2 \in [0, 1]$
ASO	$\alpha = 50$, $\beta = 0.2$
CRO	KELossRate = 0.2, MoleColl = 0.2, InitialKE = 1000, $\alpha = 500$, $\beta = 10$, $\delta = 0.2$

4.2 Comparative Results Analysis

4.2.1 Analysis of 10-Dimensional Results

Tables 3 and 4 show the performance comparisons of the methods in 10-dimensional tasks. Overall, GPTA performs well when compared to other algorithms. From the average values, GPTA outperforms most benchmark functions. For numerous functions, such as F3, F7, F12, and F19, GPTA outperforms TA. However, it should be noted that on some functions, TA performs similarly or slightly better than GPTA, demonstrating that the suggested technique does not outperform TA in all circumstances.

Algorithm	BOA			SCA			CRO			ASO			HHO			GPTA		
	Mean	Std		Mean	Std		Mean	Std		Mean	Std		Mean	Std		Mean	Std	
1	9,199,077,653	2,674,254,268	1,503,172,029	735,353,461.5	20,970,126,740	70,070,454,328	15,289,948.7	39,095,899.03	1,119,682,992	1,217,108,424	33,671,741.1	138,765,166						
2	5,8871E+12	1.2386E+13	3,427,147,872	8,739,177,578	3,43722E+14	8,97175E+14	1,061,125,440	2,223,149,105	775,355,300	1,585,602,328	383,840,548	1,853,775,345						
3	12,616.40121	2837.031633	7840.72359	2388.554491	3,918,031.209	11,777,052.58	12,269,8002	5225.647057	6210.88341	1485.11772	691.526256	506.3290004						
4	1265.577504	365.1177277	504.661855	58,14991033	2803.390598	1403.413274	442.253566	42,72923318	518.754614	93.76746006	466.032869	153.6889145						
5	609.3347604	15.64235568	561.953949	9,162099108	656.7047366	20,44237627	544.772205	21,65294331	560.199917	20.0921449	551.710274	23.20753137						
6	660.6596902	10.72352742	627.01353	6,087309767	693.0237196	18,25151316	612.199419	8,887459946	641.053832	12,42953298	622.382269	14,71312632						
7	884.7212749	34.49642884	792.639558	11,787108	1189,664963	85.42768356	744.277927	16,26192474	803.028473	14,98146705	773.328411	24.3816978						
8	876.6551341	11,2867838	859,685775	7,14745544	938,4851526	16,60505908	841,822635	17,09849833	832.239746	9,943677126	838.11619	17,91127039						
9	1737.445933	222.6449653	1225.87561	151,8909343	4784,967386	919,0291991	975.917186	116,4084588	1525.95887	209,8368955	1232.25402	271,2896171						
10	2972.998089	213,8842662	2677,27996	225,1619149	3723,444542	287,4502027	2753.62422	465,463186	2246.48409	243,5610264	2326.98677	260,864946						
11	2353.399041	1106.769215	1342.56554	128,3259563	26,343,47476	22,794,2818	1375.17899	464,1786235	1555.55673	561,5831566	1196.22061	60,04014642						
12	440.5667916	257,561,281.3	50,476,348.7	31,105,708.64	2,434,123,618	1,368,621,344	3,277,955.92	2,847,582,157	4,750,038.04	4,738,385,801	6,793,893.38	11,466,044,22						
13	8,560,049.17	7,857,144,052	270,248,637	206,998,9649	307,761,010.2	356,164,496.3	49,360,1958	75,269,9857	16,697,3872	12,866,11089	13,134.62	11,804,01961						
14	13,234,18967	9952,124105	5216,04632	3704,539583	2,887,793,829	6,485,752.87	6008,69468	5314,87708	3,435,11172	2650,513817	5534,00467	5188,887046						
15	27,660.4036	13,375,98922	9532,71827	6607,085755	11,190,459.36	19,644,205.6	26,320,9321	26,667,11742	9297,66678	4320,08616	14,323,0176	14,598,74829						
16	212,7458353	117,8991937	1917,86308	135,2302604	2702,094046	255,7403331	1987,79601	153,0197471	1988,57028	143,3878536	1857,61842	161,680489						
17	195,4128018	69,24779191	1822,70031	33,825777231	2454,328239	229,999371	1841,00824	67,95566382	1803,74337	65,6637311	1788,06926	51,22988629						
18	23,747,892.57	9,687,230.13	1,262,542.21	925,391,0618	592,244,073.3	556,746,756.5	2,234,266.92	7,452,279,579	17,205,1315	9623,027555	46,962,0352	43,714,96803						
19	378,668,9103	567,459,3294	1,250,13863	22,464,15605	62,048,469.71	68,105,304.72	21,344,9972	36,919,96093	136,157,371	263,085,6876	9899,15229	7888,907342						
20	2281,291274	71,73386354																

23

Table 4: 10-Dimensional comparison results (Part2).

Algorithm	PSO			SOA			TA			WOA			GPTA		
Func_Num	Mean	Std		Mean	Std		Mean	Std		Mean	Std		Mean	Std	
1	577.482,031.4	180.408,016		757.105,333	747.067,343.9		183.627,437.3	185.937,666		451.441,350.1	322.510,757		33.671,741.11	138.765,166	
2	179.491,408.8	196.715,445		703.594,297.1	1,518.669,367		190.505,526	283.647,519		518.037,548.7	1,154.015,020		383,840,548.5	1,853,775,345	
3	11,064.22107	4376.4657		6318.145654	3818.773983		7729.129783	4078.18601		15,549.14295	10,379.477		691.5262563	506.329	
4	449.0828309	24.3382366		463.8226099	39.32797361		461.9653503	42.1131698		496.0204584	72.5549505		466.0328688	153.688914	
5	548.6444941	6.15688197		548.6789726	12.2151391		539.6464367	15.0435038		557.0966067	16.5403096		551.7102742	23.2075314	
6	619.3770386	3.62445477		624.2126923	9.831620449		620.470213	8.16362853		643.2427526	13.6806638		622.3822691	14.7131263	
7	824.2912902	13.5004553		780.4347136	16.9250691		768.1622689	23.2380445		796.9125475	23.3005568		773.3284108	24.3816978	
8	858.1155177	6.18857796		838.1776996	9.60770359		842.7802262	14.0196736		843.7184015	12.5607672		838.1161904	17.9112704	
9	1289.749199	154.368273		1144.691348	141.6276424		1276.500265	230.806173		1722.948193	497.690682		1232.254024	271.289617	
10	2726.426372	212.608673		2337.568686	270.0282548		2337.500931	311.144381		2452.217559	380.847669		2326.986772	260.864946	
11	1292.038457	73.2363951		1344.288245	131.0496761		1254.970889	84.1091141		1313.169981	130.388358		1196.220606	60.0401464	
12	30,904,340.95	17704.028.7		6,037,037,134	4,018,952.188		7,437,638.186	10,193,183.4		4,920,446.059	4,382,201.6		6,793,893.376	11,466,044.2	
13	164,865.3296	132,016.426		48,521.6193	34,986.88722		12,901.74533	10,953.5046		19,639,79442	13,444.2842		13,134.61997	11,804.0196	
14	3479.53119	3737.50745		5665.788618	5255.326398		5520.993057	5079.41673		3836.468092	2062.49824		5534.004673	5188.88705	
15	12,094.56951	14,144.2514		14,726.12166	14,053.77236		12,929.88952	13,737.7534		16,611.66858	10,530.4765		14,323.01755	14,598.7483	
16	1865.181416	109.785174		1854.172688	127.0220689		1808.847439	85.4288413		1978.605375	184.697785		1857.618425	161.680489	
17	1833.817144	45.2090086		1836.401401	68.81642945		1812.057032	46.4412907		1845.045557	85.0468042		1788.069261	51.2298863	
18	582,450.0318	507,347.875		255,216.0533	260,016.1785		28,037,80746	46,885.3637		16,555.96925	12,148.6324		46,962.03522	43,714.968	
19	9432.519606	5615.99249		23,190.57457	21,630.91392		13,381.80963	12,069.8067		198,492.9626	476,886.953		8989.152288	7888.90734	
20	2133.462709	47,026,7715		2228.648403	100.7872826		2121.886629	51,066,7638		2217.977365	102.531289		2145.137229	88.3538099	
21	2322.011174	55.6557849		2256.162021	64.80411409		2306.332839	53,089,4889		2335.447244	57,303,8262		2323.967793	60.3899671	
22	2381.840219	18.746198		2593.526567	536.9815567		2372.014969	239.738913		2378.759255	190.768487		2306.080799	6.08439689	
23	2640.957414	6.3269938		2656.147752	10.83752249		2652.283773	16.7219054		2664.391407	17,331,8384		2671.084343	38.6371024	
24	2763.123558	38.3710724		2775.075714	44.0447154		2737.108167	75.3419339		2786.309717	43.7621605		2783.986694	82.9184711	
25	2977.332039	12.6735999		2972.773044	43.92612037		2973.989064	33.7898627		3034.621324	81.3367978		2932.86269	43.1112268	
26	3185.575405	387.682614		3431.04311	550.0021979		3111.546235	225.616398		3461.619017	498.325761		3021.599277	137.749562	
27	3111.060056	9.86808687		3099.47878	2.293182998		3127.3495	27.9698498		3161.956173	44.5176485		3162.03303	61.3141157	
28	3374.649258	119.365072		3398.257289	108.9443394		3348.77543	144.133348		3436.009037	144.311985		3274.261251	79.4411503	
29	3290.95389	72.9439109		3316.207948	84.53284407		3287.793882	60.2268922		3369.026268	123.954408		3275.570677	82.6138601	
30	2,015,971.121	1,527,813.95		781,700.8193	203,066.8677		3,450,385.786	4,408,843.06		2,582,202.921	2,123,105.5		310,177.3347	384,754.641	
WIN	19			21			15			25			-		

When compared to other algorithms such as BOA, CRO, HHO, PSO, SCA, and SOA, GPTA produces competitive results and outperforms them on a variety of test functions, particularly multimodal and composite functions. In terms of stability, GPTA generally produces lower standard deviations, indicating more consistent performance across separate runs.

Fig. 5 shows the convergence curves in 10 dimensions. Most algorithms converge quickly in the early iterations, although other approaches, such as CRO and SOA, display stalling on specific functions. GPTA's two-stage process maintains a balance between exploration and exploitation, allowing for sustained improvement throughout later iterations while avoiding premature convergence.

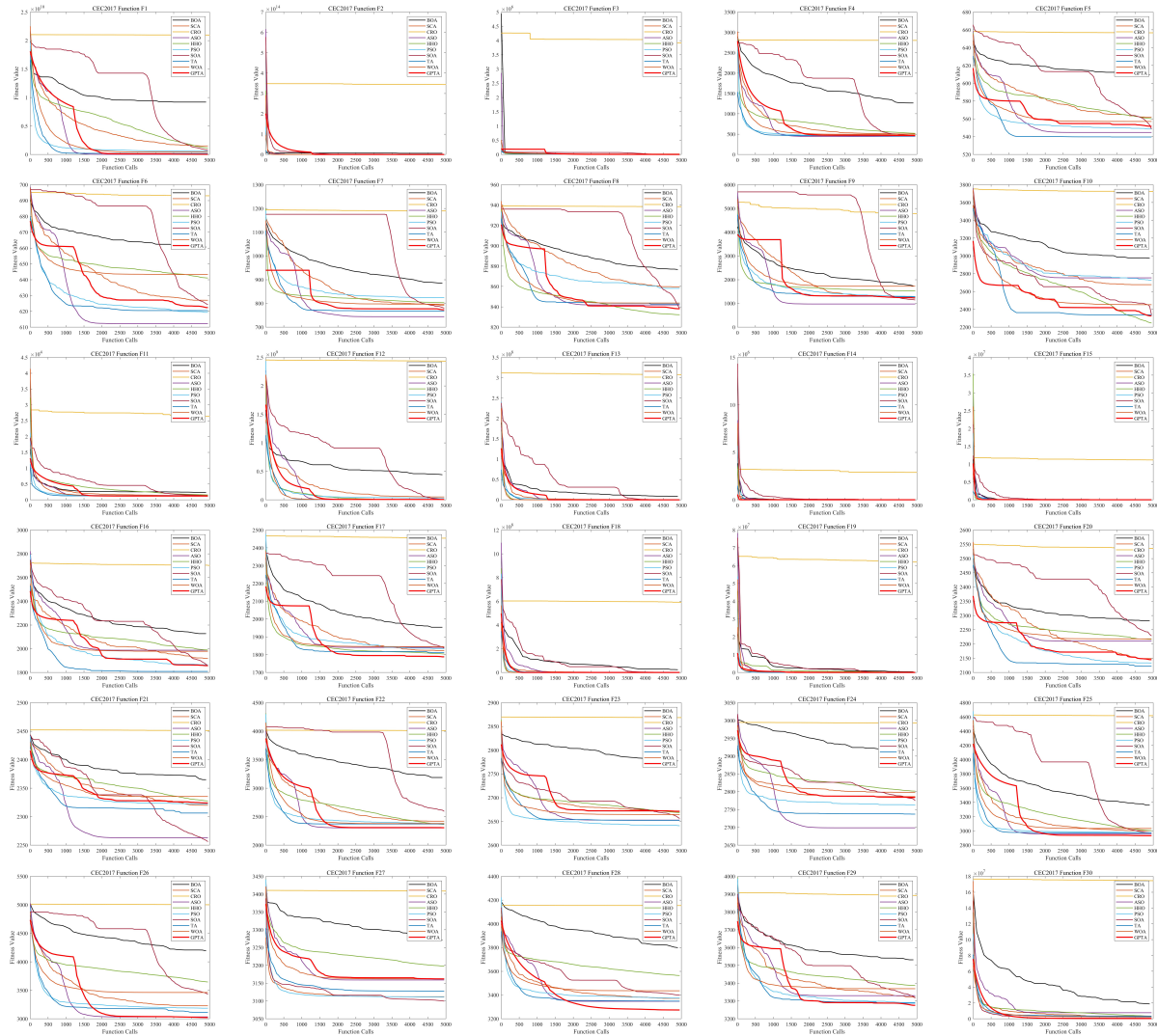


Figure 5: 10-dimensional convergence plot of the algorithm.

4.2.2 Analysis of 30-Dimensional Results

Tables 5 and 6 show the results for 30-dimensional situations. When compared to the 10-dimensional configuration, GPTA outperforms TA on a wider range of test functions, attaining higher results on the majority of benchmark functions. However, there are a few circumstances where TA or other algorithms produce comparable results.

Table 6: 30-Dimensional comparison results (Part2).

Algorithm		PSO			SOA			bib22			WOA			GPTA		
Func_Num		Mean	Std		Mean	Std		Mean	Std		Mean	Std		Mean	Std	
1		18,136.637,104	2,673,455,456		1.5818E+10	5,167,586,283		15,057,736,771	3,108,970,889		16,920,950,169	4,267,229,201		183,639,313.3	192,310,213	
2		8.30228E+36	2.67124E+37		1.2708E+35	5.29035E+35		5.50408E+37	1.77555E+38		1.95657E+40	7.49364E+40		3.92552E+42	2.09894E+43	
3		180,592.6675	43,569.88499		145,155.807	35,775.11852		147,859.0127	36,859.67156		295,471.4496	66,631.0057		122,871.499	44,799.58598	
4		1692.066593	344.2254409		1671.01072	833.3406183		2990.212825	1218.643299		3831.075518	1636.521644		932.4456144	322.3689046	
5		806.8379807	16.73959387		791.759031	31.58999147		823.2571635	47.01818797		876.3705498	60.24948727		749.351839	62.96063299	
6		651.623523	6.079014274		662.133148	8.746852102		662.8809894	15.25067148		687.310296	11.08040063		671.4648306	10.89023294	
7		1655.193083	100.8029441		1217.07967	70.78257818		1371.88317	103.5281674		1355.407495	76.50180795		1177.131176	104.8759703	
8		1116.99803	20.83510533		1053.63309	37.79804691		1099.533229	45.09298025		1107.239543	47.32368255		992.7185809	38.58591169	
9		8746.769576	1579.92183		10,727.7695	4859.176779		10,761.10626	3093.664069		12,681.02402	3538.104528		8882.794104	3913.888117	
10		9098.572614	354.2088098		8755.1753	540.5908194		7831.455469	648.7003911		7842.933191	695.8505877		6255.184718	789.0685329	
11		6674.938363	1679.692211		4484.75703	1493.72008		5429.269178	1135.428835		20,477.73926	9730.630504		1980.323402	765.7195407	
12		1,668,793,675	268,961,892.5		1,246,997,143	830,128,741.8		1,551,524,966	733,297,404.7		1,775,253,968	878,878,842.7		58,440,655.98	55,798,826.93	
13		661,107,866.5	298,432,215.5		583,217,514	880,389,722.7		153,417,035.6	221,975,039.3		133,390,677.7	92,961,906.64		1,293,124.121	2,248,465.722	
14		1,302,097,713	1,196,455,046		1,483,802.48	1,862,422.527		1,129,992.232	1,499,774.91		3,383,990.968	3,948,447,805		582,324.4095	912,731.7515	
15		152,072,026.4	85,609,178.12		56,629,699.7	60,753,285.26		10,764,436.12	35,236,860.08		27,116,765.05	41,212,844.3		156,845.9598	194,106.088	
16		4029.083032	290.6074961		3888.45923	397.2026095		3644.726332	426.0355506		4583.411871	791.5124925		4053.112666	690.3825646	
17		2832.8157	191.2230238		2679.77265	330.7578542		2510.317224	219.652454		2921.108997	372.6436408		2583.028643	291.9865867	
18		15,536,056.23	7,661,226.187		8,796,986.62	8,232,846.083		9,712,872.332	7,998,588.242		26,103,919.07	38,181,208.48		11,151,296.47	17,061,656.88	
19		202,179,413.1	132,558,304.2		59,072,231.1	63,163,989.91		23,074,113.57	31,978,382.64		54,563,939.25	38,372,469.88		3,047,197.223	2,953,360.514	
20		3106.870261	156.0567535		2933.6433	192.8861945		2810.928448	196.2016095		2995.604444	247,912,3576		2927,574,398	223,039,4314	
21		2585.940184	23.2778432		2586.32527	42.51224643		2580.526887	42.3164967		2683.470429	59.05735941		2613.953913	75.75783775	
22		8881.657626	2684.762069		10,059,8473	1658.268816		8033.010728	2394.956032		8794.893295	1505.722472		7183.208487	2046.632198	
23		2941.398812	15.59462827		2988.14396	44.58392345		3139.343002	81.59438331		3214.593464	111.5142662		3179.480164	196.8893225	
24		3081.296673	16.6824547		3126.11851	34.64336288		3259.419461	70.56249609		3328.623573	103.0327147		3346.831013	207.5035036	
25		4246.889862	404.465062		3419.89329	188.2678021		3887.837674	281.8917709		3518.110765	138.8963039		3021.967461	48.17181505	
26		7002.852096	410.5868373		6927.11216	478.8061735		8074.940749	930.0135648		9062.051425	1174.459175		6973.172839	1799.210957	
27		3367.38361	32.52715279		3401.65692	100.212101		3639.624061	97.54411392		3582.773669	122.4572494		3399.810133	324.2777795	
28		4217.583998	239.9924908		5846.53098	1440.363146		4420.646723	322.9559506		4963.892661	409.8259321		3555.280699	201.0705405	
29		5027.700323	167.8009568		5268.76852	534.9696204		5088.310041	431.898135		5946.089141	900.4535431		4635.267801	497.7300432	
30		108,381,178.6	56,107,619.97		86,847,743.5	53,265,721.41		28,385,694.18	30,787,834.32		126,150,907.2	96,773,193.9		25,271,746.63	122,010,064.5	
WTN		22			22			21			28			-		

GPTA outperforms PSO, SOA, and ASO in most areas, particularly for composite functions. In functions such as F25, F27, F28, F29, and F30, GPTA performs better, implying more scalability to higher dimensions. Nonetheless, the performance benefit is not consistent across functions.

Fig. 6 illustrates the convergence trend in 30 dimensions. GPTA often converges to lower fitness values in later rounds, yet in certain circumstances it exhibits slower initial convergence, which could be related to the algorithm's exploratory nature in the early stages.

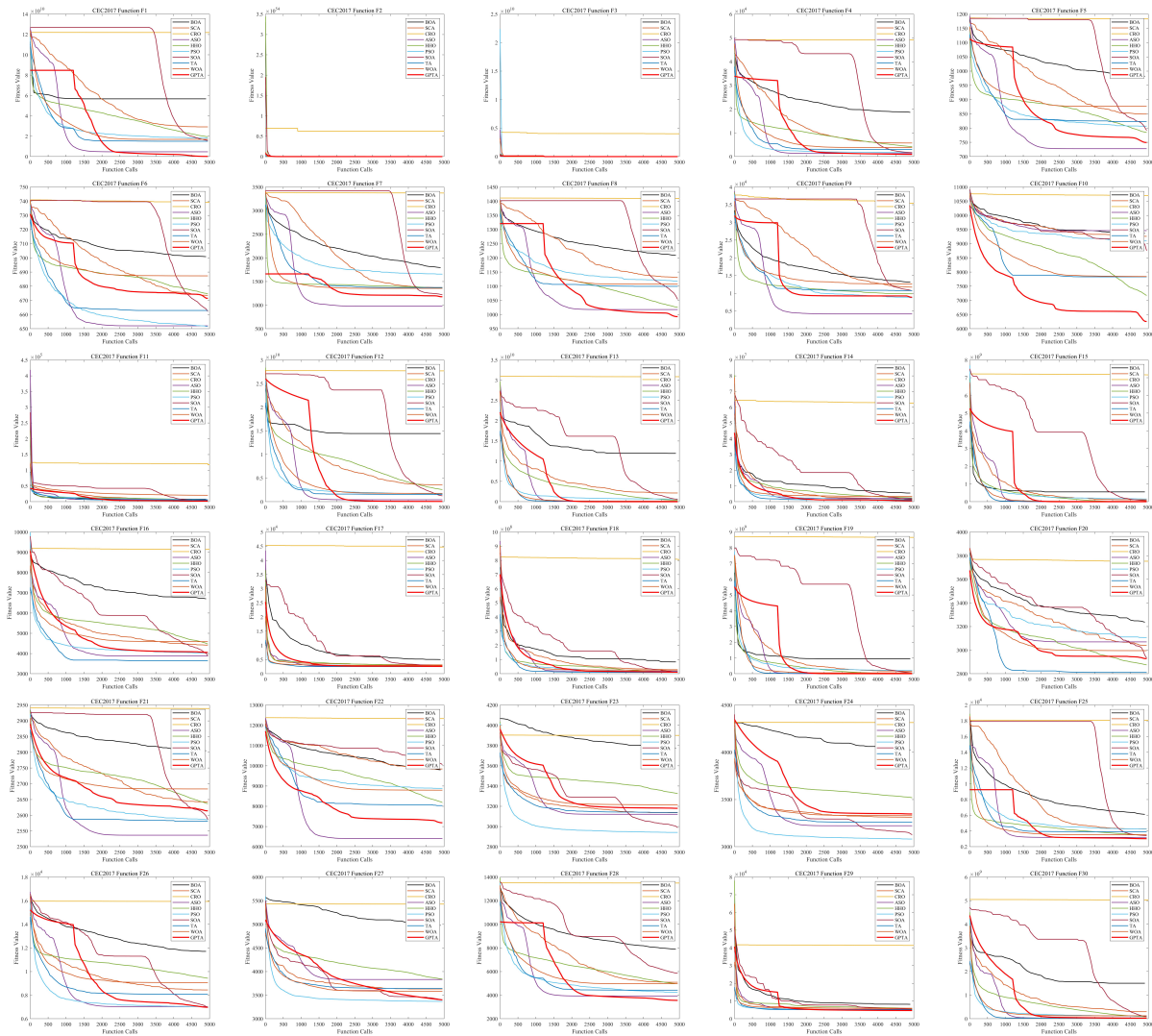


Figure 6: 30-dimensional convergence plot of the algorithm.

4.2.3 Analysis of 50-Dimensional Results

In 50-dimensional experiments, the results presented in Tables 7 and 8 demonstrate that GPTA remains competitive in tackling high-dimensional optimization issues. Despite increased problem complexity, GPTA is able to provide high-quality solutions in the majority of benchmark functions.

Table 7: 50-Dimensional comparison results (Part1).

Algorithm	SCA			CRO			ASO			HHO			GPTA		
Func_Num	Mean	Std		Mean	Std		Mean	Std		Mean	Std		Mean	Std	
1	1.09945E+11	8.437826E+112	82.976,058,967	10.962,251,315	2.4682E+11	31.288,547,505	24,865,496,129	7.043,517,932	56,722,054,059	5.803,708,807	7.321,026,321	2,279,230,792	7.321,026,321	2,279,230,792	
2	4.92709E+90	1.93224E+91	1.46884E+77	7.88477E+77	1.03039E+94	5.58879E+94	3.30634E+68	8.3263E+68	2.32717E+76	1.14911E+77	3.7834E+70	1.97418E+71	3.7834E+70	1.97418E+71	
3	612.604.8205	195.405.0994	335.068.3439	75.702.27739	4.87504.914	9.651.774.186	288.104.2795	58.477.12239	225.248.7626	37.174.17277	290.738.0943	66.090.19185	290.738.0943	66.090.19185	
4	37.222717633	4970.61946	19.459.40281	4730.942123	115.673.1458	17.439.32228	514.2.99884	1405.250719	12.880.42931	2029.46362	3493.10724	2784.542141	3493.10724	2784.542141	
5	1315.108832	44.4713879	1178.8872259	45.60707127	1687.401155	73.6920016	979.7785201	63.50169045	101.392411	33.76740856	932.9525625	48.94082362	932.9525625	48.94082362	
6	716.4673491	5.739809557	693.61842253	8.91087430	752.6383779	10.82030094	66.978182085	13.195508043	685.3599146	5.392795607	678.8477693	9.70015507	678.8477693	9.70015507	
7	2761.988298	137.8764145	2256.244304	189.2943464	5854.218012	368.0380151	1399.847014	120.0448224	1965.36374	54.50677419	1630.889823	108.4468865	1630.889823	108.4468865	
8	1642.359168	39.1595899	1495.931521	43.05433887	1981.420429	51.38260118	1270.872853	83.41194946	1325.603078	38.42076564	1259.518304	64.15380754	1259.518304	64.15380754	
9	43.759.4206	388.2032839	38.344.39581	6975.789193	94.764.96184	10.975.70806	22.606.64774	83.434.933284	34.490.55785	3558.590884	22.939.3847	4167.077658	22.939.3847	4167.077658	
10	16.106.19605	475.0572458	15.775.07557	484.6209137	17.806.70167	694.6041318	16.023.46897	1055.240291	12.795.96780	989.4504998	10.846.10854	949.7585645	10.846.10854	949.7585645	
11	25.190.36716	2639.116881	19.858.57806	4358.164464	305.903.652	540.202.2733	24.784.52625	6798.837153	12.966.87628	2328.629089	8875.308281	4500.823326	8875.308281	4500.823326	
12	82.843.189.587	11.551.925.994	29.607877.995	8.539.545.159	1.30032E+11	30.314.726.526	4.548.704.987	1.874.496.181	27.589.087.303	8.272.428.541	1.560.424.729	2.695.579.115	1.560.424.729	2.695.579.115	
13	38.980.646.919	14.759.996.627	9.659.586.894	3.042.368.088	84.751.394.923	17.300.206.520	931.652.7561	558.919.233.7	8.828.504.573	4.947.858.761	17.581.075.84	70.501.629.73	17.581.075.84	70.501.629.73	
14	97770.074.35	54.781.250.19	17715.375.51	9.97737.31.362	403.097.871.4	204.163.324.6	16.028.927.69	12.075.443.53	27.874.347.98	29.846.312.12	9.468.007.754	13.231.596.93	9.468.007.754	13.231.596.93	
15	9.576.652.482	2.659165.358	1.701.281.312	652.678.248	29.773.849.643	11.827.283.987	63.702.481.7	94.485.304.79	575.398.081.5	602.595.824.6	3.103.855.311	12.372.421.52	3.103.855.311	12.372.421.52	
16	10.882.04571	1506.900199	6845.477638	425.5887985	14.621.36799	2742.858884	5393.41287	699.8539973	6253.708193	851.1301082	5436.157468	1046.508347	5436.157468	1046.508347	
17	12.460.40708	4725.61131	5852.947923	606.2765511	1.467.926.48	1.861.013.645	4405.774499	545.9541607	4367.039056	526.9781303	3697.464745	400.989881	3697.464745	400.989881	
18	201.756.991.6	70.436.671.25	109.494.370.9	69.645.459.63	1.185.638.824	598.665.043.1	52.284.704.63	30.459.474.56	67.223.989.05	42.226.800.08	20.307.953.52	25.107.689.57	20.307.953.52	25.107.689.57	
19	4.438.394.280	1.325.306.279	1.168.184.866	474.046.354.3	12.957.145.374	3.849.788.330	8.814.239.899	6.953.907.562	133.652.523.3	151.791.851.9	7.704.168.002	11.751.498.53	7.704.168.002	11.751.498.53	
20	4596.039654	247.0019811	4505.710588	280.4809794	5480.396644	271.837597	44.43.717766	411.2346238	3705.971065	286.861364	3687.004703	382.0256028	3687.004703	382.0256028	
21	3312.758261	82.39071333	3016.789002	48.85057307	3554.996132	86.05845242	2803.666175	71.74598031	308.6.870839	70.65517062	2951.205704	172.1750799	2951.205704	172.1750799	
22	17.792.01832	474.7594566	17.467.78044	426.6503499	19.652.73654	870.9507555	17.694.4432	84.3.4272519	14.427.87333	979.3642381	12.395.47091	963.7684494	12.395.47091	963.7684494	
23	4850.031736	231.3913853	3831.338014	103.5575006	5198.298604	325.3389529	3775.309827	200.053178	4103.089748	232.6750074	3833.357152	288.8968937	3833.357152	288.8968937	
24	5364.279574	371.8301924	3979.06701	79.72241923	5809.167182	390.9130565	6901.343017	174.1703782	4383.254371	243.0357228	4088.54905	340.8962352	4088.54905	340.8962352	
25	16.396.93201	12.3234548	12.543.52025	2123.365785	6.1147.93.433	9490.817669	350.3.47407	816.595354	8532.170793	959.4274204	4260.70573	312.66014	4260.70573	312.66014	
26	18.881.95273	1096.964772	12.843.87032	1131.278976	32.446.6.6613	3313.260757	11.254.45312	1166.022154	13.963.48324	905.1825613	9716.390078	2620.328675	9716.390078	2620.328675	
27	7466.235012	676.7968468	5250.315589	350.9726582	8713.293716	857.1973376	5771.881798	625.1924405	5744.048565	779.584455	5211.677251	684.4140437	5211.677251	684.4140437	
28	13.782.05367	942.8197123	10.765.96033	1124.424859	24.211.63088	3982.820219	6705.56483	687.3591192	8761.893879	607.8704921	5470.086127	1003.177476	5470.086127	1003.177476	
29	136.491.2912	110.015.7006	11.804.24586	3218.774629	2.041.147.025	2.214.365.035	9170.949257	24.21.34074	12.986.89692	427.0.833446	6476.531935	1073.081967	6476.531935	1073.081967	
30	7.9597784.129	2.814.213.258	2.069.765.954	850.250.651.6	20.289.838.832	5.860.058.401	374.012.603.4	81.859.115.24	980.259.514.2	673.431.499.2	104.028.115.2	278.271.706.9	104.028.115.2	278.271.706.9	
WIN	30	28	30	28	30	28	21	29			-				

GPTA outperforms TA on a wide number of test functions, while both algorithms perform similarly on others, including F14, F23, F24, and F27. When compared to other algorithms such as PSO, BOA, SCA, CRO, and HHO, GPTA generally outperforms them and has advantages in numerous multimodal and hybrid functions, which can be observed from [Tables 7 and 8](#).

[Fig. 7](#) depicts the convergence of multiple algorithms on the 50-dimensional CEC2017 test function set. GPTA outperforms the other algorithms, with a lower fitness value for final convergence, implying that it can discover a better solution more precisely when tackling these specific function issues. For example, in the F5 function, when the number of function calls reaches 5000, the fitness value of GPTA is significantly lower than that of other algorithms' curves; a similar trend is also seen in the F10 and F14 functions, indicating that GPTA's optimization mechanism is better able to mine out potential better solution regions, resulting in higher convergence accuracy. Furthermore, the overall curve of GPTA continues to decline, indicating that it gradually approaches the better solution throughout the iteration process with no major fluctuations or stagnation, indicating that the algorithm has a better balancing ability and is motivated by continuous optimization. This provides solid support and a foundation for its use in the solution of high-dimensional difficult optimization problems.

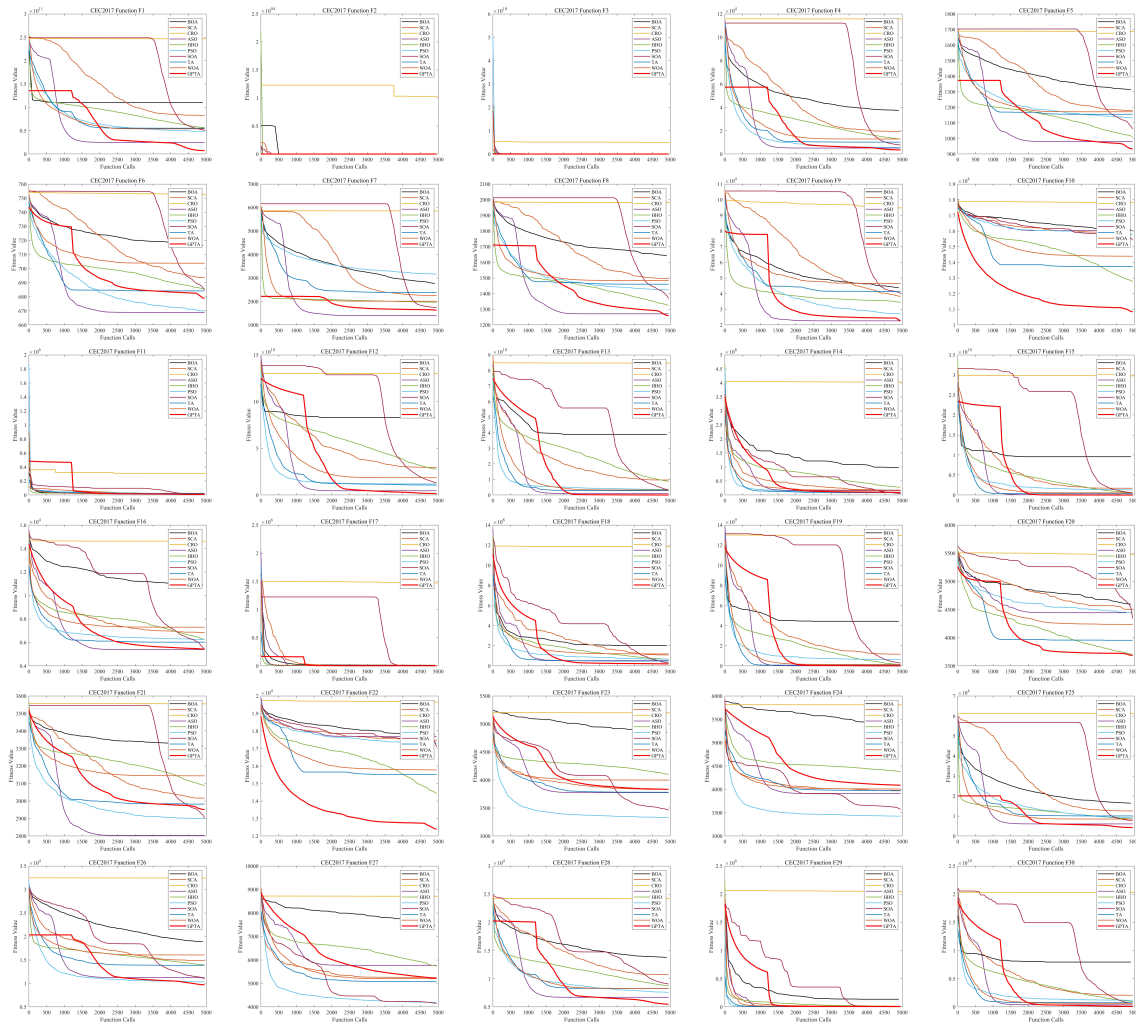


Figure 7: 50-dimensional convergence plot of the algorithm.

4.3 Statistical Significance Analysis

Although the comparative tests in [Section 4.2](#) show that the proposed algorithm outperforms others, it is still required to determine whether these differences are statistically significant rather than due to chance. To this goal, the Wilcoxon rank-sum test, a well-known nonparametric statistical test, is employed to determine the significance of performance differences between the proposed approach and the compared methods.

[Tables 9–11](#) show the Wilcoxon signed-rank test results for the proposed algorithm and comparative algorithms at 10, 30, and 50 dimensions. The significance level for this experiment was set to 0.05. The symbol “+” indicates that the suggested method greatly outperforms the comparative algorithm, “–” suggests that it performs significantly worse, and “=” indicates that no significant difference is found.

Table 9: Wilcoxon test results on 10-D functions.

Func_Num	BOA	SCA	CRO	ASO	HHO	PSO	SOA	bib22	WOA
1	+	+	+	=	+	+	+	+	+
2	+	+	+	+	+	–	+	–	+
3	+	+	+	+	+	+	+	+	+
4	+	+	+	=	+	–	–	–	+
5	+	+	+	=	=	=	=	–	=
6	+	+	+	–	+	=	=	=	+
7	+	+	+	–	+	+	=	=	+
8	+	+	+	=	=	+	=	=	=
9	+	=	+	–	+	=	=	=	+
10	+	+	+	+	=	+	=	=	+
11	+	+	+	+	+	+	+	+	+
12	+	+	+	=	=	+	=	=	=
13	+	+	+	+	=	+	+	=	+
14	+	=	+	=	=	–	=	=	=
15	+	=	+	+	=	=	=	=	=
16	+	=	+	+	+	=	=	=	+
17	+	+	+	+	=	+	+	+	+
18	+	+	+	=	–	+	+	=	–
19	+	+	+	=	+	=	+	=	+
20	+	=	+	+	+	=	+	=	+
21	+	=	+	–	=	=	–	=	=
22	+	+	+	–	+	+	+	+	+
23	+	=	+	=	=	–	=	=	=
24	+	=	+	–	=	–	–	–	=
25	+	+	+	+	+	+	+	+	+
26	+	+	+	=	+	=	+	+	+
27	+	–	+	=	+	–	–	–	=
28	+	+	+	=	+	+	+	=	+
29	+	+	+	+	+	=	=	=	+
30	+	+	+	+	+	+	+	+	+
+/=/–	30/0/0	21/8/1	30/0/0	12/12/6	18/11/1	14/10/6	14/12/4	8/17/5	20/9/1

Table 10: Wilcoxon test results on 30-D functions.

Func_Num	BOA	SCA	CRO	ASO	HHO	PSO	SOA	bib22	WOA
1	+	+	+	+	+	+	+	+	+
2	+	−	+	−	+	−	=	−	−
3	=	=	+	=	−	+	+	+	+
4	+	+	+	+	+	+	+	+	+
5	+	+	+	=	+	+	+	+	+
6	+	=	+	−	=	−	−	−	+
7	+	+	+	−	+	+	=	+	+
8	+	+	+	+	+	+	+	+	+
9	+	+	+	−	+	=	=	+	+
10	+	+	+	+	+	+	+	+	+
11	+	+	+	+	+	+	+	+	+
12	+	+	+	+	+	+	+	+	+
13	+	+	+	+	+	+	+	+	+
14	+	+	+	+	+	+	+	=	+
15	+	+	+	=	+	+	+	=	+
16	+	+	+	=	+	=	=	−	+
17	+	+	+	=	+	+	=	=	+
18	+	+	+	+	=	+	=	=	+
19	+	+	+	=	+	+	+	+	+
20	+	+	+	+	=	+	=	−	=
21	+	=	+	−	=	−	=	−	+
22	+	+	+	=	+	+	+	+	+
23	+	=	+	=	+	−	−	=	=
24	+	=	+	−	+	−	−	=	=
25	+	+	+	+	+	+	+	+	+
26	+	+	+	=	+	=	=	+	+
27	+	+	+	+	+	=	+	+	+
28	+	+	+	+	+	+	+	+	+
29	+	+	+	+	+	+	+	+	+
30	+	+	+	+	+	+	+	+	+
+ / = / −	29/1/0	24/5/1	30/0/0	15/9/6	25/4/1	21/4/5	18/9/3	19/6/5	26/3/1

Table 9 shows that the Wilcoxon rank-sum test findings give statistical support to the comparison. Overall, GPTA outperforms most competing algorithms on the bulk of benchmark functions.

When compared to BOA and CRO, GPTA consistently outperforms them in most test instances (30/0/0), demonstrating significant statistical evidence of increased performance in these comparisons. GPTA outperforms SCA, WOA, and HHO on most functions, with certain examples showing no statistically significant difference.

Table 11: Wilcoxon test results on 50-D functions.

Func_Num	BOA	SCA	CRO	ASO	HHO	PSO	SOA	bib22	WOA
1	+	+	+	+	+	+	+	+	+
2	+	+	+	−	+	−	=	+	+
3	+	+	+	=	−	+	+	=	=
4	+	+	+	+	+	+	+	+	+
5	+	+	+	+	+	+	+	+	+
6	+	+	+	−	+	−	=	=	+
7	+	+	+	−	+	+	+	+	+
8	+	+	+	=	+	+	+	+	+
9	+	+	+	=	+	+	+	+	+
10	+	+	+	+	+	+	+	+	+
11	+	+	+	+	+	+	+	+	+
12	+	+	+	+	+	+	+	+	+
13	+	+	+	+	+	+	+	+	+
14	+	+	+	+	+	=	=	=	=
15	+	+	+	+	+	+	+	+	+
16	+	+	+	=	+	+	=	+	+
17	+	+	+	+	+	+	+	+	+
18	+	+	+	+	+	+	+	+	+
19	+	+	+	+	+	+	+	+	+
20	+	+	+	+	=	+	+	+	+
21	+	=	+	−	+	=	=	=	+
22	+	+	+	+	+	+	+	+	+
23	+	=	+	=	+	−	−	=	+
24	+	=	+	=	+	−	−	=	=
25	+	+	+	+	+	+	+	+	+
26	+	+	+	+	+	=	=	+	+
27	+	=	+	+	+	−	−	=	=
28	+	+	+	+	+	+	+	+	+
29	+	+	+	+	+	+	+	+	+
30	+	+	+	+	+	+	+	+	+
+/=/−	30/0/0	26/4/0	30/0/0	20/6/4	28/1/1	22/3/5	21/6/3	23/7/0	26/4/0

The findings (14/10/6) show that GPTA outperforms PSO in many functions, while there are a few examples where the differences are not significant. When compared to TA (8/17/5), the data indicate that GPTA and TA perform similarly generally, with GPTA showing benefits only on some functions and TA outperforming on others.

It is also noticed that performance discrepancies grow more prominent in higher-dimensional settings, when GPTA performs better on more benchmark functions than in lower-dimensional circumstances.

As seen in [Table 10](#), practically all algorithms display a widening gap with GPTA in 30-dimensional problem. Notably, the advantage over PSO and SOA becomes more obvious, progressing from a minor lead in lower dimensions to a large advantage. More notably, when compared to the original TA, the number

of “+” symbols grows from 8 in 10 dimensions to 19 in 30 dimensions, highlighting GPTA’s advantage in high-dimensional issues.

Overall, in 50-dimensional instances, GPTA outperforms most other algorithms on the bulk of benchmark functions. GPTA outperforms ASO, PSO, and SOA in most cases, however a few functions show no meaningful difference. GPTA outperforms TA in higher-dimensional scenarios against lower-dimensional settings, implying superior scalability. However, the improvement is not consistent among benchmark functions.

5 GPTA Applied to Emergency Material Transportation Scheduling

To demonstrate the GPTA algorithm’s ability to be utilized in real-world issues, this study applies it to the emergency material transportation scheduling problem. Emergency material transportation scheduling is crucial in disaster relief and other crises. The goal is to efficiently deliver emergency supplies to diverse demand points within limited time and resources to meet the demands of affected individuals.

5.1 Problem Description

Following a sudden disaster, the prompt and effective delivery of emergency supplies from the distribution center to each disaster site is critical to ensuring the stability of the disaster area and minimizing losses. The geographic location of catastrophe sites is dispersed, the demand is diverse, and there is frequently time sensitivity; for example, certain places must complete the delivery of resources within a specific time frame or suffer more significant consequences. At the same time, the emergency transportation procedure is constrained by resources such as vehicle count, load capacity, and maximum driving distance.

This study analyzes an emergency material transportation scheduling problem that takes into account the restrictions of time window limitation, service time, vehicle load capacity, and maximum driving distance. We start at the distribution center and organize several emergency vehicles with limited carrying capacity to distribute materials to multiple disaster sites, with the goal of optimizing vehicle usage, path arrangement, and scheduling order to minimize total transportation costs, while meeting the demand and time requirements of all disaster sites.

Specifically, the problem has the following characteristics:

Multi-objective trade-off: The optimization objective incorporates total trip distance, service efficiency, and time-window penalty to achieve inexpensive and practical transportation schedules under many operational constraints.

Complex resource constraints: Vehicles have a limited number and maximum load and distance.

Strict time window constraints: Each catastrophe site has an allowable earliest and latest delivery time, and fines will be paid for either too early or too late delivery.

Non-negligible service time: The vehicle requires a specific amount of time to accomplish the delivery task at the impacted point, which impacts path scheduling and time control.

To address the aforementioned issues, this work develops a mathematical model based on this topic that enables scientific decision-making and effective execution of emergency material scheduling.

5.2 Mathematical Model

In order to simplify the problem and meet the actual emergency material distribution needs, this paper makes the following basic assumptions in the modeling process:

- (1) The location of the distribution center is fixed, and the vehicles uniformly depart from the distribution center and return to the distribution center after completing the distribution task.
- (2) The vehicles are of the same type and have the same maximum load Q and maximum distance traveled $D_{\max}$.
- (3) Each vehicle can make multiple stops at multiple affected points during a distribution, but must meet the load and maximum distance traveled limits.
- (4) The demand at the affected points is known and indivisible, i.e., the demand at each affected point must be delivered by one vehicle at one time.
- (5) The distance between all nodes (distribution centers and affected points) is known and satisfies symmetry, i.e., $d_{ij} = d_{ji}$.
- (6) The affected points all have service time window constraints, and the vehicle arrival time will incur a corresponding time window penalty cost if it is advanced or delayed.
- (7) The fixed vehicle trip cost and the fuel consumption cost per unit distance traveled are known fixed values.

For the purpose of subsequent mathematical modeling and solution, the main symbols involved in this paper are defined as [Table 12](#).

Table 12: Symbol description.

Symbol	Meaning
N	Collection of affected sites (excluding distribution centers)
0	Distribution center number
K	Number of available vehicles
d_{ij}	Distance from node i to node j
q_i	Demand at the affected point i
Q	Maximum weight of each vehicle
$D_{\max}$	Maximum distance traveled by each vehicle
ET_1^i, LT_1^i	Required time window
ET_2^i, LT_2^i	Acceptable time window
ST_i	Service time of disaster site i
v_{ij}	Speed of vehicles traveling between node i and node j
F	Fixed cost of utilization per vehicle
c_2	Fuel consumption cost factor per unit distance
ce_w, cl_w	Time window penalty weights for early and late arrivals
P_1	Unit value of emergency supplies (\$/t)
H	Penalty costs for hard time window violations
t_i	Time it takes for the vehicle to reach affected point i
α, β, γ	Weighting coefficients in the objective function

To specify the decision-making behavior in the model, the following decision variables are set:

- $x_{ijk} = \begin{cases} 1, & \text{If vehicle } k \text{ traveled from node } i \text{ to node } j \\ 0, & \text{otherwise} \end{cases}$
- t_i : the time when the vehicle reaches the affected point i ;

Although GPTA is a continuous optimization technique, the emergency material transportation scheduling problem is expressed as a routing optimization problem using a priority-based solution representation. GPTA generates continuous decision variables that correspond to the priority of visiting a catastrophe location. To determine a possible visitation sequence, the catastrophe sites are sorted in ascending order based on their priority levels. The sequence is then assigned sequentially to vehicles, beginning at the distribution facility. During the construction process, the vehicle load capacity, maximum travel distance, and time-window limits are all evaluated. When a constraint is breached, the current vehicle route is ended and a new vehicle is dispatched to continue serving the remaining disaster areas. This transforms the GPTA's continuous search space into feasible transportation routes for evaluating the goal function.

Eq. (11) depicts the objective function in this situation based on the information provided previously. The objective function has three components: total journey distance L , cumulative service time, and time-window penalty cost C_3 . These three phrases describe transportation efficiency and service excellence during emergency material distribution.

$$\begin{aligned}
 \min \quad & Z = \alpha \cdot L + \beta \cdot \sum_{i \in N} ST_i + \gamma \cdot C_3 \\
 \text{s.t.} \quad & \sum_{j \in N} x_{ijk} = 1, \quad \forall i \in N, \forall k \in K \\
 & \sum_{i \in N} x_{ijk} = 1, \quad \forall j \in N, \forall k \in K \\
 & \sum_{i \in N} \sum_{j \in N} q_j \cdot x_{ijk} \leq Q, \quad \forall k \in K \\
 & \sum_{i \in N} \sum_{j \in N} d_{ij} \cdot x_{ijk} \leq D_{\max}, \quad \forall k \in K \\
 & t_j \geq t_i + \frac{d_{ij}}{v} + ST_i - M(1 - x_{ijk}), \quad \forall i, j \in N, \forall k \in K \\
 & ET_1^i \leq t_i \leq LT_1^i, \quad \forall i \in N \\
 & x_{ijk} \in \{0, 1\}, \quad \forall i, j \in N, \forall k \in K
 \end{aligned}$$

where Eq. (12) denotes the total distance traveled by the vehicle.

$$L = \sum_{k=1}^K \sum_{i, j \in N} d_{ij} \cdot x_{ijk} \quad (12)$$

The time window penalty cost C_3 is expressed as Eq. (13)

$$C_3 = \sum_{i \in N} \left[\begin{cases} (ET_1^i - t_i) \cdot c_{ew} \cdot P_1 \cdot q_i, & t_i < ET_1^i \\ (t_i - LT_1^i) \cdot c_{lw} \cdot P_1 \cdot q_i, & t_i > LT_1^i \\ H, & t_i < ET_2^i \text{ or } t_i > LT_2^i \\ 0, & \text{otherwise} \end{cases} \right] \quad (13)$$

During the decoding process, infeasible transportation schemes caused by vehicle capacity, travel distance, or time-window breaches are dealt with using the penalty mechanism outlined non Eq. (13). Candidate solutions that violate the constraints face higher penalty costs, resulting in lower fitness values during optimization. As a result, GPTA gradually steers the population toward viable scheduling solutions while retaining significant exploration capability.

5.3 Experiment Analysis

The experiment's test data includes the geographic location of client nodes, material demand, service time window, service duration, inter-customer distance matrix, and vehicle speed matrix, all of which were imported from an Excel file. The main parameters for this experiment are as follows:

- Maximum number of vehicles: 6;
- Maximum distance traveled by a single vehicle: 500 km;
- Maximum single vehicle load: 5 t;
- Fixed usage cost per vehicle: 200;
- Unit fuel price: 3 Yuan/L;
- Desired time window penalty factor: $cew = 0.001$, $clw = 0.004$;
- Materiel unit value: 6000 Yuan/t;
- Penalty for violation of hard time window set to constant $H = 200$;
- Service time converted uniformly from minutes to hourly units;
- The parameters are set to a population size of 30, a maximum number of iterations of 3000, and the variable range is set to $[-100, 100]$.

The algorithm saves the resulting optimal path scheme and its cost components. Among these, the fuel cost is computed individually and not included in the overall cost. Fig. 8 provides detailed details. The GPTA algorithm achieves a large reduction in overall cost, demonstrating the algorithm's good cost control capacity in emergency material transportation scheduling, which may effectively cut resource consumption and enhance the economy of rescue operations. The algorithm effectively optimizes various objectives, as evidenced by the obvious values of fuel cost and time penalty cost.

```
Total cost: 2.038584e+02
Fuel cost: 961.500000
Time penalty cost: 169.983500
0—>18—>3—>10—>13—>17—>0—>16—>5—>4—>0—>20—>12—>7—>14—>11—>1—>0—>15—>8—>19—>9—>0—>2—>6—>0
```

Figure 8: Cost and route information.

This study compares the proposed GPTA's optimization performance to TA, WOA, HHO, PSO, BOA, SCA, and SOA. To assure the fairness of the comparison results, all eight algorithms are run using the same experimental setup and arithmetic data, with similar parameter settings.

The data in Table 13 shows that GPTA has a lower mean value than the other indicated algorithms. The mean value is the algorithm's average optimization outcome after numerous runs, and a lower mean value indicates that GPTA is more effective at finding a better scheduling solution across multiple runs, hence improving the efficiency of emergency supply delivery. The standard deviation of GPTA is 17.92732, indicating that the result swings less across repeated runs and is more stable and reliable. In comparison, the standard deviation of the other algorithms is rather substantial, indicating that GPTA can maintain more constant performance, whilst the other algorithms' results are more susceptible to changes in conditions.

Although fuel consumption costs are not directly included in the objective function, they are presented as an additional performance indicator because they indicate the practical transportation efficiency of various scheduling strategies. Table 14 shows that the average fuel consumption cost of GPTA is 872.21, somewhat more than WOA and SOA but much lower than other algorithms. This suggests that GPTA can successfully reduce fuel consumption costs and increase energy efficiency by optimizing transportation routes and scheduling schedules. This GPTA advantage not only helps to minimize operating costs, but it also plays an important part in emergency supply transportation. This is because smart energy utilization guarantees that

supplies reach affected areas quickly and at a lower cost. Furthermore, lower fuel costs contribute to a lower environmental effect, which is consistent with the standards of sustainable development.

Table 13: Algorithm performance comparison.

Algorithm	Mean	Standard Deviation
GPTA	226.87	17.93
TA	232.61	21.17
WOA	239.81	22.12
HHO	235.36	24.78
PSO	246.30	24.87
BOA	252.50	24.17
SCA	250.41	22.10
SOA	229.97	24.75

Table 14: Comparison of fuel cost of algorithms.

Algorithm	Fuel Cost
GPTA	872.21
TA	901.30
WOA	856.15
HHO	888.08
PSO	891.07
BOA	913.77
SCA	891.49
SOA	864.05

The two subfigures in Fig. 9 depict the convergence graph of the method and the convergence curves of the GPTA algorithm and other comparative algorithms, respectively. From Fig. 9a, it can be observed that the algorithm swiftly converges to a better solution in the first few rounds of iterations, and then performs a fine search to eventually achieve a stable solution. The experimental findings show that under the multi-objective of optimizing path planning and time window limitations at the same time, GPTA has strong convergence and stability, can effectively balance the cost and customer service level, and proves the practicality of the algorithm. Comparison with other algorithms reveals that, as shown in Fig. 9b, in contrast, TA, BOA, and SCA drop slowly during the initial stage, WOA and HHO show certain variations in the middle of iteration, and PSO and SOA have comparatively high ultimate convergence values. These events show that GPTA has significant superiority in the emergency material transportation scheduling problem.

This section tests the GPTA in transportation scheduling scenarios for emergency supplies. Applying the GPTA algorithm to emergency material transportation scheduling not only demonstrates the technique's applicability in real-world applications. It also displays outstanding performance in multi-objective optimization. Furthermore, it emphasizes the algorithm's capacity to handle complex restrictions. This provides a unique and practical option for the dispatch of emergency rescue materials. In conclusion, the GPTA method has significant scientific and practical value in the field of optimization due to its outstanding

performance and broad application possibilities. It also provides a solid platform for future research and practical applications.

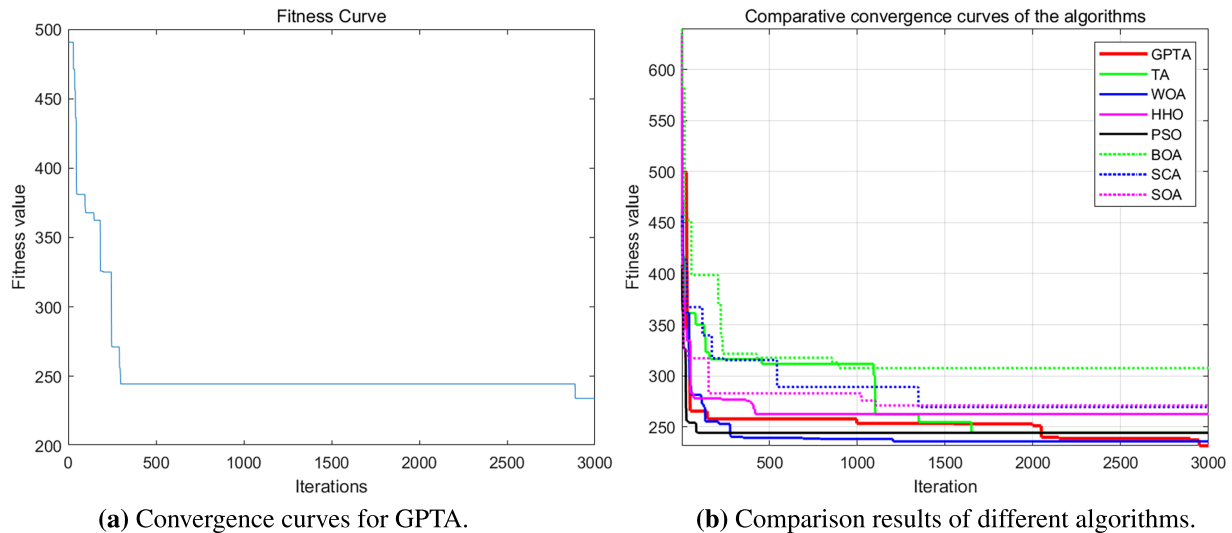


Figure 9: Convergence process curve. (a) Convergence curves for GPTA. (b) Comparison results of different algorithms.

6 Conclusion and Future Work

This study proposes a unique search strategy generation framework, the Genetic Programming-based Search Strategy Generation Framework (GP-SSGF), and develops a new form of the tumbleweed algorithm known as GPTA. The framework tries to address the issue that position updating strategies in classic metaheuristic algorithms are based on manual design and lack adaptability. It fully utilizes genetic programming's symbolic regression power to enable autonomous evolution of search formulas, hence increasing the adaptability and intelligence of the search process. Based on this framework, an improved tumbleweed algorithm, GPTA, is developed, which improves search performance by substituting the traditional TA's position update expression with the automatically evolved GP formula. The produced formulation integrates tactics such as differential guidance mechanism, nonlinear structure modification, and dynamic step size management, resulting in a better balance of global exploration and localization. Multidimensional experiments (10, 30, and 50 dimensions) on the CEC2017 standard test function set demonstrate GPTA's combined advantages in terms of convergence accuracy, stability, and resilience. GPTA outperforms PSO, BOA, HHO, CRO, and other traditional and developing intelligent algorithms on a variety of test functions. Furthermore, GPTA is applied to the emergency material transportation scheduling problem, which successfully reduces total transportation costs while meeting time window and service constraints, and demonstrates good engineering application value in path planning, vehicle utilization, and scheduling order optimization.

Although GPTA revealed considerable benefits in this trial, numerous prospective research avenues warrant additional investigation. First and foremost, the existing set of genetic programming functions is quite simple. In the future, new operators may be added to improve the algorithm's search capability and ability to tackle complex issues. Second, the current evaluation of automatically generated formulae is mostly focused on the algorithm's final performance, with no systematic review of the formula's quality, interpretability, or generalizability. Creating a scientific evaluation index system may more fully measure the benefits and drawbacks of formulae and influence the optimization direction of genetic programming. Third,

the existing emergency transportation model is based on deterministic assumptions, which do not explicitly account for dynamic uncertainties such as demand fluctuations, traffic congestion, or vehicle breakdowns. Future research can expand the suggested framework to stochastic or dynamic optimization settings, such as resilient optimization, scenario-based modeling, or real-time adaptive dispatching systems, to improve its application in complex disaster scenarios. Finally, GPTA can be applied to more realistic problems, such as production scheduling, logistics and distribution, and power system optimization, to demonstrate its effectiveness and adaptability in many settings.

Acknowledgement: Not applicable.

Funding Statement: The authors received no specific funding for this study.

Author Contributions: The authors confirm contribution to the paper as follows: Conceptualization, Jeng-Shyang Pan and Xingsi Xue; methodology, Cuijing Cao; validation, Cuijing Cao and Jeng-Shyang Pan; formal analysis, Cuijing Cao; data curation, Cuijing Cao; writing—original draft preparation, Cuijing Cao; writing—review and editing, Shu-Chuan Chu; visualization, Cuijing Cao; supervision, Lingping Kong; project administration, Jia Zhao. All authors reviewed and approved the final version of the manuscript.

Availability of Data and Materials: The data that support the findings of this study are openly available in GitHub at <https://github.com/Caocuijing/Metaheuristic-Algorithms>.

Ethics Approval: Not applicable.

Conflicts of Interest: Given his role as Associate Editor of this journal, Xingsi Xue had no involvement in the peer review of this article and had no access to information regarding its peer review. Full responsibility for the editorial process for this article was delegated to another journal editor. The authors declare no other conflicts of interest.

References

1. Tomar V, Bansal M, Singh P. Metaheuristic algorithms for optimization: a brief review. *Eng Proc.* 2024;59(1):238.
2. Sharma V, Tripathi AK. A systematic review of meta-heuristic algorithms in IoT based application. *Array.* 2022;14(1):100164. doi:10.1016/j.array.2022.100164.
3. Rajwar K, Deep K, Das S. An exhaustive review of the metaheuristic algorithms for search and optimization: taxonomy, applications, and open challenges. *Artif Intell Rev.* 2023;56(11):13187–257. doi:10.1007/s10462-023-10470-y.
4. Turgut OE, Turgut MS, Krtepe E. A systematic review of the emerging metaheuristic algorithms on solving complex optimization problems. *Neural Comput Appl.* 2023;35(19):14275–378. doi:10.1007/s00521-023-08481-5.
5. Pan JS, Hu P, Snášel V, Chu SC. A survey on binary metaheuristic algorithms and their engineering applications. *Artif Intell Rev.* 2023;56(7):6101–67.
6. Emambocus BAS, Jasser MB, Amphawan A. A survey on the optimization of artificial neural networks using swarm intelligence algorithms. *IEEE Access.* 2023;11:1280–94.
7. Tang J, Liu G, Pan Q. A review on representative swarm intelligence algorithms for solving optimization problems: applications and trends. *IEEE/CAA J Autom Sin.* 2021;8(10):1627–43. doi:10.1109/jas.2021.1004129.
8. Lambora A, Gupta K, Chopra K. Genetic algorithm—a literature review. In: *Proceedings of the 2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon); 2019 Feb 14–16; Faridabad, India.* p. 380–4.
9. Holland JH. Genetic algorithms. *Sci Am.* 1992;267(1):66–73.
10. Jain M, Saihpal V, Singh N, Singh SB. An overview of variants and advancements of PSO algorithm. *Appl Sci.* 2022;12(17):8392. doi:10.3390/app12178392.
11. Marini F, Walczak B. Particle swarm optimization (PSO). A tutorial. *Chemometr Intell Lab.* 2015;149:153–65.

12. Storn R, Price K. Differential evolution-a simple and efficient heuristic for global optimization over continuous spaces. *J Global Optim.* 1997;11(4):341–59.
13. Kachitvichyanukul V. Comparison of three evolutionary algorithms: GA, PSO, and DE. *Ind Eng Manag Syst.* 2012;11(3):215–23.
14. Karaboga D, Akay B. A comparative study of artificial bee colony algorithm. *Appl Math Comput.* 2009;214(1):108–32. doi:10.1016/j.amc.2009.03.090.
15. Karaboga D. Artificial bee colony algorithm. *Scholarpedia.* 2010;5(3):6915.
16. Karaboga D, Gorkemli B, Ozturk C, Karaboga N. A comprehensive survey: artificial bee colony (ABC) algorithm and applications. *Artif Intell Rev.* 2014;42:21–57.
17. Gharehchopogh FS, Gholizadeh H. A comprehensive survey: whale optimization algorithm and its applications. *Swarm Evol Comput.* 2019;48:1–24.
18. Mirjalili S, Lewis A. The whale optimization algorithm. *Adv Eng Softw.* 2016;95(12):51–67. doi:10.1016/j.advengsoft.2016.01.008.
19. Hatta NM, Zain AM, Sallehuddin R, Shayfull Z, Yusoff Y. Recent studies on optimisation method of Grey Wolf Optimiser (GWO): a review (2014–2017). *Artif Intell Rev.* 2019;52(4):2651–83. doi:10.1007/s10462-018-9634-2.
20. Faris H, Aljarah I, Al-Betar MA, Mirjalili S. Grey wolf optimizer: a review of recent variants and applications. *Neural Comput Appl.* 2018;30:413–35.
21. Du Z, Ni S, Pan JS, Chu SC. A surrogate-assisted multi-objective grey wolf optimizer for empty-heavy train allocation considering coordinated line utilization balance. *J Bionic Eng.* 2025;22(1):383–97. doi:10.1007/s42235-024-00613-4.
22. Yang Qy, Chu SC, Liang A, Pan JS. Tumbleweed algorithm and its application for solving location problem of logistics distribution center. In: *Proceedings of the Fourteenth International Conference on Genetic and Evolutionary Computing*; 2021 Oct 21–23; Jilin, China. Cham, Switzerland: Springer; 2022. p. 641–52.
23. Bartashevich P, Bakurov I, Mostaghim S, Vanneschi L. PSO-based search rules for aerial swarms against unexplored vector fields via genetic programming. In: *Parallel Problem Solving from Nature–PPSN XV: 15th International Conference*; 2018 Sep 8–12; Coimbra, Portugal. Cham, Switzerland: Springer; 2018. p. 41–53.
24. Poli R, Langdon WB, Holland O. Extending particle swarm optimisation via genetic programming. In: *European Conference on Genetic Programming*. Cham, Switzerland: Springer; 2005. p. 291–300.
25. Sette S, Boullart L. Genetic programming: principles and applications. *Eng Appl Artif Intell.* 2001;14(6):727–36.
26. Ahvanooy MT, Li Q, Wu M, Wang S. A survey of genetic programming and its applications. *KSII Trans Internet Inf Syst.* 2019;13(4):1765–94. doi:10.3837/tiis.2019.04.002.
27. McKay RI, Hoai NX, Whigham PA, Shan Y, O'Neill M. Grammar-based genetic programming: a survey. *Genet Program Evolvable Mach.* 2010;11:365–96. doi:10.1007/s10710-010-9109-y.
28. Hu H, He J, He X, Yang W, Nie J, Ran B. Emergency material scheduling optimization model and algorithms: a review. *J Traffic Transp Eng Engl Ed.* 2019;6(5):441–54. doi:10.1016/j.jtte.2019.07.001.
29. Liu J, Xie K. Emergency materials transportation model in disasters based on dynamic programming and ant colony optimization. *Kybernetes.* 2017;46(4):656–71. doi:10.1108/k-02-2016-0028.
30. Jia L, Xie K. Preparation and scheduling system of emergency supplies in disasters. *Kybernetes.* 2015;44(3):423–39. doi:10.1108/k-11-2014-0248.
31. Li J, Xiong Y, Jiang B. Distribution fairness in emergency material vehicle scheduling based on uncertain time and demand. *J Uncertain Syst.* 2021;14(02):2150014. doi:10.1142/s1752890921500148.
32. Beheshtinia MA, Jozi A, Fathi M. Optimizing disaster relief goods distribution and transportation: a mathematical model and metaheuristic algorithms. *Appl Math Sci Eng.* 2023;31(1):2252980.
33. Pan JS, Wang GL, Yang D, Chu SC. New optimization method based on binary tumbleweed algorithm. In: *International Conference on Intelligent Information Hiding and Multimedia Signal Processing*. Cham, Switzerland: Springer; 2022. p. 21–31.
34. Runka A. Evolving an edge selection formula for ant colony optimization. In: *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*; 2009 Jul 8–12; Montreal, QC, Canada. p. 1075–82.

35. Tavares J, Pereira FB. Evolving strategies for updating pheromone trails: a case study with the TSP. In: International Conference on Parallel Problem Solving from Nature. Cham, Switzerland: Springer; 2010. p. 523–32.
36. Li X, Jin Z, Li S, Yu D, Zhang J, Zhang W. Li-ion battery state of health prediction through metaheuristic algorithms and genetic programming. *Energy Rep.* 2024;12:368–80. doi:10.1016/j.egy.2024.06.038.
37. Burke EK, Hyde MR, Kendall G, Ochoa G, Ozcan E, Woodward JR. Exploring hyper-heuristic methodologies with genetic programming. In: Computational intelligence: collaboration, fusion and emergence. Vol. 168 of studies in computational intelligence. Cham, Switzerland: Springer; 2009. p. 177–201.
38. Yuan X, Pan JS, Chu SC, Snášel V. Binary tumbleweed algorithm for application of feature selection. In: International Conference on Intelligent Information Hiding and Multimedia Signal Processing. Cham, Switzerland: Springer; 2022. p. 13–20.
39. Wang RB, Geng FD, Xu L. Parallel binary tumbleweed algorithm and its application for intrusion detection systems. *J Netw Intell.* 2024;9(1):599–612.
40. Minke H. Optimization of aluminum electrolytic process parameters based on parallel tumbleweed algorithm. In: Proceedings of the 2023 IEEE 11th Joint International Information Technology and Artificial Intelligence Conference (ITAIC); 2023 Dec 8–10; Chongqing, China. Vol. 11. p. 587–93.