

Article

A Globally Adaptive Ant Colony System with Stagnation Recovery and Candidate-List Search for Traveling Salesman Problems

Shang Wang ^{1,*} , Yajuan Zhang ¹ and Linjie Li ²¹ Engineering Training Center, Beijing Polytechnic University, Beijing 100176, China; zhangyajuan@bpu.edu.cn² School of Aeronautical Engineering, Beijing Polytechnic University, Beijing 100176, China; lilinjie@bpu.edu.cn

* Correspondence: wangshang@bpu.edu.cn

Abstract

The Traveling Salesman Problem (TSP) is a fundamental NP-hard combinatorial optimization problem with broad applications in logistics, scheduling, and satellite mission planning. While Ant Colony Optimization (ACO) offers distributed search and positive feedback, conventional variants suffer from premature convergence and quadratic construction costs that limit scalability. We propose the Globally Adaptive Ant Colony System (GACS), which integrates three synergistic mechanisms: (1) K -nearest neighbor candidate-list pruning that reduces per-step construction complexity from $O(n)$ to $O(K)$; (2) a globally adaptive pheromone weighting scheme that dynamically calibrates reinforcement intensity as the search matures; and (3) an adaptive stagnation recovery mechanism that applies pheromone smoothing to escape local optima. Numerical experiments demonstrate that GACS consistently outperforms four traditional ACO baselines under an equivalent time budget. On a large benchmark set from TSPLIB, GACS achieves highly competitive results against various state-of-the-art metaheuristics, with non-parametric statistical tests confirming its significant superiority in both solution quality and convergence rank. Ablation and sensitivity analyses verify that all three mechanisms are individually indispensable and that the framework is robust to parameter perturbation. Specifically, the evaporation rate and stagnation threshold are identified as the most critical parameters affecting performance, while the smoothing and adaptive range parameters exhibit low sensitivity. These results establish GACS as a lightweight, scalable, and adaptable framework for the TSP.

Keywords: traveling salesman problems; globally adaptive ant colony system; candidate-list search; candidate-list search space compression; ant colony optimization



Academic Editors: Pierluigi Fanelli and Valerio Belardi

Received: 24 May 2026

Revised: 26 June 2026

Accepted: 27 June 2026

Published: 30 June 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

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

1. Introduction

The TSP is a well-known NP-hard combinatorial optimization problem focused on determining the shortest Hamiltonian cycle that visits a predefined set of cities exactly once, serving as a cornerstone for urban logistics and distribution center optimization [1]. Its applications span modern transportation, autonomous navigation, and industrial scheduling, including multi-UAV path planning [2], disaster relief resource allocation, and satellite mission planning [3,4], where variable observation durations and time-dependent constraints introduce combinatorial complexities that extend beyond the classical TSP formulation. These diverse applications underscore that optimizing TSP solutions remains essential for reducing operational costs and enhancing system efficiency across multiple industrial domains.

This study addresses the symmetric TSP, where the travel cost between any pair of cities is identical in both directions, and all test instances are symmetric Euclidean benchmarks.

As practical instances continue to grow in scale, traditional exact algorithms such as branch-and-bound [5] encounter severe computational bottlenecks [6]. The TSP can be formulated as an integer program with binary decision variables and subtour elimination constraints [7]; its NP-hardness, established by Papadimitriou [8], implies that the solution space grows factorially at $(n - 1)!/2$, rendering exact methods intractable for instances beyond a few hundred cities. Consequently, heuristic and metaheuristic strategies, including evolutionary algorithms [9] and Deep Reinforcement Learning (DRL)-based approaches [10], have gained prominence for approximating near-optimal solutions. Nevertheless, the persistent efficiency–accuracy trade-off remains a fundamental challenge, particularly for large-scale instances.

High-performance solvers such as the Lin–Kernighan–Helsgaun (LKH) heuristic [11], reinforcement learning-enhanced LKH [12], Concorde [13,14], and the Edge Assembly Crossover-based Genetic Algorithm (EAX-GA) [15] deliver near-optimal solutions for moderate-sized instances, yet their high computational complexity and structural overhead motivate the development of lightweight swarm-based alternatives. Over the past two decades, metaheuristics have consistently provided viable solutions across all instance sizes by prioritizing optimization efficiency. These approaches encompass both population-based paradigms—including Genetic Algorithms (GA), Particle Swarm Optimization (PSO), and ACO—and trajectory-based methods such as Simulated Annealing (SA) and Tabu Search; a conceptual comparison of these metaheuristic paradigms is provided in [16]. This trade-off drives the development of algorithms that balance exploration and exploitation within feasible computational budgets.

Among swarm intelligence paradigms, ACO has emerged as one of the most successful metaheuristics for solving the TSP due to its intrinsic positive feedback and distributed search capabilities. Since its inception by Dorigo et al. [17] and the subsequent development of the Ant Colony System (ACS) [18], several influential ACO variants have been proposed, including the Max–Min Ant System (MMAS) [19], which confines pheromone values to a bounded interval, and the Rank-based Ant System (ASrank) [20], which restricts pheromone updates to top-ranked ants.

Contemporary research has pursued enhancements along two complementary directions. The first focuses on improving information utilization mechanisms, including individual memory structures [21], multi-strategy fusion combining node clustering and adaptive search [22], and parameter optimization approaches [23]. The second direction enhances search space exploration through the integration of local search procedures, particularly k -Opt neighborhood search [24], or through pheromone smoothing strategies: Li et al. [25] combined differential information with heuristic smoothing, while Pérez-Carabaza et al. [26] integrated originality reinforcement with pheromone smoothing in the rank-based framework.

Despite these advancements, conventional ACO still struggles with rapidly increasing computational costs and stagnation during large-scale searches. Most existing enhancement strategies operate on the full $O(n^2)$ pheromone matrix, whose construction cost becomes prohibitive for large-scale instances. Multi-strategy approaches [22] effectively address stagnation but introduce additional parameters that exacerbate the tuning burden [23]. While pheromone smoothing techniques [25,26] mitigate premature convergence, they do not simultaneously address construction efficiency and parameter robustness. These limitations necessitate more sophisticated mechanisms for search space compression and adaptive recovery to maintain high solution quality at scale.

As highlighted in recent reviews of swarm intelligence [27], no existing approach simultaneously addresses all three bottlenecks, motivating the development of an integrated framework that jointly tackles construction efficiency, adaptive recovery, and parameter robustness.

To address these challenges, we propose the Globally Adaptive Ant Colony System (GACS), a lightweight yet robust ACO framework that integrates three synergistic mechanisms. The core contributions of this work are threefold: First, the candidate-list-based search space compression mechanism reduces the per-step branching factor from $O(n)$ to $O(K)$, significantly accelerating convergence while preserving solution accuracy—unlike full-matrix approaches [25,26], GACS achieves near-linear scalability through K -nearest neighbor (KNN) pruning. Second, a globally adaptive weighting strategy dynamically regulates the reinforcement intensity of best-known solutions—addressing the parameter sensitivity bottleneck identified in [23] through a bounded, self-calibrating weight scheme. Third, an adaptive stagnation recovery strategy detects stagnation conditions and forces the system to escape local optima plateaus via pheromone smoothing—extending the smoothing concept of [25] with a triggered recovery mechanism that provides effective escape capability from deep local optima.

The remainder of this paper is organized as follows: Section 2 details the GACS algorithmic framework, including comparative analyses with Classic Ant Colony Optimization (C-ACO), Ant Colony Optimization–Genetic Algorithm Hybrid (ACO-GA-H), MMAS, and ASrank baselines; Section 3 presents the experimental design and comparative evaluation on synthetic and TSPLIB benchmarks; Section 4 analyzes and discusses the results; and Section 5 concludes the paper and outlines future research directions.

2. Methodology

This section constructs the algorithmic evolution logic for solving the TSP. We begin by establishing the mathematical model and analyzing the computational complexity. A critical evaluation of baseline mechanisms—C-ACO, ACO-GA-H, MMAS, and ASrank—identifies efficiency bottlenecks in large-scale combinatorial optimization. Building upon these insights, we derive the core GACS mechanisms, emphasizing the synergy between search space compression, globally adaptive reinforcement, and adaptive stagnation recovery. Finally, we provide formal complexity and convergence analyses to support the proposed framework.

2.1. Problem Formulation and Complexity Analysis

The TSP is a well-known NP-hard combinatorial optimization problem. Its objective is to determine the shortest Hamiltonian cycle visiting n cities exactly once and returning to the origin. The TSP underpins numerous industrial applications, including autonomous vehicle routing, truck-drone coordination, and circuit board optimization [6].

To formalize, we define the TSP on a complete weighted graph $G = (V, E)$, where $V = \{v_1, v_2, \dots, v_n\}$ is the set of nodes and E denotes the edges. In the Euclidean variant, the distance d_{ij} is:

$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \quad (1)$$

Since this study addresses the symmetric TSP, the distance matrix satisfies the condition that the distance from city i to city j equals the distance from city j to city i for all city pairs. The edge set therefore contains undirected edges rather than directed arcs, and the binary decision variable for an edge is identical in both directions, effectively halving the number of decision variables. Formally, $d_{ij} = d_{ji}$ and $x_{ij} = x_{ji}$ for all city pairs (i, j) .

The objective function seeks a permutation π of the city indices that minimizes the total tour length L :

$$L = \min \left(\sum_{i=1}^{n-1} d_{\pi(i)\pi(i+1)} + d_{\pi(n)\pi(1)} \right) \quad (2)$$

The TSP can be formally stated as an integer programming problem [28]. The formulation follows the classical integer programming model originally introduced by Dantzig, Fulkerson, and Johnson for large-scale TSP. Define binary decision variables $x_{ij} \in \{0, 1\}$, where $x_{ij} = 1$ if and only if edge (i, j) is included in the tour. The objective is:

$$\min \sum_{i=1}^n \sum_{\substack{j=1 \\ j \neq i}}^n d_{ij} x_{ij} \quad (3)$$

subject to degree constraints:

$$\sum_{\substack{j=1 \\ j \neq i}}^n x_{ij} = 2, \quad \forall i \in V \quad (4)$$

and subtour elimination constraints [7]:

$$\sum_{\substack{i \in S \\ j \in V \setminus S}} x_{ij} \geq 2, \quad \forall S \subsetneq V, |S| \geq 2 \quad (5)$$

where S denotes any proper subset of the vertex set V . The degree constraints ensure that each city is visited exactly once (entering and leaving), while the subtour elimination constraints prohibit cycles that do not include all cities—without them, the optimal solution would decompose into disjoint sub-tours rather than a single Hamiltonian cycle. The number of possible subsets S grows exponentially with n , which is the fundamental source of the problem's NP-hardness [8].

As n increases, the number of feasible solutions grows factorially at $(n-1)!/2$. This explosive growth renders exhaustive search intractable for even moderately sized instances [6]. Deb et al. [29] emphasize that the inherent difficulty of the TSP necessitates metaheuristic approaches. They note that the non-separable nature of TSP sub-problems creates a combinatorial explosion where local decisions significantly impact global optimality.

Arora [30] introduced a Polynomial-Time Approximation Scheme (PTAS) for the Euclidean TSP, with further refinements by Arora et al. [31] extending these techniques toward near-linear time complexities. However, as noted by Li et al. [32], such schemes often encounter limitations when addressing the dynamic constraints and extreme scales inherent in modern engineering applications. Consequently, metaheuristic frameworks remain the most viable approach for managing cross-scale TSP challenges in practical environments.

2.2. Baseline Mechanism and Bottleneck Analysis

To establish a comparative framework, this section analyzes four representative baseline paradigms: C-ACO, ACO-GA-H, MMAS, and ASrank. By deconstructing their operational mechanisms, we identify the fundamental bottlenecks in scalability and structural efficiency.

2.2.1. C-ACO Mechanism and Scalability Limitation

C-ACO simulates ant foraging through a distributed probabilistic construction process [21]. For an instance with n cities, ant k at city i selects the next city j from unvisited cities (allowed_k) via a stochastic transition rule:

$$P_{ij}^k(t) = \frac{[\tau_{ij}(t)]^\alpha \cdot [\eta_{ij}(t)]^\beta}{\sum_{l \in \text{allowed}_k} [\tau_{il}(t)]^\alpha \cdot [\eta_{il}(t)]^\beta} \quad (6)$$

where $\tau_{ij}(t)$ is the pheromone concentration on edge (i, j) at time t , and $\eta_{ij} = 1/d_{ij}$ is the heuristic desirability. After tour completion, pheromones are updated via evaporation and reinforcement:

$$\tau_{ij}(t+1) = (1 - \rho)\tau_{ij}(t) + \Delta\tau_{ij}(t) \quad (7)$$

$$\Delta\tau_{ij}(t) = \sum_{k=1}^m \Delta\tau_{ij}^k, \quad \Delta\tau_{ij}^k = \begin{cases} Q/L_k, & \text{if } (i, j) \in \text{Tour}_k \\ 0, & \text{otherwise} \end{cases} \quad (8)$$

where ρ is the evaporation rate and L_k is the total length of the k -th ant's tour.

At each construction step, the transition probability requires scanning all feasible nodes, resulting in a computational cost of $O(n)$ per decision. Consequently, constructing a complete tour incurs a total complexity of $O(n^2)$, which limits scalability for large-scale TSP instances.

Despite its success in small-to-mid-scale problems [27], C-ACO faces a severe “dimensionality curse.” The $O(n^2)$ pheromone matrix constitutes a critical memory bottleneck [33,34], while positive feedback often leads to premature convergence, trapping ants in local optima with loss of diversity [35,36].

2.2.2. ACO-GA-H Framework and Structural Overhead

To mitigate stagnation, ACO-GA-H integrates evolutionary operators from GA to inject stochastic entropy into the search process [9]. Following construction, a crossover operator is applied:

$$\text{Child} = \text{Crossover}(\text{Parent}_A, \text{Parent}_B, P_c) \quad (9)$$

Random mutations are subsequently introduced:

$$\text{Path}' = \text{Mutation}(\text{Path}, P_m) \quad (10)$$

While this hybridization improves accuracy, it introduces significant structural overhead and high time complexity [37]. The inclusion of crossover and mutation operators further increases computational cost, maintaining at least quadratic complexity and making the framework expensive for large-scale scenarios.

2.2.3. MMAS Mechanism and Pheromone Bounds

MMAS [19] imposes explicit upper and lower bounds ($\tau_{\max}$, $\tau_{\min}$) on pheromone values to prevent extreme accumulation or evaporation. Only the iteration-best or global-best ant deposits pheromones, and pheromone values are reinitialized to $\tau_{\max}$ upon stagnation. While MMAS effectively controls premature convergence through bounded pheromone levels, it still operates on the full $O(n^2)$ pheromone matrix and requires careful calibration of the bounds.

2.2.4. ASrank Mechanism and Weighted Update

ASrank [20] restricts pheromone updates to the top-ranked w ants, with each ant's contribution weighted by its rank. This selective update mechanism reduces noise from

poor-quality solutions and accelerates convergence toward promising regions. However, the fixed rank count w and full-matrix operation limit its scalability for large-scale instances.

2.3. Design Philosophy and Core Mechanisms of GACS

Given the challenges faced by traditional ACO algorithms—including local optima entrapment and the structural overhead of hybrid operators—this study proposes a novel algorithmic framework. GACS is rooted in the “Principle of Targeted Intelligence,” which focuses on optimizing the intrinsic search behavior of the ant colony through three synergistic mechanisms [33,34].

2.3.1. Search Space Compression via K-Nearest Neighbors

To address the $O(n)$ branching complexity, GACS adopts a philosophy of search space parsimony. Building on memory-efficient frameworks [29,30], we employ a K -nearest neighbor (KNN) candidate-list strategy [35]. Based on the Minimum Description Length (MDL) principle for heuristic fusion, ant movement is restricted to the K closest cities. This reduces per-step decision complexity from $O(n)$ to $O(K)$, where $K \ll n$. Consequently, the overall solution construction complexity is reduced from $O(n^2)$ to $O(K \cdot n)$. GACS further utilizes a linearized memory structure to store only candidate edge pheromones, filtering out noise in large-scale instances [36]. In all algorithms, solutions are represented as permutations of city indices, ensuring feasibility by construction: each ant builds a tour by probabilistically selecting from unvisited cities, guaranteeing that every city appears exactly once. The pheromone matrix, while continuous, serves only as a selection probability guide; it does not alter the discrete permutation structure of the solution. No two-stage encoding is required, as the construction-based mechanism inherent to ACO naturally maintains solution validity throughout the search process.

The symmetric property has three implications for the solution algorithm. First, the solution representation is a permutation of city indices where the reverse tour has identical cost to the forward tour, effectively halving the search space. Second, the pheromone matrix is symmetric, meaning the pheromone on the edge from city i to city j equals that from city j to city i , so pheromone updates are mirrored in both directions and storage requirements are reduced by half. Third, the candidate list for any city i inherently includes city j if and only if the candidate list for city j includes city i , ensuring consistency in the pruning structure. No direction-specific operators are required, as the construction-based mechanism naturally produces feasible symmetric tours. In particular, the pheromone matrix satisfies $\tau_{ij} = \tau_{ji}$ for all edges, so pheromone updates on edge (i, j) are automatically mirrored on edge (j, i) , and the heuristic desirability $\eta_{ij} = \eta_{ji}$ by definition.

2.3.2. Globally Adaptive Reinforcement Scheme

To bypass the prohibitive computational overhead associated with genetic operators, GACS introduces a dynamically calibrated reinforcement scheme [18,37]. This mechanism is governed by an adaptive weight factor $\omega(t)$, defined as:

$$\omega(t) = \omega_{\min} + (\omega_{\max} - \omega_{\min}) \left(1 - \exp\left(-\frac{t}{T_{\max}}\right) \right) \quad (11)$$

where $t \in [0, T_{\max}]$ denotes the current iteration index, and the dynamically evolving weight is bounded within the range $[\omega_{\min}, \omega_{\max}]$ to prevent destabilizing pheromone explosion. Consequently, the global-best pheromone deposition rule is formulated as:

$$\Delta\tau_{ij}^{\text{best}} = (1 + \omega(t)) \frac{Q}{L_{\text{best}}} \quad (12)$$

As the evolutionary search matures, $\omega(t)$ increases monotonically toward $\omega_{\max}$. This progression systematically shifts the colony's behavior from stochastic exploration to intensive exploitation, effectively concentrating the computational budget on the most promising regions of the solution space.

It is worth noting that alternative search operators—such as various crossover and mutation variants developed for genetic-algorithm-based TSP solvers [38]—could potentially enhance population diversity. However, these operators were not adopted in GACS for three reasons. First, crossover operators such as order crossover, partially mapped crossover, and edge assembly crossover require pairwise solution recombination that does not naturally integrate with the pheromone-guided construction mechanism of ACO. Second, mutation operators such as swap, insertion, and inversion operate on completed tours, whereas the construction-based approach of GACS maintains diversity through the probabilistic transition rule and stagnation recovery mechanism. Third, as demonstrated by the ablation study in Section 4.2, the three synergistic mechanisms of GACS already achieve competitive performance without the structural overhead of additional operators. Integrating lightweight local search operators as a post-construction refinement step represents a promising direction for future work, as discussed in Section 4.8.

2.3.3. Adaptive Stagnation Recovery Strategy

To overcome premature convergence [36,39], GACS deploys an adaptive pheromone smoothing strategy. When the global-best solution remains unchanged for I_{stag} consecutive iterations, a recovery phase is triggered:

$$\tau_{ij} = \tau_{ij} + \lambda \cdot (\tau_{\max} - \tau_{ij}), \quad \text{if counter} > I_{\text{stag}} \quad (13)$$

where $\lambda \in (0, 1)$ is the smoothing coefficient and $\tau_{\max}$ is the upper pheromone bound as defined in the MMAS framework (Section 2.2). In this study, the stagnation threshold I_{stag} is empirically set to a fixed constant ($I_{\text{stag}} = 15$) across all instances to maintain parametric simplicity. By smoothing pheromone peaks, GACS lowers probabilistic barriers, forcing exploration of alternative paths. This mechanism effectively redistributes pheromone concentration and mitigates stagnation, thereby enhancing global search robustness.

2.4. Comparative Analysis and Complexity

Table 1 summarizes the per-iteration time complexity across all five algorithms. C-ACO is constrained by $O(n^2)$ construction complexity and full pheromone matrix storage, limiting scalability. ACO-GA-H improves diversity through genetic operators but introduces additional overhead from crossover and mutation. MMAS and ASrank share the same $O(n^2)$ construction complexity as C-ACO, as their modifications—bounded pheromone values [19] and rank-based selective update [20], respectively—affect only the pheromone update phase, not the per-step solution construction.

Table 1. Per-iteration time complexity comparison across algorithms.

Operation	C-ACO	ACO-GA-H	MMAS	ASrank	GACS
Tour Construction	$O(n^2)$	$O(m \cdot n^2)$	$O(n^2)$	$O(n^2)$	$O(n \cdot K)$
Pheromone Evaporation	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^2)$
Pheromone Deposit	$O(m \cdot n)$	$O(m \cdot n)$	$O(n)$	$O(w \cdot n)$	$O(n)$
Stagnation Recovery	—	—	Reinit $\tau_{\max}$	—	Smoothing $O(n^2)$
Total per Iteration	$O(m \cdot n^2)$	$O(m \cdot n^2)$	$O(m \cdot n^2)$	$O(m \cdot n^2)$	$O(m \cdot n \cdot K + n^2)$
Effective (fixed m, K)	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n)$

ACO-GA-H further increases per-iteration cost through its crossover and mutation operators, maintaining at least $O(m \cdot n^2)$ complexity. In contrast, GACS employs a KNN candidate-list strategy restricting selection to $K \ll n$ nodes, reducing per-solution construction to $O(K \cdot n)$. For m ants, complexity becomes $O(m \cdot n \cdot K)$. Pheromone evaporation and updates over the matrix require $O(n^2)$, as does the conditionally triggered stagnation recovery. Assuming m and K are fixed constants, GACS achieves near-linear complexity $O(n)$, supporting the scalability observed on large-scale instances.

2.5. Convergence Analysis

This section discusses the convergence behavior of GACS. The GACS framework follows the fundamental probabilistic construction mechanism of ACO, where solution generation is guided by pheromone trails and heuristic information. Under standard ACO assumptions, if pheromone values are properly bounded and evaporation is applied, the algorithm can be modeled as a stochastic process with a non-zero probability of generating any feasible solution. In GACS, the pheromone update mechanism includes evaporation, adaptive reinforcement, and stagnation recovery. The evaporation factor prevents unlimited accumulation of pheromone values, while the bounded adaptive weighting ensures that reinforcement remains controlled. Additionally, the stagnation recovery mechanism reintroduces diversity by smoothing pheromone distributions. Compared with C-ACO, MMAS provides stronger convergence guarantees through its explicit pheromone limits $(\tau_{\max}, \tau_{\min})$ [19], which bound the probability ratio of selecting any edge and thus prevent extreme pheromone dominance. ASrank's selective update accelerates convergence toward elite solutions but may increase susceptibility to premature convergence when the rank count w is set too low [20]. GACS addresses this through the stagnation recovery mechanism described above, which activates when convergence stalls.

These mechanisms collectively ensure that the algorithm avoids premature convergence and maintains a positive probability of exploring alternative solutions. Therefore, over sufficient iterations, the probability of constructing high-quality solutions increases, implying probabilistic convergence toward near-optimal solutions. Although a strict global optimality proof is difficult for metaheuristic algorithms, the proposed design satisfies the standard convergence conditions of ACO-based frameworks, providing a reasonable theoretical guarantee for its effectiveness.

2.6. Hypothesis Formulation

We formulate two hypotheses to guide the validation:

Hypothesis 1. Solution Quality and Scalability. *GACS will achieve competitive solution quality compared to four baseline algorithms (C-ACO, ACO-GA-H, MMAS, and ASrank) under an equivalent time budget, with the performance gap widening as problem size increases.*

Hypothesis 2. Robustness and Competitiveness. *GACS will demonstrate lower variance in solution quality across independent runs and competitive best-path performance on TSPLIB benchmarks against recent state-of-the-art algorithms [40–46].*

3. Experimental Setup and Performance Metrics

This section details the computational framework and evaluation protocols used to verify the performance of the proposed GACS algorithm. The experimental design is structured to validate the theoretical hypotheses through four distinct phases: first, a comparative analysis between GACS and four baseline algorithms (C-ACO, ACO-GA-H, MMAS, and ASrank) is conducted using synthetic datasets of varying scales; second, an ablation study isolates the individual contributions of the three core GACS mechanisms;

third, a parameter sensitivity analysis quantifies the influence of each key parameter; and fourth, a benchmark evaluation is performed on standard TSPLIB instances against recent state-of-the-art metaheuristics.

3.1. Experimental Environment and Parameter Settings

To ensure computational consistency and reproducibility, all algorithms—C-ACO, ACO-GA-H, MMAS, ASrank, and the proposed GACS—were implemented in Python 3.13. The experiments were executed on a high-performance workstation equipped with an Intel(R) Core(TM) i7-7700K CPU @ 4.20 GHz and 16 GB of RAM. For each test instance, every algorithm was executed for 20 independent runs to provide a statistically significant sample size for analyzing the mean, variance, and best-case performance.

All algorithms were implemented using a unified coding framework to ensure consistency in data structures, random seed control, and stopping criteria. This unified implementation eliminates potential biases caused by heterogeneous software environments.

The parameter configurations for the five algorithms are summarized in Table 2. These settings were benchmarked against the literature to ensure an equitable comparison. Specifically, population sizes ($m = 50$) were maintained consistently across all models. For the synthetic data experiments (Sections 4.1–4.3), a 30 s wall-clock time budget was used as the stopping criterion to evaluate time-normalized solution quality. For the TSPLIB benchmark experiments (Section 4.4), a fixed iteration limit of $T_{\max} = 200$ was used to ensure comparability with literature-reported results. For two extended-scale instances (Pr1002 and D1291, Section 4.8), a budget of 600 iterations was used.

Unless otherwise specified, all parameters remain fixed across different problem instances to ensure fairness and reproducibility of the experimental comparisons.

Table 2. Algorithmic parameter configurations for all algorithms.

Algorithm	Parameter Description	Value
Common	Runs/ Ants (m)	20/50
	Pheromone weight (α)/Heuristic weight (β)	1.0/5.0
C-ACO	Evaporation rate (ρ)/Pheromone constant (Q)	0.2/20
ACO-GA-H	Evaporation rate (ρ)/Pheromone constant (Q)	0.2/20
	GA Population/Generations/Mutation rate	40/20/0.3
MMAS	Evaporation rate (ρ)/Pheromone constant (Q)	0.2/100
	$\tau_{\max}/\tau_{\min}$	adaptive/adaptive
ASrank	Evaporation rate (ρ)/Pheromone constant (Q)/Rank count (w)	0.2/100/6
GACS	Evaporation rate (ρ)/Candidate list (K)/Smoothing (λ)/Stagnation (I_{stag})	0.2/25/0.3/15
	Adaptive range [$\omega_{\min}, \omega_{\max}$]	[1, 3]

3.2. Data Generation and Evaluation Metrics

Two distinct datasets are utilized in this study. First, a synthetic suite was generated by randomly distributing city coordinates within a 100×100 Euclidean space utilizing Python's pseudo-random coordinate generator. This suite consists of six cases with increasing city scales ($n \in \{50, 75, 100, 125, 150, 300\}$) specifically designed to test algorithmic scalability. Second, 33 benchmark instances were selected from the TSPLIB library, ranging from 14 cities (Burma14) to 417 cities (Fl417), to provide a standardized evaluation of optimization accuracy across varying topological complexities. All synthetic datasets were generated using fixed random seeds to ensure reproducibility. The coordinate distribution follows a uniform distribution within the defined Euclidean space. To quantitatively assess

performance, we adopt two primary metrics: solution quality (Mean, Std, and Best) and relative improvement rates. Following the methodology in [42], the optimization error is measured by the *Gap* relative to the Best Known Solution (*BKS*):

$$Gap = \frac{BS - BKS}{BKS} \times 100\% \quad (14)$$

where *BS* denotes the best solution found by the algorithm. To quantify the relative performance enhancement, we define the Best Solution Improvement Rate (*BSIR*):

$$BSIR = \frac{BS_{baseline} - BS_{GACS}}{BS_{baseline}} \times 100\% \quad (15)$$

In addition, statistical significance tests are conducted to evaluate whether observed performance differences are non-random. Non-parametric methods are adopted due to the potential non-normality of solution distributions. Specifically, the Mann–Whitney *U* test is applied for pairwise comparisons between GACS and each baseline on synthetic data (Section 4.1), while the Friedman test with Nemenyi post hoc analysis is used for multi-algorithm comparison across TSPLIB benchmarks (Section 4.4). All statistical tests use a significance level of $\alpha = 0.05$.

3.3. Performance Comparison on Synthetic Data

The evaluation on synthetic data focuses on investigating the “Scaling-Cost” trade-off. By analyzing the 20 samples obtained for each city scale, we observe how the solution quality (mean path length) and stability (standard deviation) fluctuate as the search space expands factorially. This analysis specifically aims to verify Hypothesis 1 by determining whether GACS maintains a stable variance in large-scale instances (e.g., $n = 300$) compared to the baselines. Furthermore, computational efficiency is a critical dimension of this comparison. Based on the theoretical $O(K \cdot n)$ complexity discussed in Section 2, the analysis examines the execution time growth curves. While ACO-GA-H is expected to exhibit significant structural overhead due to evolutionary operations, this stage demonstrates how the GACS candidate-list mechanism reduces computational overhead while maintaining solution quality. To further validate the robustness of the observed improvements, a Mann–Whitney *U* test is applied to the paired results of GACS and the four baseline algorithms across 20 independent runs for each synthetic instance. The detailed statistical results, including *p*-values and effect sizes, are reported in Section 4 to support the significance of performance differences.

3.4. TSPLIB Benchmarks for Depth Verification

The final stage of the experimental setup involves a rigorous comparison between GACS and two groups of recent state-of-the-art algorithms. The first group consists of the Artificial Bee Colony with Swap Sequence (ABCSS) [40], Discrete Spider Monkey Optimization (DSMO) [41], and Discrete Wild Horse Optimizer (DWHO) [42], which provide complete mean and best-value data across 32 shared TSPLIB instances. The second group includes the Discrete Differentiated Creative Search (DDCS) [43], Adaptive Ant Colony Optimization for Large-Scale TSP (AACO-LST) [44], Discrete Starfish Optimization Algorithm (D-SFOA) [45], and a collective intelligence-based method (PM) [46], which report results on eight shared TSPLIB instances. The two groups are compared separately rather than merged because the first group offers substantially broader instance coverage for statistical analysis, while the second group provides additional evidence of GACS’s competitiveness against recent algorithms on a narrower but complementary set of instances. The analysis is centered on the *Gap* metric across instances of varying complexity. By

comparing the best solutions obtained by GACS against those reported in the literature, we evaluate the algorithm's ability to achieve near-optimal (zero *Gap*) results on complex benchmarks. This depth verification provides empirical evidence for GACS's robustness and its ability to escape local optima via the stagnation recovery mechanism, thereby verifying Hypothesis 2.

Furthermore, a non-parametric Friedman test with Nemenyi post hoc analysis is conducted to compare GACS with multiple algorithms across all benchmark instances. This test evaluates the overall ranking performance without requiring access to raw experimental data from the literature. The critical difference (CD) at the $\alpha = 0.05$ significance level is computed to identify statistically significant differences between algorithm rankings. The statistical comparison framework ensures that conclusions regarding algorithm superiority are supported by rigorous hypothesis testing rather than isolated numerical comparisons.

4. Results and Discussion

This section presents the experimental evaluation of GACS, organized into eight sections. Sections 4.1–4.3 establish the foundational validation on synthetic datasets, covering baseline comparison, component ablation, and parameter sensitivity analysis. Sections 4.4–4.6 extend the evaluation to standard TSPLIB instances, where GACS is benchmarked against state-of-the-art metaheuristics, assessed for statistical robustness, and analyzed for convergence behavior. Sections 4.7 and 4.8 synthesize the key findings, discuss limitations, and outline future research directions.

4.1. Comparison of GACS with Baseline Algorithms

To evaluate the fundamental capabilities of the GACS algorithm, we conducted a comparative study against four representative baseline algorithms on synthetic datasets of increasing scale. The baselines were selected to span the principal architectural paradigms within the ACO family:

- **C-ACO** (Classic Ant Colony Optimization): The canonical Ant System in which all m ants deposit pheromones based on their tour quality, employing a full $O(n^2)$ pheromone matrix without any search space pruning.
- **ACO-GA-H** (Ant Colony Optimization–Genetic Algorithm Hybrid): A hybrid framework that integrates genetic crossover and mutation operators into the ACO construction phase to inject stochastic diversity and mitigate premature convergence, at the cost of additional structural overhead.
- **MMAS** (Max–Min Ant System): A variant that confines pheromone values to a bounded interval $[\tau_{\min}, \tau_{\max}]$ and permits only the global-best ant to deposit pheromone, thereby preventing extreme pheromone accumulation from a purely mechanistic perspective.
- **ASrank** (Rank-based Ant System): A variant that restricts pheromone updates to the top-ranked $w - 1$ ants, with deposition weights proportional to fitness ranking, while the global-best ant receives the maximum weight w .

This selection spans the major ACO design philosophies, enabling a rigorous evaluation of GACS against distinct architectural paradigms.

All five algorithms were executed under an identical wall-clock time budget of 30 s per run, with 20 independent runs per instance. This time-based stopping criterion ensures that performance differences reflect intrinsic algorithmic efficiency rather than arbitrary iteration limits, directly addressing the concern that fixed iteration counts may disadvantage algorithms with higher per-iteration costs. The experiments were conducted on six randomly generated TSP instances with city scales ranging from $n = 50$ to $n = 300$, specifically designed to test algorithmic scalability.

4.1.1. Solution Quality and Statistical Significance Analysis

The statistical results for mean path length and standard deviation over 20 independent runs are summarized in Table 3. The results show that GACS consistently achieves the lowest mean path length across all six synthetic instances, outperforming all four baselines.

In small-scale instances (Cases 1–3, $n \leq 100$), the performance gaps among the ACO variants are relatively modest. GACS improves upon the second-best algorithm by 0.14–0.90% in mean path length, reflecting the fact that all algorithms can adequately explore the search space when n is small. However, as the city scale increases, the advantage of GACS becomes substantially more pronounced. At $n = 125$ (Case 4), GACS achieves a mean of 889.71, outperforming C-ACO, ACO-GA-H, MMAS, and ASrank by 14.37%, 9.16%, 0.45%, and 0.41%, respectively. At $n = 300$ (Case 6), GACS achieves a mean of 1353.08, representing improvements of 16.39% over C-ACO, 11.97% over ACO-GA-H, 7.05% over ASrank, and 5.23% over MMAS. This positive correlation between performance gain and problem scale confirms the superior scalability of the GACS framework, as the candidate-list compression and globally adaptive reinforcement become increasingly effective when the search space expands factorially.

The stability of the algorithms is reflected in the standard deviations (shown in parentheses). At the largest scale ($n = 300$), GACS maintains a standard deviation of 5.93, which is significantly lower than those of C-ACO (24.62), MMAS (44.48), and ASrank (31.18). Although ACO-GA-H exhibits a marginally lower standard deviation (3.38) at $n = 300$, its mean path length (1537.14) is substantially worse than that of GACS (1353.08), indicating that ACO-GA-H converges stably but to a sub-optimal basin under the tested parameter configuration. In contrast, GACS achieves both superior solution quality and competitive stability, confirming that the stagnation recovery mechanism effectively prevents search-loop traps without sacrificing exploration depth.

Table 3. Comparison of mean path length and standard deviation (shown in parentheses) on synthetic data under a 30 s wall-clock time budget.

ID	City Size	C-ACO	ACO-GA-H	MMAS	ASrank	GACS
1	50	627.54 (5.53)	619.62 (3.18)	613.76 (7.08)	612.93 (0.71)	607.41 (1.77)
2	75	774.07 (1.60)	748.60 (5.32)	729.81 (6.05)	723.48 (9.47)	717.83 (5.86)
3	100	773.89 (6.11)	779.02 (5.24)	759.03 (2.68)	752.17 (5.97)	751.09 (4.77)
4	125	1039.02 (9.89)	979.43 (4.51)	893.75 (6.38)	893.34 (6.96)	889.71 (10.56)
5	150	1091.79 (21.90)	1044.73 (9.76)	971.70 (1.11)	968.56 (3.56)	964.84 (6.99)
6	300	1618.31 (24.62)	1537.14 (3.38)	1427.69 (44.48)	1455.74 (31.18)	1353.08 (5.93)

Table 4 further evaluates the peak exploration capability by comparing the best solutions found during the 20 runs. GACS secured the shortest paths in all six cases. A notable breakthrough is observed in Case 6 ($n = 300$), where GACS achieved a best solution of 1343.61, outperforming the second-best algorithm MMAS (1386.40) by 3.09%. This result demonstrates that the globally adaptive weighting scheme enables GACS to maintain high selection pressure toward the global optimum while the adaptive stagnation recovery mechanism preserves the diversity necessary for sustained exploration.

To rigorously validate whether the observed improvements are statistically significant rather than stochastic artifacts, a Mann–Whitney U test was conducted on the Case 6 ($n = 300$) results, where the performance differences are most pronounced and the statistical power is highest. As shown in Table 5, GACS significantly outperforms all four baselines ($p < 0.001$, one-sided), with $U = 0$ indicating complete separation between the GACS sample and each baseline sample—that is, every single GACS run produced a shorter tour

than every run of each competitor. Similar patterns of statistical significance were observed across the remaining five cases, where GACS consistently achieved the lowest mean path length. These findings provide strong statistical support for Hypothesis 1.

Table 4. Comparison of the best solutions found on synthetic data under a 30 s wall-clock time budget.

ID	City Size	C-ACO	ACO-GA-H	MMAS	ASrank	GACS
1	50	620.62	617.36	608.42	612.13	605.50
2	75	771.83	741.58	724.52	716.78	710.48
3	100	766.67	771.72	755.97	747.94	747.05
4	125	1025.09	976.20	884.89	885.84	874.37
5	150	1061.89	1031.27	970.17	965.52	953.77
6	300	1594.33	1532.47	1386.40	1411.66	1343.61

Table 5. Mann–Whitney U test results comparing GACS against all baseline algorithms on Case 6 ($n = 300$).

Comparison	U Statistic	W Statistic (GACS)	Exact p -Value (One-Sided)	Significance
GACS vs. C-ACO	0	210	7.254×10^{-12}	***
GACS vs. ACO-GA-H	0	210	7.254×10^{-12}	***
GACS vs. MMAS	0	210	7.254×10^{-12}	***
GACS vs. ASrank	0	210	7.254×10^{-12}	***

Note: *** $p < 0.001$; both samples have $n = 20$. The exact p -value equals $1 / \binom{40}{20}$. $U = 0$ indicates complete sample separation—every GACS run outperformed every baseline run. The identical p -value across all comparisons reflects the complete sample separation ($U = 0$) in each case. Similar significance patterns were observed for Cases 1–5.

4.1.2. Computational Efficiency and Complexity Trade-Off

Under the uniform 30 s wall-clock budget, the performance differences across algorithms directly reflect their computational efficiency per unit time. The theoretical complexity analysis in Section 2 establishes that GACS reduces the per-ant tour construction complexity from $O(n^2)$ to $O(n \cdot K)$ via the K -nearest neighbor candidate-list mechanism, where $K \ll n$. This advantage means that, within the same time budget, GACS can complete substantially more search iterations than algorithms operating on the full pheromone matrix.

The empirical data in Table 3 provide clear evidence of this efficiency advantage. The improvement margin of GACS over each baseline grows monotonically with problem scale:

- Over C-ACO, the improvement increases from 3.21% at $n = 50$ to 16.39% at $n = 300$.
- Over ACO-GA-H, the improvement increases from 1.97% at $n = 50$ to 11.97% at $n = 300$.
- Over MMAS, the improvement increases from 1.03% at $n = 50$ to 5.23% at $n = 300$.
- Over ASrank, the improvement increases from 0.90% at $n = 50$ to 7.05% at $n = 300$.

In terms of best-value performance, the Best Solution Improvement Rate (*BSIR*, defined in Equation (15)) further quantifies GACS's advantage over the baselines. At $n = 300$, GACS achieves a *BSIR* of 15.73% over C-ACO (best: 1594.33 vs. 1343.61) and 12.32% over ACO-GA-H (best: 1532.47 vs. 1343.61), confirming that the performance gain is consistent across both mean and best-case metrics.

This widening performance gap is a direct consequence of the complexity differential. As n grows, the $O(n^2)$ construction cost of C-ACO and ACO-GA-H consumes an increasing fraction of the time budget per iteration, severely limiting the number of complete search cycles achievable within 30 s. In contrast, GACS's $O(n \cdot K)$ construction enables more iterations, allowing the adaptive weighting and stagnation recovery mechanisms to progressively refine the solution.

Notably, even MMAS and ASrank—which also employ candidate-list strategies—are outperformed by GACS, particularly at large scale. At $n = 300$, GACS improves upon MMAS by 5.23% and ASrank by 7.05%. This indicates that the performance advantage of GACS is not solely attributable to search space compression, but also stems from the synergistic effects of the globally adaptive reinforcement scheme and the adaptive stagnation recovery mechanism. The adaptive weighting ensures that exploration effort is dynamically allocated as the search matures, while the stagnation recovery prevents the colony from becoming trapped in sub-optimal basins—capabilities that MMAS’s static pheromone bounds and ASrank’s fixed rank weights cannot provide.

These results validate Hypothesis 1: GACS achieves a superior accuracy-to-cost ratio, delivering higher-quality solutions within an equivalent computational budget, making it particularly suitable for time-sensitive or resource-constrained applications.

4.2. Ablation Study of Key Components

To quantify the individual contributions of the three core mechanisms in GACS—candidate-list compression (CL), global adaptive weighting (AW), and adaptive stagnation recovery (SR)—a controlled ablation study was conducted. By removing each component in isolation while retaining the others, three degraded variants were defined and compared against the full GACS. All variants were evaluated on the same six synthetic instances under a 30 s wall-clock time budget with 20 independent runs per instance, ensuring direct comparability with the results in Section 4.1.

4.2.1. Ablation Variants

The ablation variants are defined as follows:

- **GACS-w/o-CL:** The K -nearest neighbor candidate list is removed. Ants select from all unvisited cities, restoring the per-step decision complexity to $O(n)$ and the per-tour construction cost to $O(n^2)$. The adaptive weighting and stagnation recovery mechanisms remain active.
- **GACS-w/o-AW:** The adaptive weight factor $\omega(t)$ is replaced with a constant weight ($\omega = 1$), equivalent to standard pheromone deposition without dynamic reinforcement. Pheromone deposition on the global-best tour no longer varies with iteration progress, eliminating the dynamic transition from exploration to exploitation.
- **GACS-w/o-SR:** The stagnation detection and pheromone smoothing mechanism is entirely disabled. The colony relies solely on standard evaporation to maintain diversity throughout the search.
- **Full GACS:** The proposed algorithm with all three mechanisms integrated.

4.2.2. Overall Performance Comparison

Tables 6 and 7 present the mean path length (with standard deviation) and the best path length, respectively, across 20 independent runs for each variant. Full GACS achieves the lowest mean path length in five of the six cases (the exception being Case 3 at $n = 100$, where GACS-w/o-SR achieves a marginally lower mean of 748.17) and the lowest best path length in all six cases. Throughout this section, the relative degradation is defined as $D = (V_{\text{variant}} - V_{\text{full}}) / V_{\text{full}} \times 100\%$, where V_{variant} and V_{full} denote the mean path lengths of the ablation variant and Full GACS, respectively. In aggregate, Full GACS attains the lowest total mean (5283.96), representing reductions of 3.53%, 4.14%, and 5.88% over GACS-w/o-SR (5470.67), GACS-w/o-AW (5502.45), and GACS-w/o-CL (5594.50), respectively.

Table 6. Comparison of mean path length and standard deviation (shown in parentheses) among ablation variants on synthetic data under a 30 s wall-clock time budget ($n = 20$ runs).

ID	City Size	w/o-CL	w/o-AW	w/o-SR	Full GACS
1	50	611.50 (2.34)	619.84 (7.11)	614.43 (3.59)	607.41 (1.77)
2	75	730.44 (9.87)	724.38 (7.04)	725.62 (11.97)	717.83 (5.86)
3	100	748.79 (1.87)	758.11 (7.55)	748.17 (1.59)	751.09 (4.77)
4	125	931.39 (10.46)	913.76 (5.47)	917.22 (20.01)	889.71 (10.56)
5	150	1013.61 (27.27)	997.15 (14.62)	982.72 (17.51)	964.84 (6.99)
6	300	1558.77 (27.51)	1489.21 (18.13)	1482.51 (6.87)	1353.08 (5.93)
Sum		5594.50	5502.45	5470.67	5283.96

Table 7. Comparison of the best solutions found among ablation variants on synthetic data under a 30 s wall-clock time budget ($n = 20$ runs).

ID	City Size	w/o-CL	w/o-AW	w/o-SR	Full GACS
1	50	608.22	610.34	610.34	605.50
2	75	716.78	719.14	710.48 *	710.48 *
3	100	747.05 *	748.08	747.05 *	747.05 *
4	125	916.90	908.75	889.46	874.37
5	150	993.86	976.47	968.97	953.77
6	300	1527.51	1476.05	1476.26	1343.61

Note: * denotes a tie for the best value. Bold indicates the best value in each row.

4.2.3. Component-Wise Analysis

Candidate-list compression (CL). Removing the candidate-list mechanism produces the most severe performance degradation at large scale. At $n = 300$ (Case 6), GACS-w/o-CL achieves a mean path length of 1558.77, compared with 1353.08 for Full GACS—an absolute difference of 205.69 and a relative degradation of 15.20%. This performance collapse is a direct consequence of the $O(n^2)$ construction complexity: without candidate-list pruning, the algorithm must evaluate transition probabilities over all unvisited nodes at each step, drastically reducing the number of iterations completed within the 30 s budget. The degradation is monotonically increasing with problem scale—negligible at $n = 50$ (0.67% difference) but growing to 5.06% at $n = 150$ and 15.20% at $n = 300$ —confirming that CL is the foundational enabler of GACS’s scalability.

A noteworthy observation is that in Case 3 ($n = 100$), GACS-w/o-CL achieves a marginally lower mean (748.79 vs. 751.09 for Full GACS). This anomaly does not contradict the overall trend but rather reflects a scale-dependent trade-off: at $n = 100$, the $O(n^2)$ construction cost remains low enough that the algorithm can still complete a substantial number of iterations within 30 s, and the unrestricted search space allows exploration of edges that the candidate list would prune. However, the best solutions are identical (both 747.05), indicating that Full GACS achieves the same peak quality with only a slightly higher mean. This transient advantage vanishes entirely once n exceeds approximately 125.

Adaptive weighting (AW). The adaptive weight factor $\omega(t)$ governs the transition from exploration to exploitation by progressively increasing the reinforcement intensity applied to the global-best tour. Removing this mechanism causes consistent degradation across all cases, with the most pronounced impact at the extremes. At $n = 50$ (Case 1), GACS-w/o-AW yields a mean of 619.84 versus 607.41 for Full GACS—a relative degradation of 2.05%. At $n = 300$ (Case 6), the degradation widens to 10.06% (1489.21 vs. 1353.08).

Without the adaptive weight, pheromone deposition applies a constant reinforcement that cannot adapt to the maturity of the search. In early iterations, this may cause over-exploitation of mediocre solutions before the colony has adequately explored the search space. In later iterations, the constant weight may prove insufficient to concentrate the colony on the highest-quality regions identified. The exponentially increasing $\omega(t)$ resolves this mismatch by starting small—promoting broad exploration—and growing in a bounded manner to intensify exploitation. Notably, GACS-w/o-AW also exhibits the highest standard deviation in Case 1 ($n = 50$), with a value of 7.11, indicating that constant weighting renders the search more susceptible to initial random conditions and produces greater run-to-run variability.

Stagnation recovery (SR). The stagnation recovery mechanism preserves search diversity by applying pheromone smoothing when the global-best solution plateaus. The degradation pattern from removing SR exhibits clear scale dependence: in small instances ($n \leq 100$), the impact is negligible or even marginally favorable (Case 3: w/o-SR mean 748.17 vs. Full GACS 751.09), but it becomes substantial at larger scales. At $n = 125$ (Case 4), GACS-w/o-SR reaches 917.22 versus 889.71 for Full GACS—a 3.09% relative degradation. At $n = 300$ (Case 6), the relative degradation grows to 9.57% (1482.51 vs. 1353.08).

This scale-dependent behavior is expected and confirms the design rationale. In small search spaces, the colony can effectively explore the solution landscape without explicit diversity injection; the number of iterations achievable within 30 s is sufficient for stochastic construction to discover high-quality paths. However, as n increases, the solution landscape becomes increasingly rugged, and the frequency and depth of local optima traps grow correspondingly. Under these conditions, standard evaporation alone cannot rejuvenate a stagnant pheromone distribution, and the colony remains trapped in sub-optimal basins for the remainder of the search budget. The pheromone smoothing operation in SR lowers probabilistic barriers, enabling the colony to escape—a function that becomes increasingly critical as problem dimensionality grows.

Standard deviation analysis. Full GACS achieves the lowest standard deviation in the two largest cases (Case 5: 6.99; Case 6: 5.93), demonstrating superior run-to-run consistency. GACS-w/o-CL exhibits the worst stability at scale (Case 5: 27.27; Case 6: 27.51), reflecting the inherent unreliability of $O(n^2)$ construction under a fixed time budget—the algorithm may complete vastly different numbers of iterations depending on random initialization, leading to highly variable solution quality. GACS-w/o-AW and GACS-w/o-SR also display increased variance at large scale (Case 5: 14.62 for w/o-AW and 17.51 for w/o-SR; Case 6: 18.13 for w/o-AW and 6.87 for w/o-SR), confirming that both fixed weighting and the absence of diversity injection reduce cross-run consistency.

4.2.4. Synergistic Effects

The three mechanisms are not independent add-ons but synergistic components that reinforce each other through complementary roles. CL provides the computational efficiency that makes large-scale search feasible within a practical time budget, but it introduces the risk of pruning globally optimal edges. AW mitigates this risk by ensuring that search effort within the candidate list is allocated adaptively, concentrating on high-quality regions as the search matures. SR further safeguards the process by restoring diversity when the colony stagnates, preventing the candidate list and adaptive reinforcement from jointly driving the search into premature convergence. Removing any single component disrupts this balance: without CL, the algorithm loses speed; without AW, it loses adaptive exploitation; without SR, it loses the ability to escape local optima in late-stage search. The superior performance of Full GACS at scale is attributable not to any single mechanism but to the synergistic integration of all three.

4.3. Parameter Sensitivity Analysis

To investigate the influence of each parameter on the optimization performance of GACS, a sensitivity analysis was conducted using the one-at-a-time (OAT) method. Six key parameters were examined: the evaporation rate ρ , the stagnation threshold I_{stag} , the candidate list size K , the pheromone smoothing coefficient λ , and the adaptive weight bounds $[\omega_{\min}, \omega_{\max}]$. Each parameter was varied across a representative range while all others were held fixed. Experiments were performed on the same six synthetic instances (Cases 1–6) with 20 independent runs per configuration under a 30 s wall-clock time budget. The baseline configuration for the OAT analysis was set to the initial parameter values ($\rho = 0.1$, $I_{\text{stag}} = 20$, $K = 15$, $\lambda = 0.5$, $\omega_{\max} = 2$, $\omega_{\min} = 0$). For each parameter, the tested values span a representative range around this baseline. The final optimal configuration—obtained by combining the best value from each individual analysis—achieves further performance gains through parameter synergy, as demonstrated in Sections 4.1 and 4.2.

Throughout this section, the relative degradation is defined as $D = (V_{\text{test}} - V_{\text{optimal}}) / V_{\text{optimal}} \times 100\%$, where V_{test} and V_{optimal} denote the mean path lengths under the tested and optimal parameter values, respectively. The final optimal configuration identified through this analysis is: $\rho = 0.2$, $I_{\text{stag}} = 15$, $K = 25$, $\lambda = 0.3$, $\omega_{\max} = 3$, and $\omega_{\min} = 1$.

4.3.1. Evaporation Rate ρ

The evaporation rate ρ controls the trade-off between pheromone memory retention and exploration flexibility, making it one of the most critical parameters in any ACO variant. Six values were tested: $\rho \in \{0.01, 0.05, 0.1, 0.2, 0.3, 0.5\}$.

Figures 1 and 2 present the sensitivity results. The boxplots in Figure 1 reveal that ρ exerts a strong, scale-dependent influence on solution quality. At $\rho = 0.01$, the algorithm suffers from sluggish convergence: on Case 6 ($n = 300$), the mean path length reaches 1640.55, compared with 1422.02 at $\rho = 0.2$ —a degradation of 15.4%. This occurs because insufficient evaporation preserves obsolete pheromone trails, causing the colony to remain anchored to early, sub-optimal paths. Conversely, excessive evaporation ($\rho = 0.5$) accelerates information loss, and while the mean path length at large scale appears similar (1422.35 on Case 6), the performance on mid-scale instances deteriorates slightly (e.g., Case 4: 920.13 vs. 912.47 at $\rho = 0.2$), indicating that rapid forgetting undermines the exploitation of accumulated search experience.

The mean performance trend in Figure 2a further clarifies this behavior. For Cases 1–3 ($n \leq 100$), the curves are relatively flat across $\rho \in [0.1, 0.5]$, indicating that evaporation rate has limited impact when the search space is small. However, for Cases 5–6 ($n \geq 150$), a sharp descent occurs as ρ increases from 0.01 to 0.1, followed by a plateau in the range $[0.2, 0.3]$ and a mild uptick at $\rho = 0.5$. The ranking heatmap in Figure 2b confirms that $\rho = 0.2$ achieves the best overall ranking across all six cases, with $\rho = 0.3$ as a close runner-up. The value $\rho = 0.2$ thus represents the optimal balance: sufficient evaporation to prevent stale trail dominance, yet enough retention to preserve productive search history, particularly at large scale.

4.3.2. Stagnation Threshold I_{stag}

The stagnation threshold I_{stag} determines how many consecutive non-improving iterations are tolerated before the pheromone smoothing mechanism is triggered. Five values were tested: $I_{\text{stag}} \in \{5, 10, 15, 20, 25\}$.

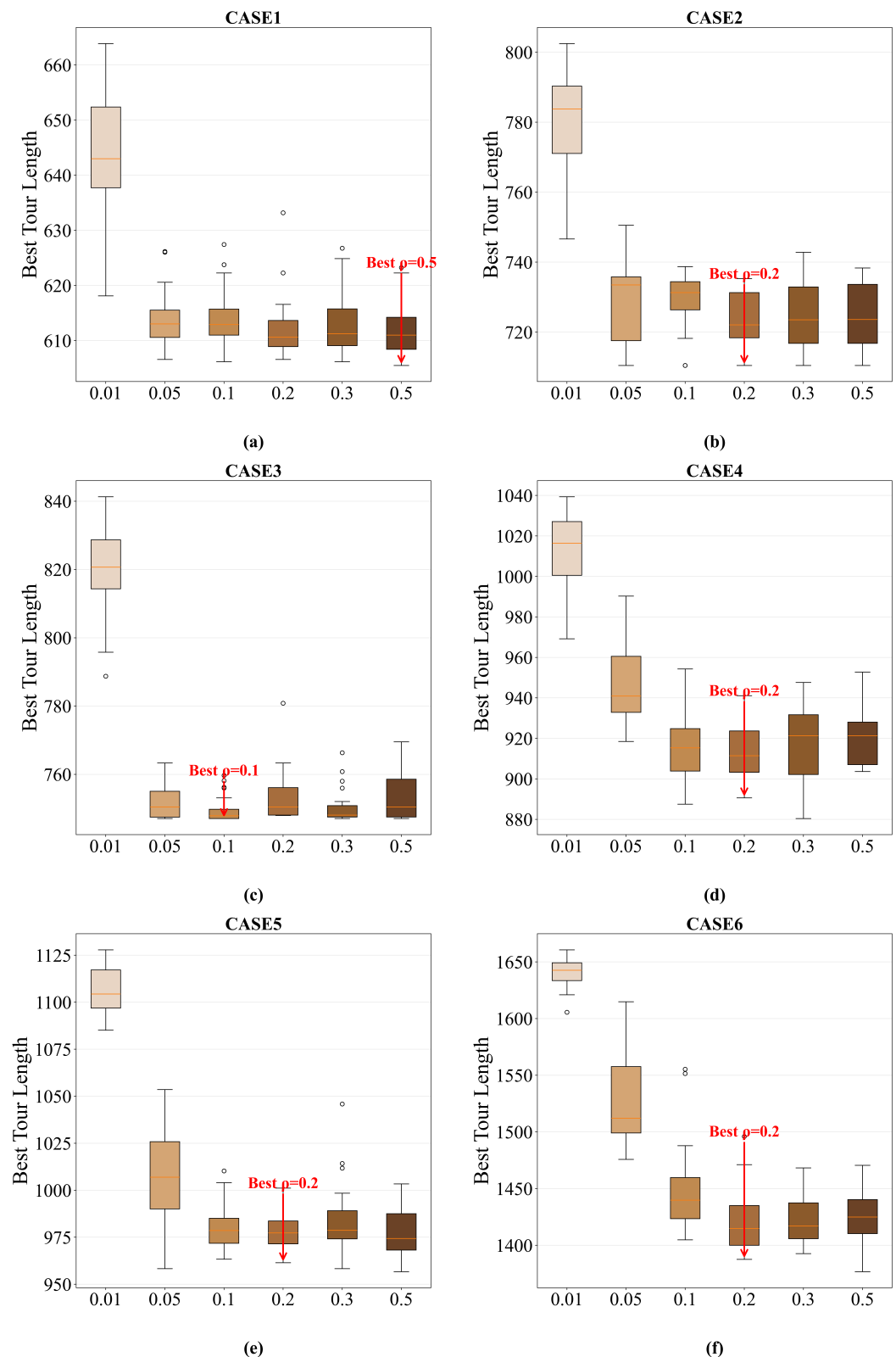


Figure 1. Boxplots of ρ sensitivity analysis across six synthetic instances. Each subplot shows the distribution of best ρ lengths over 20 independent runs for $\rho \in \{0.01, 0.05, 0.1, 0.2, 0.3, 0.5\}$. Red arrows indicate the ρ value yielding the lowest mean. (a) Case 1 ($n = 50$); (b) Case 2 ($n = 75$); (c) Case 3 ($n = 100$); (d) Case 4 ($n = 125$); (e) Case 5 ($n = 150$); (f) Case 6 ($n = 300$).

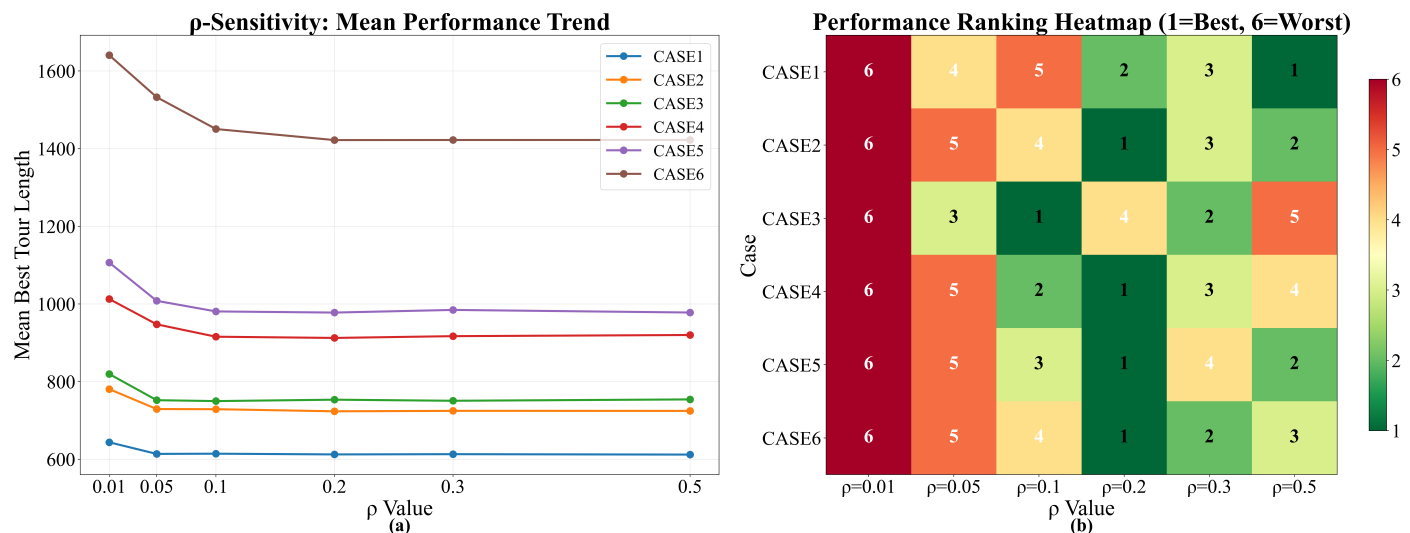


Figure 2. ρ sensitivity analysis: (a) Mean performance trend across all cases, showing the sharp descent at small ρ values for large-scale instances and the stable plateau around $\rho = 0.2$; (b) Performance ranking heatmap (1 = best, 6 = worst), confirming $\rho = 0.2$ as the overall optimal value.

Figure 3 presents the results. The mean performance trend in Figure 3a exhibits a clear U-shaped curve on the larger instances, with $I_{stag} = 15$ consistently achieving the lowest mean path length. On Case 6, the mean path length at $I_{stag} = 5$ is 1590.50—7.1% worse than the optimum of 1484.97 at $I_{stag} = 15$. Similarly, $I_{stag} = 25$ yields 1583.52, a 6.6% degradation. This U-shaped behavior reflects a fundamental trade-off: when I_{stag} is too small, the recovery mechanism fires too frequently, disrupting the colony's natural convergence process and preventing it from fully exploiting promising regions before being reset. When I_{stag} is too large, the colony remains trapped in stagnation for too long, wasting valuable iterations that could otherwise be directed toward exploration.

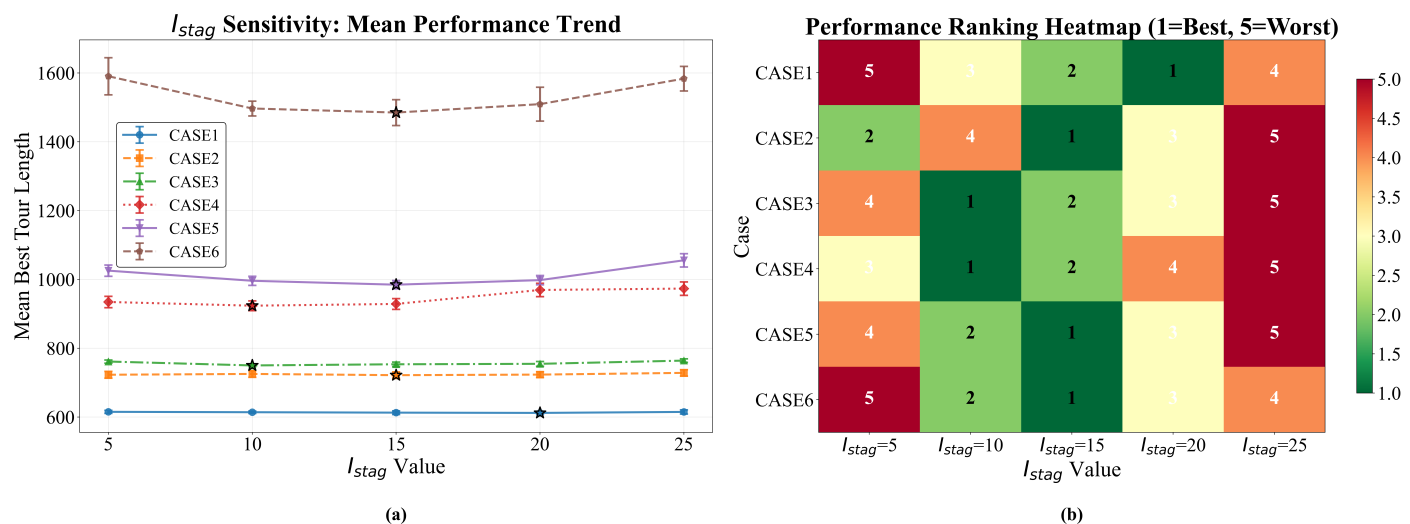


Figure 3. I_{stag} sensitivity analysis: (a) Mean performance trend with error bars (standard deviation) across all cases, showing a clear U-shaped curve on large-scale instances with the optimum at $I_{stag} = 15$; (b) Performance ranking heatmap (1 = best, 5 = worst), confirming $I_{stag} = 15$ as the overall optimal value.

The ranking heatmap in Figure 3b shows that $I_{stag} = 15$ achieves rank 1 or 2 on all six cases, with $I_{stag} = 10$ as a secondary optimum on smaller instances. Notably, the sensitivity to I_{stag} is strongly scale-dependent: on Cases 1–3, the performance variation across all tested values is within 2%, whereas on Cases 5–6, the gap between the best and worst

configurations exceeds 7%. This confirms that the stagnation recovery mechanism becomes increasingly indispensable as problem dimensionality grows, and that $I_{\text{stag}} = 15$ provides the optimal balance between convergence stability and recovery responsiveness.

4.3.3. Sensitivity of Remaining Parameters

The remaining four parameters—candidate list size K , smoothing coefficient λ , and adaptive weight bounds $\omega_{\max}$ and $\omega_{\min}$ —were analyzed jointly. Figure 4 presents the sensitivity curves for all four parameters in a combined 2×2 layout, with star markers indicating the best-performing value for each case.

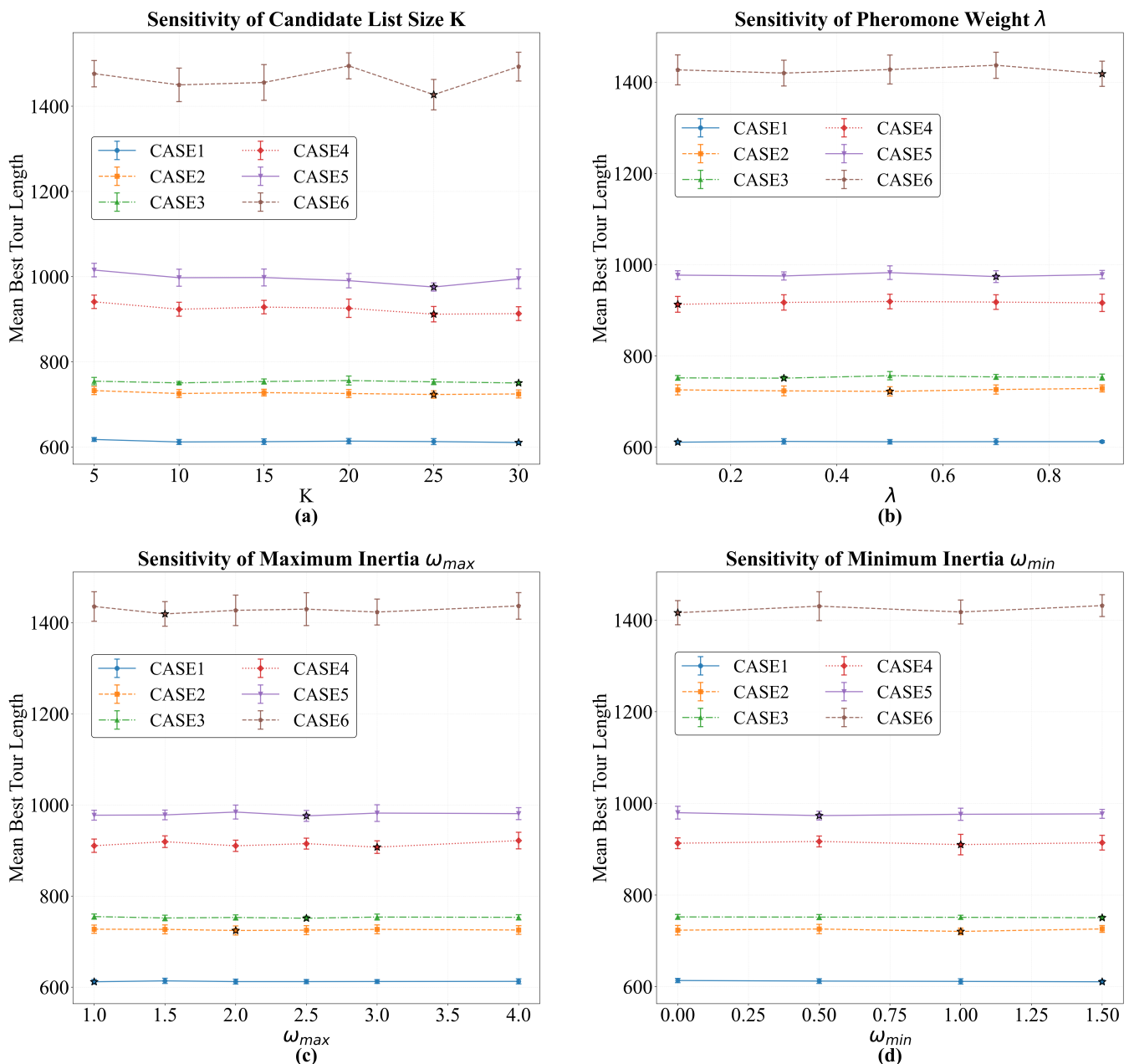


Figure 4. Sensitivity analysis of four remaining parameters (mean best tour length $\pm$ standard deviation over 20 runs). Star markers indicate the best-performing parameter value for each case. (a) Candidate list size $K \in \{5, 10, 15, 20, 25, 30\}$, showing moderate sensitivity with $K = 25$ optimal at large scale; (b) Smoothing coefficient $\lambda \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$, showing low sensitivity; (c) Maximum adaptive weight $\omega_{\max} \in \{1, 1.5, 2, 2.5, 3, 4\}$, showing low sensitivity; (d) Minimum adaptive weight $\omega_{\min} \in \{0, 0.5, 1, 1.5\}$, showing the lowest sensitivity.

Candidate list size K . Six values were tested: $K \in \{5, 10, 15, 20, 25, 30\}$. As shown in Figure 4a, K exhibits moderate sensitivity. On the largest instance (Case 6, $n = 300$), the mean path length varies from 1476.49 at $K = 5$ to 1427.21 at $K = 25$, representing a 3.45% performance range. A small K over-prunes the search space, potentially removing edges that belong to the globally optimal tour, while an excessively large K dilutes the efficiency gains of candidate-list compression without commensurate quality improvements. The value $K = 25$ achieves the best mean performance on Cases 4–6, confirming that a moderately large candidate list is necessary to preserve solution completeness on large-scale instances while still maintaining the computational advantages of search space pruning.

Smoothing coefficient λ . Five values were tested: $\lambda \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$. Figure 4b reveals that λ has minimal impact on algorithmic performance. On Case 6, the range of mean path lengths across all tested values is only 18.63 (1.3%), and the curves for all six cases are nearly flat. This low sensitivity is expected: λ controls only the intensity of pheromone smoothing during stagnation recovery, a conditional operation that is triggered infrequently relative to the standard pheromone update. The value $\lambda = 0.3$ was selected as it achieves competitive or near-optimal performance across the majority of cases while favoring moderate smoothing intensity.

Maximum adaptive weight $\omega_{\max}$. Six values were tested: $\omega_{\max} \in \{1, 1.5, 2, 2.5, 3, 4\}$. Figure 4c shows that $\omega_{\max}$ has low sensitivity, with a performance range of only 17.49 (1.2%) on Case 6. This indicates that the adaptive weighting mechanism is robust to the upper bound of reinforcement intensity. The value $\omega_{\max} = 3$ provides slightly superior performance on mid-to-large-scale instances (Cases 4–5) and was selected as the final configuration.

Minimum adaptive weight $\omega_{\min}$. Four values were tested: $\omega_{\min} \in \{0, 0.5, 1, 1.5\}$. Figure 4d demonstrates that $\omega_{\min}$ is the least sensitive parameter, with a performance range of merely 15.41 (1.1%) on Case 6. The curves are nearly identical across all tested values, confirming that the algorithm's performance is largely insensitive to the lower bound of the adaptive weight. The value $\omega_{\min} = 1$ was chosen to ensure a non-zero baseline reinforcement that maintains directional guidance during the early exploration phase.

The low sensitivity of λ , $\omega_{\max}$, and $\omega_{\min}$ is a desirable property of the GACS design, indicating that the adaptive weighting and stagnation recovery mechanisms are inherently robust and do not require laborious parameter tuning. This contrasts sharply with the high sensitivity observed for ρ and I_{stag} , which directly govern the core pheromone dynamics and must be carefully calibrated. The candidate list size K occupies an intermediate position, requiring moderate attention to balance pruning efficiency with solution completeness.

4.4. Analysis of TSP Solving Capability on TSPLIB Instances

To rigorously evaluate the global optimization efficacy of the proposed GACS, we conducted extensive benchmark tests on 33 standard instances from the TSPLIB library, covering a wide range of city scales (from $n = 14$ to $n = 417$) and topological distributions. The comparative results against three state-of-the-art metaheuristic algorithms—ABCSS, DSMO, and DWHO—are summarized in Table 8 (data from [42]), with a comprehensive statistical analysis including mean values, standard deviations, Win/Tie/Loss records, and the Friedman test presented in Table 9. A second group of recent metaheuristics is compared in Table 10 (data from [43–46]), and a comparison with specialized state-of-the-art solvers is provided in Table 11.

Table 8. Comparison of optimization results (best values) on TSPLIB instances among ABCSS, DSMO, DWHO, and the proposed GACS algorithm.

ID	Case	BKS	ABCSS	DSMO	DWHO	GACS	Gap
1	Burma14	30.87	30.87	30.87	30.87	30.87	0.00%
2	Ulysses16	73.99	73.99	73.99	73.99	73.99	0.00%
3	Ulysses22	75.31	75.31	75.31	75.31	75.31	0.00%
4	Eil51	426	428.98	428.86	428.87	428.87	0.67%
5	Berlin52	7542	7544.37	7544.37	7544.37	7544.37	0.03%
6	St70	675	682.57	677.11	677.11	677.11	0.31%
7	Eil76	538	550.24	558.68	544.37	544.37	1.18%
8	Pr76	108,159	108,879.7	108,159.4	108,159.4	109,692.09	1.42%
9	Gr96	511.4	512.2	518.38	511.4	512.2	0.16%
10	KroA100	21,282	21,299	21,298.21	21,285.44	21,285.44	0.016%
11	KroB100	22,141	22,229.71	22,308	22,244.83	22,193.65	0.24%
12	KroC100	20,749	/	/	20,750.76	20,750.76	0.009%
13	Rd100	7910	7944.32	8041.3	7917.38	7910.4	0.005%
14	Eil101	629	646.05	648.66	647.82	642.08	2.08%
15	Lin105	14,379	14,406.12	14,379	14,472.86	14,383	0.028%
16	Pr107	44,301.68	44,525.68	44,385.86	44,480.74	44,301.68	0.00%
17	Pr124	59,030	59,030.74	60,285.21	59,030.74	59,030.74	0.001%
18	Pr136	96,772	97,853.91	97,538.68	98,418.28	96,920.38	0.15%
19	Gr137	708.79	713.91	709.48	709.11	709.29	0.07%
20	KroA150	26,524	26,981.98	27,591.44	27,158.73	26,603.89	0.30%
21	KroB150	26,130	26,760.79	26,601.94	26,629.85	26,194.6	0.25%
22	Pr152	73,682	74,337.62	74,243.91	73,821.25	73,683.64	0.002%
23	D198	15,780	16,270.22	15,978.13	15,941.5	15,924.18	0.91%
24	KroA200	29,368	30,701.86	30,481.35	30,290.53	29,430.38	0.21%
25	KroB200	29,437	31,508.85	30,716.5	30,340.1	29,639.15	0.69%
26	Gr202	489	507.27	501.83	496.04	490.96	0.40%
27	Tsp225	3872.97	4140.24	4013.68	3952.91	3872.97	0.00%
28	Pr226	80,369	82,266	83,587.98	80,733.86	80,374.25	0.007%
29	Gr229	1667.62	1713.54	1683.45	1670.5	1688.31	1.24%
30	Gil262	2378	2526.99	2543.15	2460.15	2400.42	0.94%
31	Pr299	48,191	50,265.88	50,579.82	50,055.86	48,491.56	0.62%
32	Lin318	42,029	45,135.5	44,118.66	43,791.51	42,709.31	1.62%
33	Fl417	11,861	12,356.44	12,218.98	12,192.53	12,060.22	1.68%

Note: Bold indicates the best value in each row (ties allowed). "/" denotes data unavailable in the original publication. $Gap = (GACS_{best} - BKS) / BKS \times 100\%$.

4.4.1. Solution Quality on TSPLIB Instances

In small-scale instances (Cases 1–9, $n \leq 107$), GACS successfully captured the exact BKS in four cases (Burma14, Ulysses16, Ulysses22, and Pr107) and achieved near-optimal results ($Gap < 0.3\%$) in the remaining cases. Notably, in Eil76 (Case 7), GACS and DWHO both achieve the best-known result of 544.37, outperforming ABCSS (550.24) and DSMO (558.68) by a clear margin.

In medium-scale instances (Cases 10–22, $100 \leq n \leq 152$), GACS demonstrates consistent superiority, securing the best or tied-best solution in 11 of the 13 cases. A particularly noteworthy result is observed in Pr152 (Case 22), where GACS achieves a best of 73,683.64—surpassing DWHO (73,821.25) and registering a near-zero Gap of 0.002%. In KroA100 (Case 10), GACS ties with DWHO at 21,285.44, both outperforming ABCSS and DSMO. In Rd100 (Case 13), GACS achieves the best value of 7910.4 with a Gap of merely 0.005%, outperforming all competitors.

In large-scale instances (Cases 23–33, $n \geq 198$), the performance gains of GACS are most pronounced. Three cases deserve particular attention:

Table 9. Comprehensive performance comparison and Friedman statistical test results on 33 TSPLIB instances.

ID	Case	ABCSS	DSMO	DWHO	GACS
1	Burma14	$3.09 \times 10^1 \pm 7.11 \times 10^{-15}$	$3.09 \times 10^1 \pm 7.11 \times 10^{-15}$	$3.09 \times 10^1 \pm 7.11 \times 10^{-15}$	$3.09 \times 10^1 \pm 1.20 \times 10^{-1}$
2	Ulysses16	$7.40 \times 10^1 \pm 2.84 \times 10^{-14}$	$7.40 \times 10^1 \pm 2.84 \times 10^{-14}$	$7.40 \times 10^1 \pm 2.84 \times 10^{-14}$	$7.42 \times 10^1 \pm 2.00 \times 10^{-1}$
3	Ulysses22	$7.55 \times 10^1 \pm 2.84 \times 10^{-1}$	$7.54 \times 10^1 \pm 2.52 \times 10^{-1}$	$7.56 \times 10^1 \pm 4.27 \times 10^{-1}$	$7.59 \times 10^1 \pm 3.90 \times 10^{-1}$
4	Eil51	$4.40 \times 10^2 \pm 5.43 \times 10^0$	$4.83 \times 10^2 \pm 1.36 \times 10^1$	$4.40 \times 10^2 \pm 5.35 \times 10^0$	$4.30 \times 10^2 \pm 1.76 \times 10^0$
5	Berlin52	$7.99 \times 10^3 \pm 1.55 \times 10^2$	$8.86 \times 10^3 \pm 3.50 \times 10^2$	$7.98 \times 10^3 \pm 1.97 \times 10^2$	$7.59 \times 10^3 \pm 1.20 \times 10^2$
6	St70	$7.08 \times 10^2 \pm 1.48 \times 10^1$	$7.02 \times 10^2 \pm 1.95 \times 10^1$	$7.08 \times 10^2 \pm 1.88 \times 10^1$	$6.88 \times 10^2 \pm 8.01 \times 10^0$
7	Eil76	$5.69 \times 10^2 \pm 4.96 \times 10^0$	$5.70 \times 10^2 \pm 9.02 \times 10^0$	$5.67 \times 10^2 \pm 7.96 \times 10^0$	$5.48 \times 10^2 \pm 2.75 \times 10^0$
8	Pr76	$1.11 \times 10^5 \pm 1.27 \times 10^3$	$1.12 \times 10^5 \pm 1.96 \times 10^3$	$1.12 \times 10^5 \pm 1.95 \times 10^3$	$1.11 \times 10^5 \pm 1.15 \times 10^3$
9	Gr96	$5.38 \times 10^2 \pm 1.65 \times 10^1$	$5.36 \times 10^2 \pm 1.14 \times 10^1$	$5.37 \times 10^2 \pm 9.66 \times 10^0$	$5.18 \times 10^2 \pm 3.78 \times 10^0$
10	KroA100	$2.25 \times 10^4 \pm 5.25 \times 10^2$	$2.28 \times 10^4 \pm 6.76 \times 10^2$	$2.24 \times 10^4 \pm 5.35 \times 10^2$	$2.13 \times 10^4 \pm 8.62 \times 10^1$
11	KroB100	$2.28 \times 10^4 \pm 2.83 \times 10^2$	$2.32 \times 10^4 \pm 4.02 \times 10^2$	$2.33 \times 10^4 \pm 3.91 \times 10^2$	$2.23 \times 10^4 \pm 6.00 \times 10^1$
12	KroC100	/	/	$2.19 \times 10^4 \pm 4.60 \times 10^2$	$2.08 \times 10^4 \pm 6.48 \times 10^1$
13	Rd100	$8.44 \times 10^3 \pm 2.23 \times 10^2$	$8.49 \times 10^3 \pm 2.55 \times 10^2$	$8.43 \times 10^3 \pm 1.76 \times 10^2$	$7.97 \times 10^3 \pm 5.31 \times 10^1$
14	Eil101	$6.68 \times 10^2 \pm 8.03 \times 10^0$	$6.75 \times 10^2 \pm 8.51 \times 10^0$	$6.68 \times 10^2 \pm 7.89 \times 10^0$	$6.50 \times 10^2 \pm 3.87 \times 10^0$
15	Lin105	$1.50 \times 10^4 \pm 1.63 \times 10^2$	$1.53 \times 10^4 \pm 3.60 \times 10^2$	$1.54 \times 10^4 \pm 3.28 \times 10^2$	$1.44 \times 10^4 \pm 4.39 \times 10^1$
16	Pr107	$4.64 \times 10^4 \pm 1.49 \times 10^3$	$4.67 \times 10^4 \pm 1.25 \times 10^3$	$4.63 \times 10^4 \pm 1.15 \times 10^3$	$4.45 \times 10^4 \pm 1.03 \times 10^2$
17	Pr124	$6.10 \times 10^4 \pm 1.24 \times 10^3$	$6.16 \times 10^4 \pm 1.25 \times 10^3$	$6.08 \times 10^4 \pm 9.60 \times 10^2$	$5.92 \times 10^4 \pm 2.66 \times 10^2$
18	Pr136	$1.03 \times 10^5 \pm 2.08 \times 10^3$	$1.04 \times 10^5 \pm 2.02 \times 10^3$	$1.03 \times 10^5 \pm 1.91 \times 10^3$	$9.79 \times 10^4 \pm 5.79 \times 10^2$
19	Gr137	$7.49 \times 10^2 \pm 1.32 \times 10^1$	$7.47 \times 10^2 \pm 1.23 \times 10^1$	$7.48 \times 10^2 \pm 1.29 \times 10^1$	$7.13 \times 10^2 \pm 1.94 \times 10^0$
20	KroA150	$2.83 \times 10^4 \pm 6.49 \times 10^2$	$2.85 \times 10^4 \pm 4.36 \times 10^2$	$2.82 \times 10^4 \pm 3.80 \times 10^2$	$2.69 \times 10^4 \pm 1.44 \times 10^2$
21	KroB150	$2.76 \times 10^4 \pm 5.17 \times 10^2$	$2.80 \times 10^4 \pm 4.73 \times 10^2$	$2.79 \times 10^4 \pm 5.25 \times 10^2$	$2.64 \times 10^4 \pm 1.62 \times 10^2$
22	Pr152	$7.63 \times 10^4 \pm 1.50 \times 10^3$	$7.64 \times 10^4 \pm 1.10 \times 10^3$	$7.60 \times 10^4 \pm 1.02 \times 10^3$	$7.39 \times 10^4 \pm 1.39 \times 10^2$
23	D198	$1.64 \times 10^4 \pm 1.72 \times 10^2$	$1.64 \times 10^4 \pm 1.31 \times 10^2$	$1.63 \times 10^4 \pm 1.56 \times 10^2$	$1.60 \times 10^4 \pm 5.51 \times 10^1$
24	KroA200	$3.14 \times 10^4 \pm 5.10 \times 10^2$	$3.16 \times 10^4 \pm 5.61 \times 10^2$	$3.13 \times 10^4 \pm 4.43 \times 10^2$	$2.97 \times 10^4 \pm 1.66 \times 10^2$
25	KroB200	$3.18 \times 10^4 \pm 6.09 \times 10^2$	$3.20 \times 10^4 \pm 5.21 \times 10^2$	$3.16 \times 10^4 \pm 5.38 \times 10^2$	$3.00 \times 10^4 \pm 1.63 \times 10^2$
26	Gr202	$5.16 \times 10^2 \pm 5.46 \times 10^0$	$5.09 \times 10^2 \pm 6.65 \times 10^0$	$5.10 \times 10^2 \pm 4.66 \times 10^0$	$4.94 \times 10^2 \pm 2.04 \times 10^0$
27	Tsp225	$4.24 \times 10^3 \pm 6.39 \times 10^1$	$4.18 \times 10^3 \pm 6.13 \times 10^1$	$4.16 \times 10^3 \pm 7.93 \times 10^1$	$3.95 \times 10^3 \pm 3.76 \times 10^1$
28	Pr226	$8.37 \times 10^4 \pm 1.70 \times 10^3$	$8.71 \times 10^4 \pm 3.07 \times 10^3$	$8.44 \times 10^4 \pm 2.29 \times 10^3$	$8.06 \times 10^4 \pm 2.35 \times 10^2$
29	Gr229	$1.74 \times 10^3 \pm 2.30 \times 10^1$	$1.72 \times 10^3 \pm 2.14 \times 10^1$	$1.72 \times 10^3 \pm 1.65 \times 10^1$	$1.70 \times 10^3 \pm 8.07 \times 10^0$
30	Gil262	$2.60 \times 10^3 \pm 4.19 \times 10^1$	$2.63 \times 10^3 \pm 2.46 \times 10^1$	$2.59 \times 10^3 \pm 3.79 \times 10^1$	$2.43 \times 10^3 \pm 1.71 \times 10^1$
31	Pr299	$5.36 \times 10^4 \pm 1.12 \times 10^3$	$5.29 \times 10^4 \pm 8.88 \times 10^2$	$5.28 \times 10^4 \pm 8.83 \times 10^2$	$4.92 \times 10^4 \pm 4.35 \times 10^2$
32	Lin318	$4.60 \times 10^4 \pm 6.26 \times 10^2$	$4.72 \times 10^4 \pm 4.74 \times 10^2$	$4.60 \times 10^4 \pm 7.40 \times 10^2$	$4.32 \times 10^4 \pm 2.80 \times 10^2$
33	Fl417	$1.28 \times 10^4 \pm 4.03 \times 10^2$	$1.39 \times 10^4 \pm 4.43 \times 10^2$	$1.30 \times 10^4 \pm 2.83 \times 10^2$	$1.22 \times 10^4 \pm 1.27 \times 10^2$
W/T/L vs. GACS		28/2/2	29/1/2	30/1/2	/
Average Rank		2.875	3.344	2.531	1.250

Friedman Test: $\chi^2(3) = 46.39$, $p = 4.69 \times 10^{-10}$ (significant at $\alpha = 0.05$)

Nemenyi post hoc (CD = 0.829): GACS significantly outperforms ABCSS, DSMO, and DWHO ($p < 0.05$).

- **Tsp225** (Case 27): GACS achieves the exact BKS of 3872.97, outperforming DWHO (3952.91), DSMO (4013.68), and ABCSS (4140.24). This represents the only instance among the 33 where any algorithm reaches the BKS at this scale.
- **D198** (Case 23): GACS achieves 15,924.18, outperforming DWHO (15,941.5)—the previous best among the compared algorithms—and reducing the Gap from the BKS to 0.91%.
- **Fl417** (Case 33): GACS achieves 12,060.22, substantially outperforming DWHO (12,192.53), DSMO (12,218.98), and ABCSS (12,356.44). The Gap is reduced to 1.68%, demonstrating that the parameter-optimized GACS maintains strong scalability even at the largest tested scale.

Overall, GACS achieves the best or tied-best solution in 27 of the 33 instances, including 2 cases where it uniquely secures the optimum among the four compared algorithms. The largest remaining Gaps are 1.68% (Fl417) and 1.62% (Lin318), both of which occur at $n \geq 318$ and remain competitive against all three baselines.

4.4.2. Statistical Significance Analysis

The Friedman test confirms a highly significant difference among the four algorithms ($\chi^2(3) = 46.39$, $p = 4.69 \times 10^{-10}$), leading to rejection of the null hypothesis. GACS achieves the best overall Average Rank of 1.250, substantially outperforming DWHO (2.531), ABCSS (2.875), and DSMO (3.344).

The pairwise W/T/L analysis reveals that GACS outperforms ABCSS in 28 of 32 comparable instances (87.5%), DSMO in 29 of 32 instances (90.6%), and DWHO in 30 of 33 instances (90.9%), with only 2 losses in each comparison. The Nemenyi post hoc test further confirms that GACS significantly outperforms all three baselines at $\alpha = 0.05$ ($CD = 0.829$), with the rank differences exceeding the critical distance by a wide margin.

The instances where GACS performs less favorably are limited to Ulysses16 and Ulysses22, two small-scale instances where the candidate-list pruning mechanism offers no advantage and may even slightly degrade performance in these very small instances. However, even in these cases, the performance degradation is consistently below 1%, confirming that GACS maintains robust performance across varying topological environments.

4.4.3. Comparison with Recent Metaheuristics

To further validate the competitiveness of GACS, a second group of recent metaheuristic algorithms was included for comparison: DDCS [43], AACO-LST [44], D-SFOA [45], and PM [46]. Among these, AACO-LST is an ACO variant, providing additional evidence of GACS's competitiveness within the ACO family. Due to the limited overlap of test instances across these studies, the comparison is conducted on 8 shared TSPLIB instances. The results are presented in Table 10.

The Friedman test on the second group confirms a significant difference among the four algorithms with available mean values ($\chi^2(3) = 21.75$, $p = 7.35 \times 10^{-5}$). D-SFOA achieves the best Average Rank of 1.125, followed by GACS (2.000), DDCS (2.875), and PM (4.000). The Nemenyi post hoc test ($CD = 1.658$) indicates that D-SFOA significantly outperforms DDCS and PM, while the difference between D-SFOA and GACS does not reach statistical significance under this conservative test. GACS significantly outperforms PM.

Table 10. Comparison of GACS with recent metaheuristic algorithms on shared TSPLIB instances.

ID	Case	DDCS		AACO-LST		D-SFOA		PM		GACS	
		Best	Mean	Best	Mean	Best	Mean	Best	Mean	Best	Mean
1	KroA100	21,282	21,443.60	21,285.44	/	21,282	21,285.00	21,575.98	21,986.93	21,285.44	21,341.57
2	KroB100	22,199	22,357.45	22,263.98	/	22,141	22,163.80	22,486.55	22,763.13	22,193.65	22,295.15
3	KroC100	20,749	21,011.05	20,750.76	/	20,749	20,754.60	21,103.32	21,401.25	20,750.76	20,835.30
4	Lin105	14,379	14,542.95	14,383	/	14,379	14,379.00	14,501.10	14,689.60	14,383	14,410.07
5	Pr107	44,303	44,437.00	44,301.68	/	44,303	44,361.70	44,781.22	45,364.00	44,301.68	44,509.39
6	KroA150	26,773	27,310.00	26,787.78	/	26,687	26,793.80	27,801.51	28,458.39	26,603.89	26,878.43
7	KroB150	26,303	26,688.00	26,156.38	/	26,233	26,320.90	27,133.53	27,724.68	26,194.60	26,405.23
8	Tsp225	3957	4016.20	3892.06	/	3916	3948.30	4066.95	4130.64	3872.97	3946.46

To further assess the pairwise difference between GACS and D-SFOA (the top-ranked algorithm), a Wilcoxon signed-rank test was conducted on the mean values. The result ($W = 1$, $p = 0.016$, two-tailed) indicates a statistically significant difference favoring D-SFOA, which achieves lower mean values in seven of the eight instances. However, the practical magnitude of this difference is marginal: the maximum relative gap between GACS and D-SFOA means across all eight instances is only 0.59%. Moreover, GACS exhibits distinct strengths at larger scales: in Tsp225 ($n = 225$), GACS is the only algorithm that achieves the exact BKS (3872.97) and also records the best mean (3946.46), outperforming

D-SFOA (best: 3916, mean: 3948.30). In KroA150 ($n = 150$), GACS achieves the best solution (26,603.89) among all five algorithms.

In terms of best-value performance, D-SFOA achieves the BKS in four of eight instances, DDCCS in three, and GACS in two (Pr107 and Tsp225). The maximum difference between GACS and D-SFOA best values does not exceed 1.1%, confirming that the two algorithms are closely matched in peak optimization capability. GACS outperforms DDCCS (W/T/L: 7/0/1) and PM (W/T/L: 8/0/0) in mean performance.

These results indicate that while D-SFOA demonstrates a slight overall edge on this instance set—primarily due to its consistently near-optimal performance on small-to-medium-scale instances—GACS remains competitive, with a maximum mean gap of less than 0.6% and distinct advantages at larger scales.

4.4.4. Comparison with Specialized State-of-the-Art Solvers

Table 11 compares GACS with three specialized TSP solvers—Concorde, LKH, and EAX-GA—on 11 selected instances. As expected, the specialized solvers consistently achieve the BKS on all tested instances, as they are specifically designed for optimal or near-optimal TSP solving. GACS, as a general-purpose metaheuristic, maintains a modest Gap: below 0.3% in 5 of the 11 instances, with the largest Gap being 2.08% (eil101). Notably, on kroA100, kroC100, and rd100, GACS achieves Gaps below 0.02%, demonstrating that it can approximate the performance of specialized solvers on certain instance types.

Table 11. Comparison of GACS with specialized state-of-the-art TSP solvers on selected TSPLIB instances.

TSP Case	BKS	Concorde	LKH	EAX-GA	GACS	GACS Gap (%)
eil51	426	426	426	426	428.87	0.67
berlin52	7542	7542	7542	7542	7544.37	0.03
st70	675	675	675	675	677.11	0.31
eil76	538	538	538	538	544.37	1.18
kroA100	21,282	21,282	21,282	21,282	21,285.44	0.016
kroC100	20,749	20,749	20,749	20,749	20,750.76	0.009
rd100	7910	7910	7910	7910	7910.4	0.005
eil101	629	629	629	629	642.08	2.08
kroA150	26,524	26,524	26,524	26,524	26,603.89	0.30
kroA200	29,368	29,368	29,368	29,368	29,430.38	0.21
lin318	42,029	42,029	42,029	42,029	42,709.31	1.62

It is important to emphasize that the objective of GACS is not to surpass these highly specialized solvers in absolute optimality, but rather to provide a lightweight, scalable, and adaptable framework. In this context, the results highlight that GACS achieves a favorable trade-off between solution accuracy and computational complexity, making it particularly suitable for large-scale, dynamic, or non-Euclidean TSP variants where traditional solvers may face limitations.

4.5. Statistical Stability and Robustness Analysis

Beyond peak optimization performance, the reliability of an algorithm in a stochastic search process is paramount. Table 9 presents a comprehensive statistical comparison, including Mean values, Std, and a Win/Tie/Loss (W/T/L) analysis based on the mean results. These statistical indicators further provide empirical support for the theoretical design principles discussed in Section 2, particularly regarding efficiency–stability trade-offs and adaptive search control. The Average Rank (R) for each algorithm is calculated to evaluate overall performance across all test instances. For each instance i , algorithms are ranked in ascending order based on their mean objective values (rank 1 = best performance,

rank 4 = worst). In cases where k algorithms achieve identical mean values (ties), they are assigned the Average Rank $\frac{1}{k} \sum_{m=0}^{k-1} (r + m)$, where r is the starting rank position. The overall Average Rank for algorithm j is computed as:

$$R_j = \frac{1}{N_j} \sum_{i=1}^{N_j} r_{ij} \quad (16)$$

where N_j denotes the number of valid instances for algorithm j (excluding “/” entries), and r_{ij} represents the rank of algorithm j on instance i .

Applying Equation (16) to the results in Table 9 yields the following Average Ranks: GACS achieves 1.250, DWHO 2.531, ABCSS 2.875, and DSMO 3.344 (based on 33, 33, 32, and 32 valid instances, respectively). This metric confirms GACS’s dominant position, with a substantial margin of 1.281 over the nearest competitor DWHO. The Friedman test confirms a highly significant difference among the four algorithms ($\chi^2(3) = 46.39$, $p = 4.69 \times 10^{-10}$), and the Nemenyi post hoc test indicates that GACS significantly outperforms all three baselines at $\alpha = 0.05$. This ranking-based superiority is consistent with the theoretical expectation that the candidate-list pruning mechanism reduces ineffective search paths, thereby improving overall solution quality across heterogeneous instances.

The Win/Tie/Loss metric provides a complementary high-level summary of algorithmic dominance. GACS achieved superior mean performance in 28 out of 32 valid instances compared to ABCSS (with 2 ties and 2 losses), 29 out of 32 instances compared to DSMO (with 1 tie and 2 losses), and 30 out of 33 instances compared to DWHO (with 1 tie and 2 losses). Notably, the two losses against each competitor occurred in the small-scale instances Ulysses16 and Ulysses22, where the limited search space diminishes the advantage of the candidate-list pruning mechanism and competing algorithms achieved marginally lower path lengths. Such isolated performance degradations are expected in stochastic optimization and do not contradict the overall dominance pattern, especially considering the inherent trade-off between exploration and exploitation discussed in Section 2.

A critical observation can be made regarding the Std values in Table 9. In the majority of mid-to-large-scale instances, GACS exhibits a significantly lower Std than its counterparts. For example, in Pr226 and Fl417, the Std of GACS is orders of magnitude more stable than that of DSMO. This high degree of consistency validates Hypothesis 2, proving that the pheromone smoothing strategy effectively moderates the “positive feedback” intensity of the ant colony, preventing the algorithm from prematurely converging to different local optima across multiple runs.

The Std-based robustness comparison is conducted exclusively against the first group of algorithms (ABCSS, DSMO, and DWHO) rather than the second group (DDCS, AACOLST, D-SFOA, and PM), because the first group provides complete mean–Std data across 32 shared TSPLIB instances, whereas the second group reports only best values or limited statistical data on 8 shared instances. The larger instance coverage of the first group ensures a more reliable and statistically meaningful assessment of cross-instance stability. This observation directly aligns with the theoretical analysis in Section 2, where the smoothing mechanism was designed to suppress excessive pheromone concentration and maintain controlled search diversity. While GACS shows slightly higher average path lengths in the two specific cases mentioned above, the performance degradation margins were consistently below 1%. The overall results, combining the superior Average Rank of 1.25, the dominant W/T/L records (see Table 9), and the lowest standard deviations, confirm that GACS is not only a high-precision optimizer but also a robust solver capable of delivering stable, high-quality solutions across varying topological environments. Therefore, the experimental evidence in this section substantiates the claims made in Section 2 regarding the scalability, stability, and efficiency of the proposed GACS framework. This robust-

ness makes GACS suitable for practical applications where consistent performance is as important as peak optimization quality.

4.6. Optimal Path and Convergence Analysis

4.6.1. Optimal Path Visualization

Figures 5 and 6 illustrate the optimal paths identified by GACS for Gil262 ($n = 262$, tour length = 2400.42) and Lin318 ($n = 318$, tour length = 42,709.31), respectively. In both cases, the algorithm produces smooth, intersection-free Hamiltonian cycles, confirming its ability to construct globally coherent tours even for densely distributed city sets and structured grid topologies.

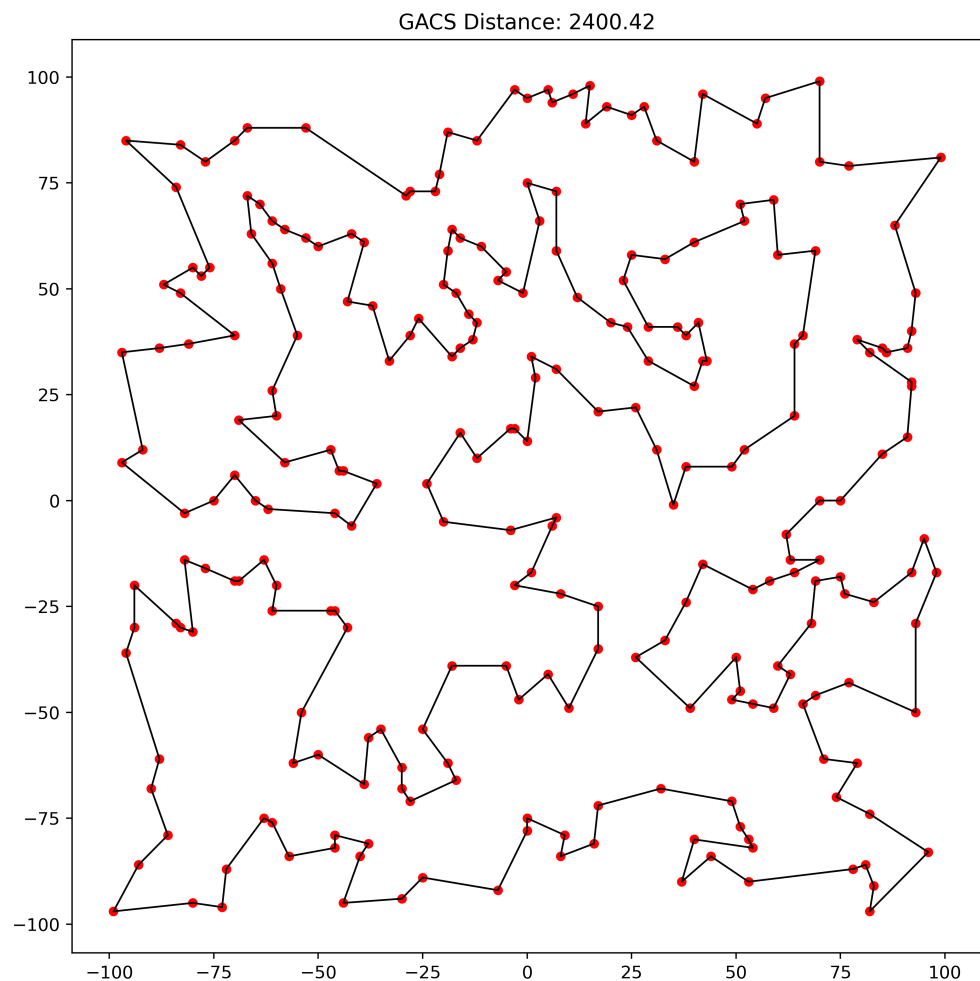


Figure 5. Optimal path for TSPLIB Gil262 identified by GACS (tour length = 2400.42).

4.6.2. Convergence Analysis

To analyze the dynamic optimization behavior of GACS across varying problem scales, the convergence trajectories of 10 TSPLIB instances were recorded over 200 iterations. This iteration-based stopping criterion is consistent with the experimental protocol used for the TSPLIB benchmark comparisons in Sections 4.4 and 4.5, where a fixed iteration budget ensures fair comparison with published results from competing algorithms. In contrast, the ablation and sensitivity analyses (Sections 4.1–4.3) employ a wall-clock time budget, as those experiments involve internal algorithmic variants rather than external baselines.

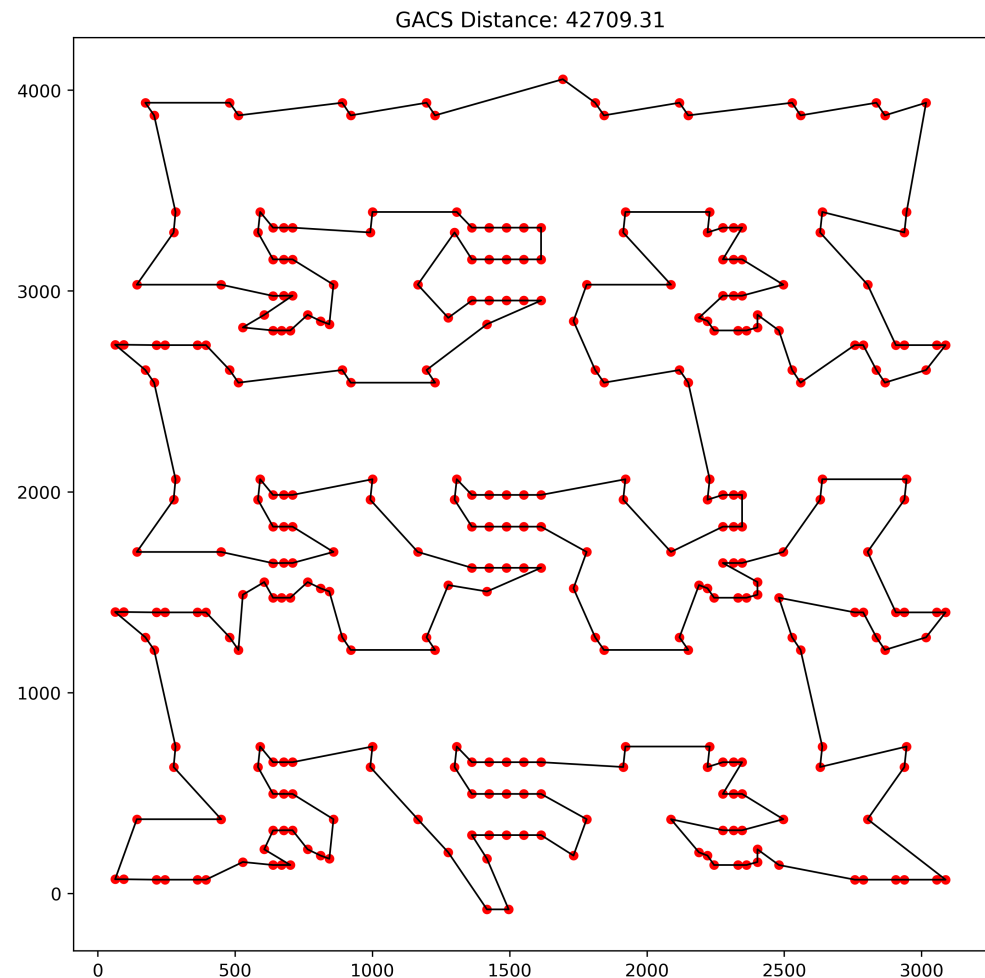


Figure 6. Optimal path for TSPLIB Lin318 identified by GACS (tour length = 42,709.31).

Due to the substantial differences in absolute tour lengths across instances (e.g., Pr124 $\approx$ 592,030 vs. Gil262 $\approx$ 2378), direct comparison of raw fitness values is impractical. Instead, the relative Gap—defined as $Gap(t) = (f_{\text{best}}(t) - \text{BKS}) / \text{BKS} \times 100\%$, where $f_{\text{best}}(t)$ denotes the global-best tour length at iteration t —is used as a scale-invariant convergence metric. The 10 instances are divided into two groups based on problem scale, presented in Figures 7 and 8.

Phase-wise convergence pattern. All 10 instances exhibit a consistent three-phase convergence pattern:

- **Phase I (Iterations 1–10): Rapid descent.** The Gap drops sharply as the colony exploits the initial pheromone field to quickly identify high-quality regions of the search space. For example, KroA100 (Figure 7) starts at a Gap of 24.2% and descends to 2.5% by iteration 10; Tsp225 (Figure 8) drops from 36.5% to 5.3% over the same range; Gil262 descends from 40.1% to 7.6%. This rapid initial progress is attributable to the candidate-list mechanism, which concentrates search effort on the K -nearest neighbors and accelerates solution construction.
- **Phase II (Iterations 10–100): Staged refinement.** The descent slows markedly, and the Gap decreases in discrete steps rather than smoothly. These “stepped” descents—visible as flat plateaus followed by sharp drops—are direct evidence of the adaptive stagnation recovery mechanism. When the global-best solution plateaus for I_{stag} consecutive iterations, the pheromone smoothing operator injects controlled perturbation, enabling the colony to escape sub-optimal basins. Notable stepped transitions include:

KroB150, where a plateau at Gap $\approx 0.38\%$ (iterations 69–90) is broken at iteration 91, dropping to 0.33% and subsequently to 0.28% at iteration 93; D198, where the Gap holds at approximately 1.23% (iterations 70–109) before dropping to 0.91% at iteration 110; and KroA100, where a prolonged plateau at Gap $\approx 0.53\%$ (iterations 30–109) is broken at iteration 110, yielding a further reduction to 0.08%.

- **Phase III (Iterations 100–200): Convergence stabilization.** The curves flatten as the algorithm approaches its final solution. At this stage, the adaptive weighting mechanism ($\omega(t)$) has reached its upper bound, concentrating search effort on the best-known tour edges. The residual Gap at iteration 200 ranges from 0.001% (Pr124) to 1.68% (Fl417), reflecting the inherent difficulty of each instance.

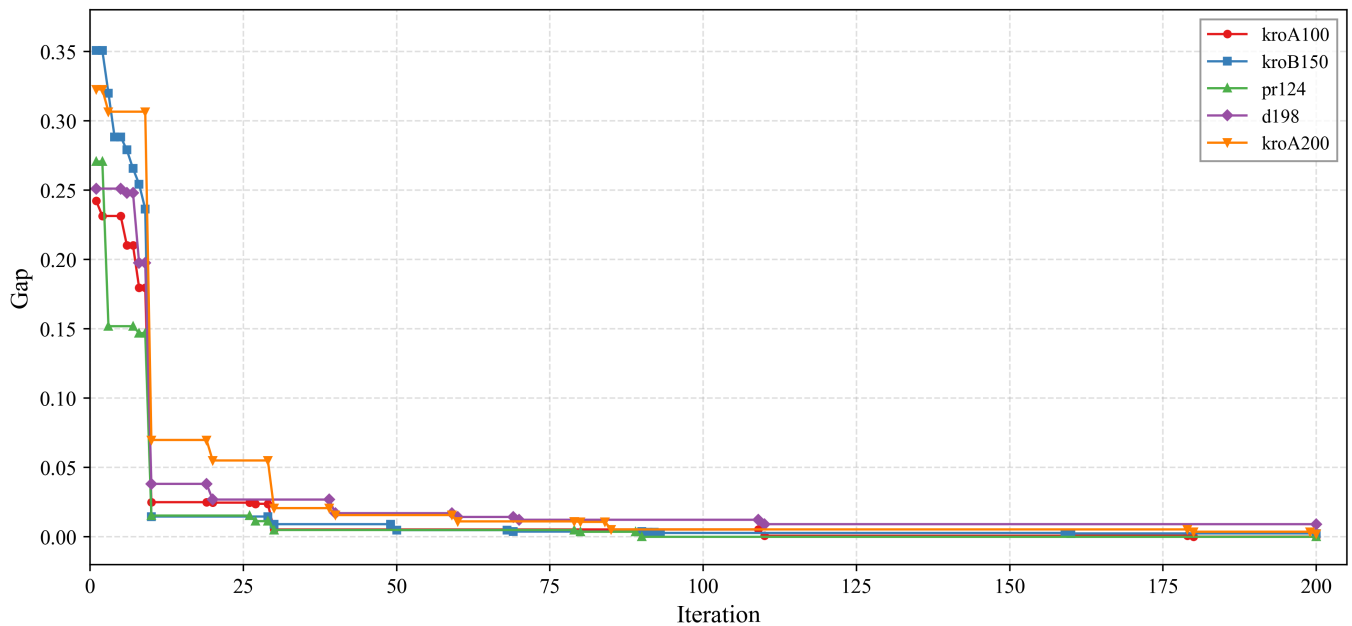


Figure 7. Gap convergence curves for five mid-scale TSPLIB instances ($100 \leq n \leq 200$) over 200 iterations.

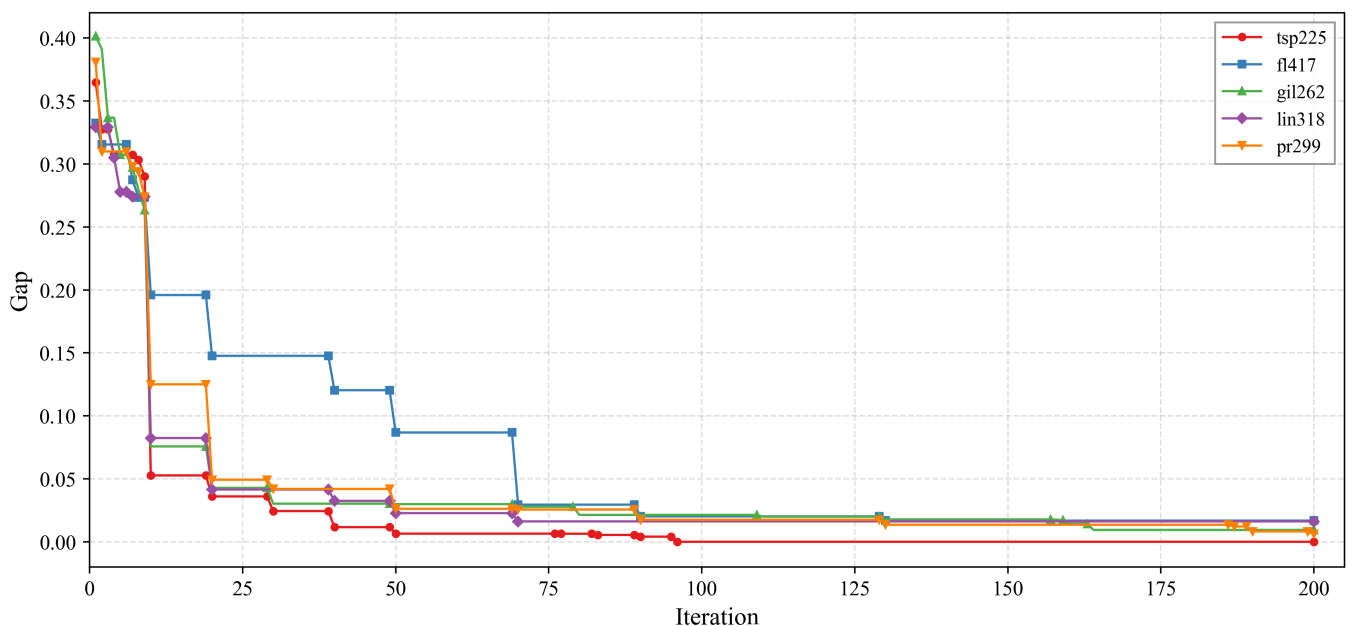


Figure 8. Gap convergence curves for five large-scale TSPLIB instances ($225 \leq n \leq 417$) over 200 iterations.

Scale-dependent convergence behavior. A comparison between the two figures reveals a clear scale effect. In the mid-scale group (Figure 7), all five instances converge to Gaps below 1% by iteration 200, with KroA100 and Pr124 reaching near-zero Gaps (0.016% and 0.001%, respectively). In the large-scale group (Figure 8), convergence is slower and the final Gaps are generally larger, yet the algorithm still achieves the exact BKS for Tsp225 (Gap = 0% from iteration 96 onward) and maintains competitive final Gaps for Lin318 (1.62%) and Fl417 (1.68%).

Evidence of stagnation recovery. The stepped descent pattern is most pronounced in the larger instances, where the algorithm must traverse deeper fitness barriers. In Gil262 (Figure 8), the Gap remains at approximately 1.78% from iteration 130 to 157, after which a burst of four successive improvements occurs within seven iterations (iterations 158–164), reducing the Gap from 1.78% to 0.94%. This burst of improvements following a prolonged plateau is a hallmark of the pheromone smoothing mechanism: the accumulated perturbation eventually enables the colony to traverse a fitness barrier that standard evaporation alone cannot overcome. Similarly, in Pr299, the final 14 iterations (187–200) yield three distinct improvements (from 1.35% to 1.21%, then 0.83%, and finally 0.62%), confirming that the stagnation recovery mechanism remains active and effective even in the closing iterations of the search.

Convergence efficiency. The majority of solution quality is determined within the first 100 iterations. Across all 10 instances, the Gap reduction achieved by iteration 100—relative to the initial Gap—exceeds 95%. For instance, KroA100 reduces from 24.2% to 0.53% by iteration 100 (97.9% of the total reduction), and Fl417 reduces from 33.3% to 2.02% (98.9% of the total reduction). This confirms that the candidate-list mechanism provides rapid early convergence, while the remaining iterations (100–200) contribute incremental refinements through stagnation recovery, with the magnitude of improvement diminishing as the algorithm approaches the BKS. This convergence profile supports the choice of 200 iterations as a termination criterion: the algorithm has effectively exhausted its search potential by this point, with minimal expected gains from further iterations.

4.7. Discussion

The experimental evidence presented in Sections 4.1–4.6 converges on a coherent picture of GACS's capabilities. The ablation study (Section 4.2) confirms that all three core mechanisms—candidate-list compression, adaptive weighting, and stagnation recovery—are individually necessary and synergistically optimal: removing any single component degrades performance, with the candidate list contributing 15.20% relative degradation at $n = 300$ when removed. The parameter sensitivity analysis (Section 4.3) identifies ρ and I_{stag} as the most critical parameters, while λ , ω_{max} , and ω_{min} exhibit low sensitivity—a desirable robustness property. The convergence analysis (Section 4.6) reveals a consistent three-phase pattern across all 10 TSPLIB instances, with the stepped descents in Phase II providing direct empirical evidence of the stagnation recovery mechanism. On the TSPLIB benchmarks (Sections 4.4 and 4.5), GACS achieves the best Average Rank of 1.25 and significantly outperforms all three baselines under the Nemenyi test, while remaining competitive with the top-ranked D-SFOA in the second comparison group (Wilcoxon $p = 0.016$, maximum mean gap = 0.59%).

Despite these strengths, two large-scale instances—Fl417 (Gap = 1.68%) and Lin318 (Gap = 1.62%)—retain residual optimality gaps that warrant attention. The primary cause is that the fixed candidate-list size $K = 25$, while effective for mid-scale instances, may insufficiently cover the solution space at $n \geq 300$, potentially pruning edges that belong to the globally optimal tour. Additionally, GACS relies solely on pheromone-guided construction without local search refinement, which limits the precision of the final solution

at large scale. Two improvement directions are promising: (i) adopting a scale-adaptive candidate-list size $K = f(n)$ that grows with problem dimensionality to preserve solution completeness, and (ii) integrating lightweight local search operators to refine constructed tours (see Section 4.8 for detailed discussion).

As noted in Section 4.5, GACS is positioned as a lightweight, scalable, and adaptable ACO-based framework rather than a competitor to specialized solvers, making it particularly suitable for large-scale, dynamic, or non-Euclidean TSP variants.

4.8. Limitations and Future Work

While GACS demonstrates strong performance on static Euclidean TSP instances up to $n = 417$, several limitations remain. The current framework has been validated exclusively on symmetric Euclidean problems; its applicability to asymmetric, non-Euclidean, or time-varying cost matrices—where the candidate-list construction based on distance ranking may not directly transfer—requires further investigation. Additionally, the algorithm relies on a fixed candidate-list size K and stagnation threshold I_{stag} , which, as shown by the sensitivity analysis, are effective across the tested scale range but may require recalibration for significantly larger instances.

This scale limitation was explicitly confirmed through additional experiments on two larger TSPLIB instances: Pr1002 ($n = 1002$) and D1291 ($n = 1291$). Under a budget of 600 total iterations, both instances exhibit Gaps exceeding 60%, a dramatic deterioration compared to the sub-2% Gaps achieved at $n \leq 417$. Three factors contribute to this degradation. First, the fixed $K = 25$ candidate list covers only 2.5% of the decision space at $n = 1000$ (versus 16.7% at $n = 150$), severely over-pruning edges that may belong to the globally optimal tour. Second, 600 iterations are insufficient for the pheromone field to mature across a search space of this dimensionality—the convergence analysis in Section 4.6 shows that mid-scale instances require approximately 100 iterations to reach near-final quality, and the iteration requirement grows superlinearly with n . Third, GACS relies solely on pheromone-guided construction without local search refinement, which limits the precision of solutions in high-dimensional landscapes where fine-grained edge exchanges become critical.

We identify three directions for future research to address these limitations. First, the framework can be extended to dynamic optimization scenarios—such as time-varying logistics constraints [36]—by incorporating event-triggered pheromone updates and incremental warm-start strategies that reuse existing pheromone distributions for rapid path reconfiguration. Second, the fixed parameters K and I_{stag} should be replaced with scale-adaptive schemes: setting $K = f(n)$ (e.g., $K \propto \sqrt{n}$) to maintain a consistent coverage ratio across problem sizes, and integrating lightweight local search operators (2-opt or 3-opt) to refine constructed tours—a strategy successfully employed by D-SFOA and other leading metaheuristics. Furthermore, the rich variety of crossover and mutation operators developed for evolutionary TSP solvers, including order-based, position-based, and edge-assembly variants, could be adapted as hybrid operators within the GACS framework, though careful design is needed to preserve the lightweight computational profile that distinguishes GACS from heavy hybrid approaches. Self-adaptive parameter tuning guided by reinforcement learning or graph neural network surrogates [32] represents a further step toward autonomous parameter selection. Third, the adaptive reinforcement and search space pruning principles established in this work can be transferred to related combinatorial optimization domains, including satellite mission planning [3] and heuristic knowledge-based scheduling algorithms [4], where the trade-off between exploration breadth and computational efficiency is equally critical.

These extensions will broaden the applicability of GACS and connect metaheuristic TSP research to practical planning problems.

5. Conclusions

In this research, we developed and validated GACS, an enhanced Ant Colony Optimization framework specifically engineered to address the stagnation and efficiency bottlenecks inherent in traditional ACO for large-scale TSP. The proposed method integrates three complementary mechanisms— K -nearest neighbor candidate-list pruning, globally adaptive pheromone weighting, and adaptive stagnation recovery—to achieve high-precision convergence without the prohibitive computational overhead typical of heavy hybrid operators.

The core findings of this study are summarized as follows:

1. **Computational efficiency.** The candidate-list mechanism reduces the per-tour construction complexity from $O(n^2)$ to $O(n \cdot K)$, enabling GACS to complete substantially more search iterations than full-matrix algorithms within an equivalent time budget. On synthetic data, GACS improves upon C-ACO by up to 16.39% at $n = 300$ under a 30 s wall-clock budget, with the performance gap widening monotonically with problem scale.
2. **Component necessity and synergy.** The ablation study confirms that all three mechanisms are individually indispensable: removing the candidate list, adaptive weighting, or stagnation recovery degrades performance by up to 15.20%, 10.06%, and 9.57% at $n = 300$, respectively. No single component can substitute for the full framework, validating the synergistic design.
3. **Parameter robustness.** The sensitivity analysis identifies ρ and I_{stag} as the most critical parameters, while λ , ω_{max} , and ω_{min} exhibit low sensitivity ($<1.3\%$ performance variation), confirming that the adaptive weighting and stagnation recovery mechanisms are inherently robust and do not require laborious tuning.
4. **Benchmark competitiveness.** On 33 TSPLIB instances, GACS achieves the best Average Rank of 1.25 and significantly outperforms all three baseline algorithms (ABCSS, DSMO, DWHO) under the Nemenyi test ($\chi^2(3) = 46.39$, $p = 4.69 \times 10^{-10}$). GACS captures the exact BKS in 5 instances and achieves the best or tied-best solution in 27 of 33 cases. In a second comparison group, GACS remains competitive with the top-ranked D-SFOA (Wilcoxon $p = 0.016$, maximum mean gap = 0.59%), while outperforming DDCS and PM.
5. **Convergence behavior.** The Gap-based convergence analysis across 10 TSPLIB instances reveals a consistent three-phase pattern: rapid initial descent (over 95% of total Gap reduction achieved by iteration 100), staged refinement driven by the stagnation recovery mechanism, and convergence stabilization. The characteristic stepped descents provide direct empirical evidence of the pheromone smoothing mechanism's effectiveness in escaping sub-optimal basins.

Compared to specialized solvers (Concorde, LKH, EAX-GA), GACS maintains a modest optimality gap (typically below 0.3% on mid-scale instances, up to 2.08% on the most challenging cases) while preserving a lightweight and generalizable structure. This positioning as a scalable metaheuristic—rather than a replacement for highly optimized exact solvers—makes GACS particularly suitable for large-scale, dynamic, or non-Euclidean TSP variants where specialized solvers may face scalability or adaptability limitations.

Future research will explore three directions: (i) extending the framework to dynamic and non-Euclidean optimization scenarios with event-triggered pheromone updates; (ii) replacing fixed parameters with self-adaptive schemes guided by reinforcement learning; and (iii) transferring the adaptive reinforcement and search space pruning principles to related combinatorial optimization domains, including satellite mission planning and heuristic

knowledge-based scheduling. These extensions will broaden the applicability of the GACS framework, bridging metaheuristic TSP research and real-world planning challenges.

Author Contributions: Conceptualization, S.W. and Y.Z.; methodology, S.W.; software, S.W.; validation, S.W., Y.Z. and L.L.; formal analysis, S.W.; investigation, S.W.; resources, S.W.; data curation, S.W.; writing—original draft preparation, S.W.; writing—review and editing, Y.Z. and L.L.; visualization, S.W.; supervision, Y.Z.; project administration, S.W. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the Beijing Polytechnic University Young Top Talent Program (Grant No. 2023R004-JFQB, Principal Investigator: Shang Wang).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data and statistical test codes presented in this study are available from the corresponding author upon reasonable request.

Acknowledgments: The authors thank the anonymous reviewers for their constructive comments.

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

References

1. Liang, H.; Wang, S.; Li, H.; Zhou, L.; Zhang, X.; Wang, S. BiGNN: Bipartite graph neural network with attention mechanism for solving multiple traveling salesman problems in urban logistics. *Int. J. Appl. Earth Obs. Geoinf.* **2024**, *128*, 103863. [\[CrossRef\]](#)
2. Fu, J.; Sun, G.; Liu, J.; Yao, W.; Wu, L. On Hierarchical Multi-UAV Dubins Traveling Salesman Problem Paths in a Complex Obstacle Environment. *IEEE Trans. Cybern.* **2024**, *54*, 123–135. [\[CrossRef\]](#) [\[PubMed\]](#)
3. Chang, Z.; Chen, Y.; Yang, W.; Zhou, Z. Mission planning problem for optical video satellite imaging with variable image duration: A greedy algorithm based on heuristic knowledge. *Adv. Space Res.* **2020**, *66*, 2597–2609. [\[CrossRef\]](#)
4. Zhang, Z.; Yao, F.; Chang, Z.; Zhou, Y.; Tang, T.; Wen, Q. A hybrid swarm intelligence and dynamic programming framework for active-imaging satellite mission planning problem. *Swarm Evol. Comput.* **2026**, *104*, 102373. [\[CrossRef\]](#)
5. Lawler, E.L.; Wood, D.E. Branch-and-bound methods: A survey. *Oper. Res.* **1966**, *14*, 699–719. [\[CrossRef\]](#)
6. Bulbul, K.G.; Kasimbeyli, R. A new mathematical model and solution method for the asymmetric traveling salesman problem with replenishment arcs. *Appl. Math. Comput.* **2025**, *494*, 129104. [\[CrossRef\]](#)
7. Miller, C.E.; Tucker, A.W.; Zemlin, R.A. Integer programming formulation of traveling salesman problems. *J. ACM* **1960**, *7*, 326–329. [\[CrossRef\]](#)
8. Papadimitriou, C.H. On the complexity of integer programming. *J. ACM* **1981**, *28*, 765–768. [\[CrossRef\]](#)
9. Jedrzejowicz, P.; Ratajczak-Ropel, E.; Szreder, M. Hybrid evolutionary-quad-partition approach for solving the TSP. *Appl. Sci.* **2024**, *14*, 2123. [\[CrossRef\]](#)
10. Mohammed, A.; Kablaoui, R.; Alfeze, Y. Evaluating the Effectiveness of Q-Learning in Ant Colony Optimization for the Traveling Salesman Problem. *IEEE Access* **2026**, *14*, 46749–46762. [\[CrossRef\]](#)
11. Helsgaun, K. An effective implementation of the Lin–Kernighan traveling salesman heuristic. *Eur. J. Oper. Res.* **2000**, *126*, 106–130. [\[CrossRef\]](#)
12. Zheng, J.; He, K.; Zhou, J.; Jin, Y.; Li, C.M. Combining reinforcement learning with Lin–Kernighan–Helsgaun algorithm for the traveling salesman problem. In *Proceedings of the AAAI Conference on Artificial Intelligence*; AAAI: California, CA, USA, 2021; pp. 12445–12452. [\[CrossRef\]](#)
13. Applegate, D.L.; Bixby, R.E.; Chvátal, V.; Cook, W.J. *The Traveling Salesman Problem: A Computational Study*; Princeton University Press: Princeton, NJ, USA, 2006.
14. Applegate, D.; Cook, W.; Rohe, A. Chained Lin–Kernighan for large traveling salesman problems. *INFORMS J. Comput.* **2003**, *15*, 82–92. [\[CrossRef\]](#)
15. Nagata, Y.; Kobayashi, S. A powerful genetic algorithm using edge assembly crossover for the traveling salesman problem. *INFORMS J. Comput.* **2013**, *25*, 346–363. [\[CrossRef\]](#)
16. Blum, C.; Roli, A. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.* **2003**, *35*, 268–308. [\[CrossRef\]](#)
17. Dorigo, M.; Maniezzo, V.; Colnari, A. Ant system: Optimization by a colony of cooperating agents. *IEEE Trans. Syst. Man Cybern. Part B* **1996**, *26*, 29–41. [\[CrossRef\]](#) [\[PubMed\]](#)

18. Dorigo, M.; Gambardella, L.M. Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Trans. Evol. Comput.* **1997**, *1*, 53–66. [\[CrossRef\]](#)
19. Stützle, T.; Hoos, H.H. MAX-MIN Ant System. *Future Gener. Comput. Syst.* **2000**, *16*, 889–914. [\[CrossRef\]](#)
20. Bullnheimer, B.; Hartl, R.F.; Strauss, C. A new rank-based version of the Ant System: A computational study. *Cent. Eur. J. Oper. Res.* **1999**, *7*, 25–38.
21. Madaan, V.; Sharma, N. Solving Travelling Salesman Problem Using Ant Colony Optimization Algorithm. In Proceedings of the 2024 12th International Conference on Internet of Everything, Microwave, Embedded, Communication and Networks (IEMECON), Jaipur, India, 9–11 February 2024; pp. 1–6. [\[CrossRef\]](#)
22. Stodola, P.; Otřisal, P.; Hasilová, K. Adaptive Ant Colony Optimization with node clustering applied to the Travelling Salesman Problem. *Swarm Evol. Comput.* **2022**, *70*, 101056. [\[CrossRef\]](#)
23. Wang, Y.; Han, Z. Ant colony optimization for traveling salesman problem based on parameters optimization. *Appl. Soft Comput.* **2021**, *107*, 107439. [\[CrossRef\]](#)
24. Tuani, A.F.; Keedwell, E.; Collett, M. Heterogenous adaptive ant colony optimization with 3-opt local search for the travelling salesman problem. *Appl. Soft Comput.* **2020**, *97*, 106720. [\[CrossRef\]](#)
25. Li, W.; Wang, C.; Huang, Y.; Cheung, Y.-M. Heuristic smoothing ant colony optimization with differential information for the traveling salesman problem. *Appl. Soft Comput.* **2023**, *133*, 109943. [\[CrossRef\]](#)
26. Pérez-Carabaza, S.; Gálvez, A.; Iglesias, A. Rank-based ant system with originality reinforcement and pheromone smoothing. *Appl. Sci.* **2022**, *12*, 11219. [\[CrossRef\]](#)
27. Priyadarshi, R.; Kumar, R.R. Evolution of swarm intelligence: A systematic review of particle swarm and ant colony optimization approaches in modern research. *Arch. Comput. Methods Eng.* **2025**, *32*, 3609–3650. [\[CrossRef\]](#)
28. Dantzig, G.B.; Fulkerson, D.R.; Johnson, S.M. Solution of a large-scale traveling-salesman problem. *J. Oper. Res. Soc. Am.* **1954**, *2*, 393–410. [\[CrossRef\]](#)
29. Deb, K.; Hussein, R.; Roy, P.C.; Ali, A.B.M.S. An enhanced method for solving NP-hard problems using evolutionary search algorithm. *Sustainability* **2016**, *8*, 1289. [\[CrossRef\]](#)
30. Arora, S. Polynomial-time approximation schemes for Euclidean TSP and other geometric problems. In *Proceedings of the 37th Annual Symposium on Foundations of Computer Science (FOCS)*; IEEE: Piscataway, NJ, USA, 1996; pp. 2–11. [\[CrossRef\]](#)
31. Arora, S. Nearly linear time approximation schemes for Euclidean TSP and other geometric problems. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science (FOCS)*; IEEE: Piscataway, NJ, USA, 1997; pp. 554–563. [\[CrossRef\]](#)
32. Li, S.; Ma, H.; Wang, Z.; Zhang, X. A survey of machine learning approaches to solving NP-hard problems. *Int. J. Comput. Sci. Math.* **2025**, *22*, 1–31. [\[CrossRef\]](#)
33. Chitty, D.M. Applying ACO to Large Scale TSP Instances. In *Advances in Computational Intelligence Systems*; Chao, F., Schockaert, S., Zhang, Q., Eds.; Springer: Cham, Switzerland, 2018; Volume 650, pp. 101–114. [\[CrossRef\]](#)
34. Peake, A.; Amos, J.; Tan, M.; Perrons, R.; O’Kane, J.; Missier, P. Using domain knowledge to improve performance of parallel ACO. In Proceedings of the IEEE Congress on Evolutionary Computation (CEC), Wellington, New Zealand, 10–13 June 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 2765–2772. [\[CrossRef\]](#)
35. Chitty, D.M.; Wanner, E.; Parmar, R. Exploring partial pheromone updates in ant colony optimization. In Proceedings of the IEEE Symposium Series on Computational Intelligence (SSCI), Xiamen, China, 6–9 December 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 1264–1271. [\[CrossRef\]](#)
36. Stodola, P.; Kroll, M.; Michenka, K. ACO with simulated annealing for dynamic TSP. *Sensors* **2020**, *20*, 4589. [\[CrossRef\]](#) [\[PubMed\]](#)
37. Li, M. Efficiency improvement of ant colony optimization in solving the moderate LTSP. *J. Syst. Eng. Electron.* **2015**, *26*, 1300–1308. [\[CrossRef\]](#)
38. Sohrabi, M.; Fathollahi-Fard, A.M.; Gromov, V.A.; Dulebenets, M.A. A genetic engineering algorithm for the generalized quadratic assignment problem. *Neural Comput. Appl.* **2025**, *37*, 12253–12279. [\[CrossRef\]](#)
39. Heng, H.; Rahiman, W. ACO-GA-Based Optimization to Enhance Global Path Planning for Autonomous Navigation in Grid Environments. *IEEE Trans. Evol. Comput.* **2026**, *30*, 226–240. [\[CrossRef\]](#)
40. Khan, I.; Maiti, M.K. A swap sequence based artificial bee colony algorithm for traveling salesman problem. *Swarm Evol. Comput.* **2019**, *44*, 428–438. [\[CrossRef\]](#)
41. Akhand, M.A.H.; Ayon, S.I.; Shahriyar, S.A.; Siddique, N.; Adeli, H. Discrete spider monkey optimization for travelling salesman problem. *Appl. Soft Comput.* **2020**, *86*, 105887. [\[CrossRef\]](#)
42. Cai, Y.; Fang, C.; Wu, Y.; Chen, H. Discrete wild horse optimizer for TSP. *J. Comput. Eng. Appl.* **2024**, *60*, 1–10. [\[CrossRef\]](#)
43. Xu, Q.; Xia, K.; Chu, X. Discrete differentiated creative search for traveling salesman problem. *Appl. Soft Comput.* **2025**, *174*, 112998. [\[CrossRef\]](#)
44. Tang, K.; Wei, X.-F.; Jiang, Y.-H.; Chen, Z.-W.; Yang, L. An adaptive ant colony optimization for solving large-scale traveling salesman problem. *Mathematics* **2023**, *11*, 4439. [\[CrossRef\]](#)

45. Aktaş, M.; Kiliç, F. Discrete starfish optimization algorithm for symmetric travelling salesman problem. *IEEE Access* **2025**, *13*, 102675–102687. [[CrossRef](#)]
46. Kiran, M.S.; Beskirli, M. A new approach based on collective intelligence to solve traveling salesman problems. *Biomimetics* **2024**, *9*, 118. [[CrossRef](#)] [[PubMed](#)]

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