mathematics

MDPI

Article

An Adaptive Ant Colony Optimization for Solving Large-Scale Traveling Salesman Problem

Kezong Tang [1], Xiong-Fei Wei [1,*], Yuan-Hao Jiang [2,*], Zi-Wei Chen [1] and Lihua Yang [1]

[1] School of Information Engineering, Jingdezhen Ceramic University, Jingdezhen 333403, China; tangkezong@jci.edu.cn (K.T.); 2220042009@stu.jcu.edu.cn (Z.-W.C.); yanglihua@jci.edu.cn (L.Y.)
[2] School of Computer Science and Technology, East China Normal University, Shanghai 200062, China
* Correspondence: 2120044005@stu.jcu.edu.cn (X.-F.W.); jiangyuanhao@stu.ecnu.edu.cn (Y.-H.J.)

Abstract: The ant colony algorithm faces dimensional catastrophe problems when solving the large-scale traveling salesman problem, which leads to unsatisfactory solution quality and convergence speed. To solve this problem, an adaptive ant colony optimization for large-scale traveling salesman problem (AACO-LST) is proposed. First, AACO-LST improves the state transfer rule to make it adaptively adjust with the population evolution, thus accelerating its convergence speed; then, the 2-opt operator is used to locally optimize the part of better ant paths to further optimize the solution quality of the proposed algorithm. Finally, the constructed adaptive pheromone update rules can significantly improve the search efficiency and prevent the algorithm from falling into local optimal solutions or premature stagnation. The simulation based on 45 traveling salesman problem instances shows that AACO-LST improves the solution quality by 79% compared to the ant colony system (ACS), and in comparison with other algorithms, the PE of AACO-LST is not more than 1% and the *Err* is not more than 2%, which indicates that AACO-LST can find high-quality solutions with high stability. Finally, the convergence speed of the proposed algorithm was tested. The data shows that the average convergence speed of AACO-LST is more than twice that of the comparison algorithm. The relevant code can be found on our project homepage.

Keywords: meta-heuristic algorithm; adaptive optimization; traveling salesman problem; large-scale optimization; path optimization

MSC: 68T20; 68T07

Citation: 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. https://doi.org/10.3390/math11214439

Academic Editor: Xibei Yang

Received: 4 September 2023
Revised: 3 October 2023
Accepted: 11 October 2023
Published: 26 October 2023

1. Introduction

Optimization problems belong to an important branch of operations research, and the corresponding solution problems can be mainly classified into three categories: discrete, continuous, and mixed-integer planning problems, depending on the problem to be solved. Continuous optimization problems are characterized by continuous decision variables and are usually solved by Newton's method [1] and the gradient descent method [2]. Combinatorial optimization problems [3] differ from continuous optimization problems in that their decision variables are discrete. Accurately solving algorithms, such as integer programming [4], cut-plane method [5], branch-and-bound method [6], dynamic programming [7], and minimum spanning tree method [8] are widely used in solving combinatorial optimization type problems. However, as the scale of TSP increases, the time cost of accurately solving algorithms also increases exponentially, which poses obstacles to the use of this type of algorithms.

Currently, artificial intelligence has brought significant changes to the development of the world, and complex tasks represented by the recognition of DNA-protein binding have been effectively solved [9]. However, for combinatorial optimization problems, the difficulty of solving the resulting NP-hard problem increases exponentially as the

problem size increases. In this class of problems, if a strictly exact solution is required, the computational complexity will increase exponentially with the increase of decision variables. To cope with complex optimization problems, heuristic algorithms are introduced and used to solve this class of problems while obtaining superior search efficiency. However, heuristic algorithms are difficult to jump out of the local optimal solution, which also creates a bottleneck for the improvement of search efficiency. Therefore, meta-heuristic algorithms have been proposed and widely used in this field. These algorithms are represented by differential evolution [10], genetic algorithm [11,12], and ant colony optimization [13–16], which introduce stochastic search into a heuristic algorithm to help the algorithm better jump out of locally optimal solutions and increase the possibility of finding globally optimal solutions.

The traveling salesman problem (TSP) is a classical combinatorial optimization problem where the objective of the problem is to find the shortest route between a given set of cities that can visit all of them and return to the starting city at the end. Although the TSP has been proven to be NP-hard theoretically, it still receives the attention of many researchers due to its applications in many practical scenarios, such as route planning, circuit board design, and military communication [12]. The ant colony algorithm, which has a strong advantage in solving TSP, is a heuristic optimization algorithm proposed by Dorigo [17], who was inspired by people's research results on the collective behavior of real ant colonies in nature, and it has the same advantages as genetic algorithm [11], particle swarm optimization [14–21], and immune algorithm [22–25], such as strong robustness and easy to combine with other methods, and has a wide range of application areas.

With the development of modern technology, there is an increasing need to solve TSP on a larger scale and higher complexity. On the one hand, in terms of practical applications, in the manufacturing field, large-scale TSP optimization reduces equipment downtime and improves the output of the production line through the optimization of paths. In the field of bioinformatics, the large-scale TSP derived from gene sequencing provides a method for understanding the structure and function of the genome. In addition, it has a huge role to play in the modeling of navigation satellite systems, in the logistics and transportation industry, in the wiring of large-scale circuits, and many other areas. On the other hand, in terms of the importance of the research, large-scale TSP is an important challenge in the field of computer science and operations research, and studying and solving this problem can help to advance algorithmic research. For this reason, scholars have proposed many improvement strategies. Wang et al. [26] proposed hybrid symbiotic organisms search (SOS) and ant colony optimization (ACO) to solve the TSP, which can adaptively optimize the parameters of ACO through SOS, reduce the complexity of parameter allocation of the algorithm, and also cited a simple local optimization strategy to improve the convergence rate and solution quality of it. Yang et al. [27] introduced the nucleolar game strategy into the algorithm, and the cooperative ant colony system (ACS) groups shared the pheromone distribution and allocated the cooperative profits through nucleolus to improve the convergence of the algorithm. In reference [28], a dynamic flight ant colony optimization algorithm is used to solve the TSP. The colony in the algorithm includes common ants and flying ants, and the flying ants store pheromones by injecting them from a distance, so the pheromones add to the path between adjacent nodes, and the number of nodes receiving pheromones will change according to the quality of the solution compared with the rest of the solutions, thus achieving a balance in exploitation and exploration. Zhang et al. [29] applied opposition-based learning (OBL) to the ant colony algorithm. According to the characteristics of the solution of TSP, two strategies of constructing opposite paths by using OBL were proposed, and three different algorithm frameworks were designed to utilize the information of opposite paths to improve the algorithm performance. In the traditional ant colony algorithm, the heuristic information is obtained by the reciprocal of the distance between nodes, without considering the need to return to the starting city in the last step. Shahadat et al. [30] adopted the general formula of visibility heuristic associated with the final destination city to intelligently deal with the problem of returning

to the starting city. Yu et al. [31] found that the traditional ant colony algorithm would fall into the local optimum when increasing the pheromone concentration factor. To solve this problem, a disturbance factor was added to the algorithm to eliminate the interference of other factors on the ant colony movement, and then the functional relationship between the moving distance and the pheromone concentration was increased. In addition, the firefly algorithm was introduced to further optimize it. Li et al. [32] proposed a heuristic smooth ant colony algorithm with differential edge information to solve TSP, which strengthened the exploration of differential edge by using the differential edge information obtained from candidate solutions and made the evaporation operation of pheromone trail can get more reasonable guidance. In this article, the state transfer rule of ACS [17] is improved by using the sine-cosine function, then the 2-opt operator is introduced to locally optimize the better part of paths, thus the adaptive pheromone update rule is constructed. Experiments show that adaptive ant colony optimization for large-scale TSP (AACO-LST) has a significant effect on solving large-scale TSP.

In this article, the related work was introduced in Section 2. In Section 3, we discussed the principles of the ant colony algorithm and gave the main framework and details of the AACO-LST algorithm. Section 4 presents comparison results between AACO-LST and other comparison algorithms on different TSP instances. Section 5 outlines future research directions.

2. Related Work

Combinatorial optimization problems have always been a significant research direction in the fields of computer science and operations research, which can be described in a formulaic manner:

$$\begin{cases} \min f(x) \\ s.t.\ \ g(x) \geq 0 \\ \quad\ x \in D \end{cases} \tag{1}$$

In Equation (1), f(x) is the objective function, g(x) is the constraint function, x is the decision variable, and D denotes a finite discrete decision space. The traveling salesman problem is a popular field of study in combinatorial optimization. There is a lot of related work on the TSP, and exact algorithms belong to the early part of algorithms that were used to solve it, specifically dating back to 1954. When a dynamic programming algorithm for solving the symmetric TSP was proposed by Dantzig, Fulkerson, and Johnson, it was also called the Dantzig-Fulkerson-Johnson (DFJ) algorithm. The model of the algorithm includes allocation constraints and sub-loop elimination constraints in addition to binary constraints on the decision variables. However, the exponential increases in the number of constraints of the model with the increase in the size of the city causes the algorithm to be ineffective in solving large-scale TSP. Therefore, it is necessary to construct a TSP model with only polynomial constraints to improve the solution efficiency. For example, Pérez-Carabaza S. et al. [33] extended the current popular Rank-based Ant System (ASrank) by introducing a strategy of initial reinforcement for top ranked solutions and a pheromone smoothing mechanism to avoid the stagnation, effectively improving the search ability of the algorithm; Miller C. E. et al. [34] proposed a mixed-integer programming model with a polynomial number of constraints to describe the TSP. Mixed-integer programming problems are usually solved exactly using the branch-and-bound method. Nikolaev A. et al. [35] proposed a branch-and-bound algorithm based on the 1-tree Lagrangian relaxation method, which proposes a new branching strategy, i.e., branching on 1-tree edges belonging to vertices in the 1-tree of the highest degree with the largest tolerance, which is shown by testing on benchmark instances in TSPLIB to have higher solution quality compared to branching on the shortest edge and the strong branching proposed by Held M. et al. [36]. However, the exact algorithm still has limitations for solving large-scale TSP due to the limitations of the number of variables and the number of constraints.

To cope with the computational complexity of the TSP, various heuristic algorithms have been introduced to find approximate optimal solutions. Among them, the ant colony

optimization, genetic algorithm, particle swarm optimization, and simulated annealing algorithm have become the focus of research. Roy A. et al. [37] proposed a novel memetic genetic algorithm (NMGA) to solve the TSP, which incorporates Boltzmann probabilistic selection and random variation multipoint crossover techniques. During multipoint crossover, the paternal chromosomes and common crossover points are randomly selected. Two zygotic chromosomes are produced after a cost and distance comparison of neighboring crossover points of the paternal chromosomes involved in the crossover. Emambocus B. A. S. et al. [38] proposed an enhanced swap sequence-based particle swarm optimization. The strategy of integrating the extended particle swarm optimization (XPSO) in the exchange sequence-based particle swarm optimization is because even though XPSO is only suitable for solving sequential optimization problems, it also has a high performance in variants of PSO. İlhan İ. et al. [39] proposed a new optimization algorithm for solving TSP called a list-based simulated annealing algorithm with crossover operator (LBSA-CO), a population-based meta-heuristic algorithm. In the LBSA-CO, a list-based temperature cooling scheme is used which adapts to the topology of the solution space. The solutions in the population are improved by inversion, insertion, and 2-opt local search operators. The sequential crossover and edge-reorganization crossover operators are applied to the improved solutions to accelerate the convergence. In addition, the parameters of LBSA-CO were tuned using the Taguchi method, resulting in a significant improvement in the performance of LBSA-CO over SA. These heuristic algorithms mentioned above find suboptimal solutions in acceptable time through their optimization process, which also show good performance and scalability in solving large-scale TSPs.

In recent years, the field of machine learning has been making impressive achievements, and the application of reinforcement learning (RL) to solve combinatorial optimization problems has become a new research direction. Ma et al. [40] used RL to train hierarchical graph pointer networks (HGPNs) to find the optimal city alignments under constraints by learning a hierarchical strategy, which can effectively solve large-scale TSP. Zheng et al. [41] applied Q-learning, Sarsa, and Monte Carlo to the Lin-Kernighan-Helsgaun (LKH) algorithm, replacing the inflexible traversal operation in the LKH algorithm by using RL to allow the program to learn to make choices at each step of the search. However, the exploration of intelligence in RL is built on sequences of state information, such as the features of state, action, reward, and new state, in which the feature states need to be set up in a time-consuming manner. It is difficult to cope with the complexity of continuous states and high-dimensional action spaces, which can easily lead to a large increase in dimensionality.

The research work of this article is to propose an adaptive ant colony algorithm for large-scale TSP (AACO-LST). Compared with ACS, the key parameters in AACO-LST will be adjusted adaptively and dynamically with the iteration of the algorithm, expanding the solution space while searching, accelerating the convergence speed of the algorithm, and introducing an effective local optimization strategy into it, which can avoid the algorithm from falling into the local optimum prematurely. Therefore, the advantage of the AACO-LST will be more and more obvious with the increase in the number of cities. The main contributions of this work are as follows:

(1) The state transfer rules of the ACO are a crucial part of the algorithm, which determines the rule for choosing the next action during the search process for each ant. However, the ant of the traditional ACO often falls into the local optimal solution too early during the search process, which leads to the premature convergence of the algorithm and prevents the algorithm from finding a better solution. This is usually due to the over-reliance of state transfer rules on pre-existing pheromone guidance and the lack of exploration mechanisms. To help ants strike a balance between exploration and exploitation, this article improves the state transfer rule of the traditional ant colony algorithm. When the number of iterations of the algorithm increases, the pheromone concentration on the optimal path is gradually much higher than that on the other paths. To prevent the algorithm from falling into local optimality, the importance of the pheromone concentration

in the path selection should be gradually reduced. The weight factors α and β will be adjusted stochasticly and adaptively by combining the sine-cosine function, i.e., the value of α will gradually decrease and the value of β will increase accordingly in a nonlinear manner, expanding the search space of the solution and finding the optimal solution faster.

(2) The *k*-opt is a commonly used local search optimization algorithm. It generates new solutions by disconnecting *k*-th edges on a path and reconnecting them. By introducing the local search mechanism of *k*-opt, ants can try different ways of path reconnection and perform a broader search within the local neighborhood. To improve the solution performance, a simplified 2-opt operator is introduced in AACO-LST to locally optimize the ants' path. During each iteration, to avoid the huge time cost associated with local optimization of all ants' paths, AACO-LST performs this operation only for the paths between two nodes of some of the selected ants. At this point, the paths between these two nodes are reversed and new paths are formed in this way, and then the better one is compared and selected. This helps to extend the exploration capability of the algorithm in the search space, jumping out of the local optimum and further exploring the globally optimal solution.

(3) To improve population diversity, the global search capability of the algorithm is strengthened, and a set of adaptive pheromone updating rules are constructed in AACO-LST. Incorporating the ant's number factor into the local pheromone update rule makes the pheromone initialized with a uniform distribution of individuals and good population diversity, increasing the probability of finding the global optimal solution. Regarding the dimensions of the pheromone release, the global pheromone update is performed on the locally optimized paths, and in the global pheromone update rules, those rules from the ASrank [33] are incorporated. Then, the locally optimized paths are ranked in ascending order of length, with the higher-ranked ants releasing more pheromones, so the algorithm can strongly guide ant searches through differences in the concentration of different paths. Regarding the dimension of pheromone volatilization, an adaptive method is used to adjust the global pheromone volatilization factor, and an iterative threshold is set for the global volatilization factor so that once the algorithm falls into a local optimum in the mid-to-late stages, the value of the global volatilization factor is reduced to avoid the algorithm from falling into premature stagnation. The improved local pheromone updating rules and global pheromone updating rules cooperate, and jointly promote the AACO-LST to gradually converge to the global optimal solution during the search process through the adaptive method, and maintain a certain balance of exploration and exploitation.

3. Design of Algorithm Framework

3.1. Principles of the Experiments

In this section, a general summary of the formulas in this article is provided. The formulas in Section 3.2 are derived from the ant system proposed by Dorigo M. et al. [42], which describes the working principle of the ant system. Equation (2) shows a state transition rule that determines the probability that the next city may be visited by the *k*-th ant. To prevent ants from choosing cities that have already been visited, $tabu_k$ is used to represent the taboo table for the *k*-th ant. So Equation (3) calculates the set of cities that ant k can visit next. Equation (4) is used to update the amount of pheromone on each edge. Dorigo M. et al. [42] have given three different models, each model of ants has their way of releasing pheromones, respectively, as shown in Equations (5)–(7), the current general use of Equation (7). Kumar S. et al. [43] integrated the sine-cosine functions into the ant colony algorithm to enhance the ability of global exploration and local development of their proposed algorithm. Inspired by this, this article adaptively adjusts the weight factors α and β through Equations (8) and (9) in Section 3.3 by using the mathematical properties of the cosine function and sine function, so that the parameter values are close to the optimal value in the search, so the state transition Equation (2) is adjusted to Equation (10). Tuani A. F. et al. [44] use 3-opt to locally optimize the path taken by ants, and its algorithm complexity is $O(n^3)$. On this basis, a simplified 2-opt operator

with time complexity $O(n)$ is designed in this article to improve the quality of the solution while reducing the time complexity, as shown in Equations (11) and (12) in Section 3.4. The formulas in Section 3.5 are all modifications of the ACS pheromone update rules and Equation (13) incorporates an ant population factor into the local pheromone update rules to enhance population diversity. Meanwhile, the global pheromone update rules are modified to Equations (14)–(16), focusing on adaptive regulation in both the release and evaporation of pheromones. Equations (17)–(19) in Section 4.2 are several indicators for measuring the quality of solutions. Equation (17) represents the degree of deviation between the algorithm's obtained optimal solution and the known optimal solution. Equation (18) represents the relative error between the algorithm's average solution and the known optimal solution. Equation (19) represents the relative error between the obtained optimal solution and the average solution [26]. The experiment will refer to these three indicators to verify the quality of algorithmic solutions. Equations (20)–(22) show the principle of calculating the diversity of a population. The relevant code can be found on our project homepage: https://github.com/xiongfeide/AACO-LST (accessed on 29 September 2023).

3.2. Basic Ant Colony Algorithm

Suppose there is a set C of n cities, m ants, $D(i, j)$ ($i, j = 1, 2, \ldots, n$) denotes the distance from city i to j, and $\tau_{ij}(t)$ denotes the concentration of pheromone on the line connecting city i to j at the moment t. At the initial moment, the amount of pheromone on each path is equal. The k-th ant ($k = 1, 2, \ldots, m$) decides the transfer direction according to the amount of information on each path during its movement. $P^k{}_{ij}(t)$ denotes the probability that k-th ant transfers from position i to j at the moment t:

$$\begin{cases} P_{ij}^k(t) = \begin{cases} \frac{[\tau_{ij}(t)]^{\alpha(t)} \cdot [\eta_{ij}(t)]^{\beta(t)}}{\sum_{s \in allowed_k} [\tau_{is}(t)]^{\alpha(t)} \cdot [\eta_{is}(t)]^{\beta(t)}}, & \text{if } j \in allowed_k \\ 0, & \text{otherwise} \end{cases} \\ s.t. \ \eta_{ij} = \frac{1}{D(i,j)} \end{cases} \tag{2}$$

In Equation (2), η_{ij} is the heuristic information, which indicates the desired degree of transferring from city i to j, and α, β reflect the number of pheromones accumulated by the ants during the movement process and the importance of the heuristic information in the ant's choice of path, respectively. At this time, the set of cities that the k-th ant is allowed to choose next can be expressed as:

$$allowed = \{C - tabu_k\} \tag{3}$$

To avoid the infinite accumulation of pheromones, the updating rules of pheromone concentration on each path after an ant has traveled all cities is as follows:

$$\begin{cases} \tau_{ij}(t+1) = (1-\rho) \cdot \tau_{ij}(t) + \Delta\tau_{ij}(t), \ \rho \in (0,1) \\ \Delta\tau_{ij}(t) = \sum_{k=1}^{m} \Delta\tau_{ij}^k(t) \end{cases} \tag{4}$$

In Equation (4), ρ is the volatilization factor of the pheromone and $\tau_{ij}(t)$ is the pheromone concentration on path ij in this iteration, the initial moment $\tau_{ij}(t) = 0$, $\Delta\tau_{ij}(t)$ is the sum of pheromone concentrations released by all ants on path ij in this iteration, and $\Delta\tau_{ij}^k(t)$ denotes the number of pheromones released by the k-th ant when it passes through path ij in this iteration.

The ant colony algorithm can be categorized into three classical models, which are called ant quantity system, ant density system, and ant cycle system, and their differences lie in the different expressions for the calculation of $\Delta\tau_{ij}^k(t)$. In the models of the

ant quantity system and ant density system, the calculation method of $\Delta\tau_{ij}^{k}(t)$ follows Equations (5) and (6), respectively:

$$\Delta\tau_{ij}^{k}(t) = \begin{cases} \frac{Q}{d_{ij}}, \text{ when } k-\text{th ant passes through the path } ij \\ 0, \text{ otherwise} \end{cases} \tag{5}$$

$$\Delta\tau_{ij}^{k}(t) = \begin{cases} Q, \text{ when } k-\text{th ant passes through the path } ij \\ 0, \text{ otherwise} \end{cases} \tag{6}$$

In the above equations, Q denotes the pheromone increase intensity coefficient, and L_k is the total length of the path passed by the k-th ant during this iteration. In addition, the calculation method of $\Delta\tau_{ij}^{k}(t)$ follows Equation (7) in the ant cycle system model:

$$\Delta\tau_{ij}^{k}(t) = \begin{cases} \frac{Q}{L_k}, \text{ when } k-\text{th ant passes through the path } ij \\ 0, \text{ otherwise} \end{cases} \tag{7}$$

In the above three models, the first two utilize local information, that is, the ants update the pheromones on the path after each step, while the latter utilizes global information, that is, the ants update all the pheromones on the path after completing a cycle, which has better practical results. Usually, the ant cycle system is used as the basic model. The complexity of this algorithm is $O(nc{\cdot}m{\cdot}n^2)$ according to the theory of algorithm complexity analysis, where nc represents the number of iterations.

3.3. Improvements in State Transfer Rules

In the ant state transfer rules, the weight factors α, β control the weight relationship between the role of heuristic information and the pheromone concentration, but the α, β in the traditional ACO are fixed during each iteration. Indeed, in the real biological world, the sensitivity of organisms to external stimuli changes over time. Thinking further from this principle, dynamically adjusting the weighting factors α and β during the iteration of the proposed algorithm helps to improve the diversity of path selection. At the beginning of the iteration of the algorithm, the concentration of pheromone on each path is low, and at this time the pheromone weighting factor dominates the path selection. In the experiments, α can be pre-set to be a larger value and β to be a smaller value, and as the iteration continues, the pheromone concentration on certain paths accumulates. To avoid falling into the local optimal solution, the value of the weighting factor α can be gradually reduced and the value of β can be increased, which can accelerate the convergence speed of the AACO-LST while expanding the search space of the solution.

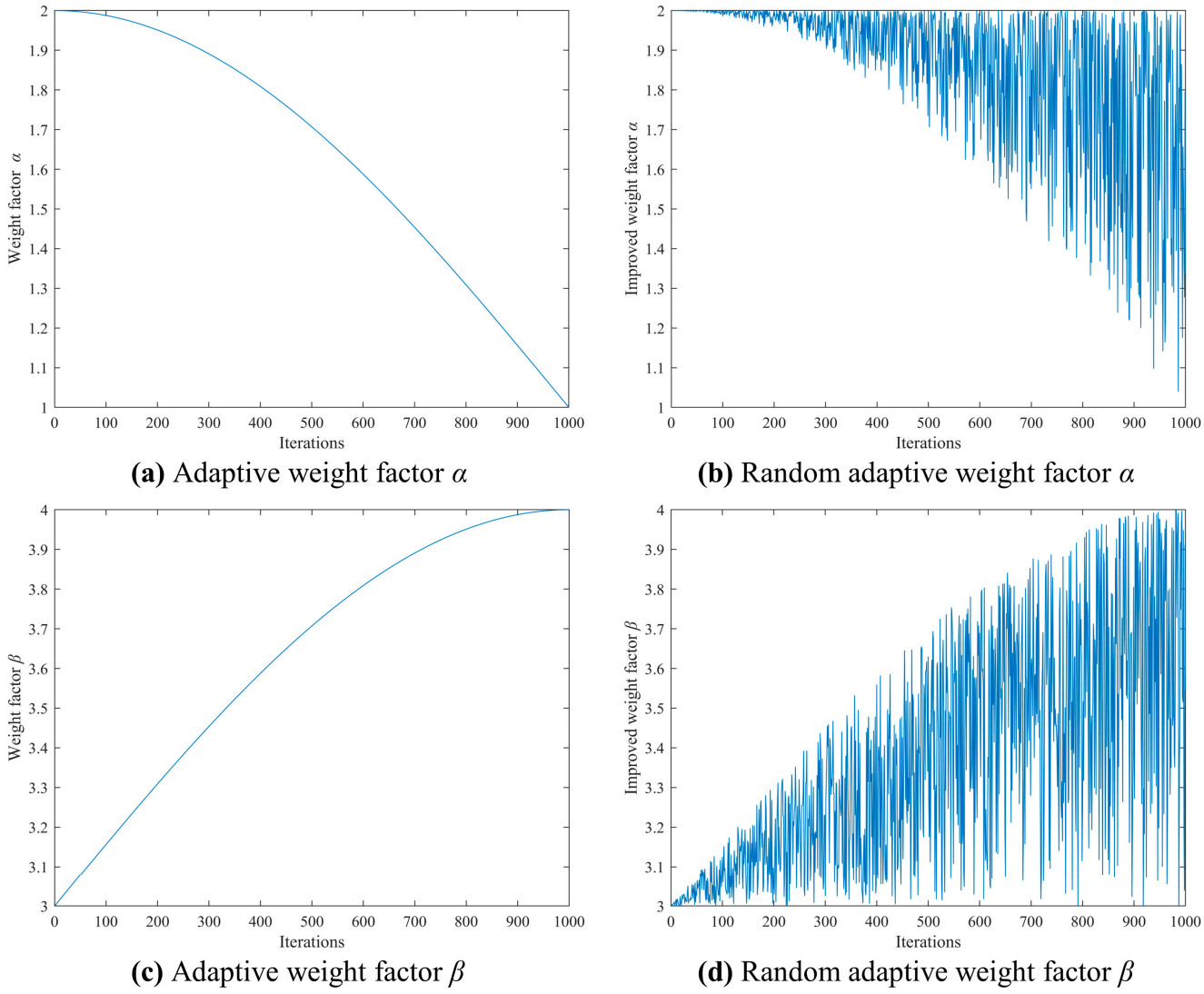
To make the changes in the values of the weighting factors α, and β smoother and more stable, some researchers have considered adaptive adjustment with the sine-cosine functions. As shown in Figure 1a,c, this article proposes a stochastic adaptive adjustment method based on the randomness as shown in Figure 1b,d, and the improved rules as follows:

$$\alpha(nc) = \cos\left(\frac{r_1{\cdot}nc{\cdot}\pi}{2{\cdot}nc_{\max}}\right) + A \tag{8}$$

$$\beta(nc) = \sin\left(\frac{r_2{\cdot}nc{\cdot}\pi}{2{\cdot}nc_{\max}}\right) + B \tag{9}$$

In the Equations (8) and (9), r_1 and r_2 are random numbers between [0, 1], and $nc_{\max}$ is the maximum number of iterations, A and B are experimentally given values. In this article, we set A = 2, B = 3, so the improved transfer probability can be calculated by:

$$P_{ij}^{k}(t) = \begin{cases} \frac{[\tau_{ij}(t)]^{\alpha(t)}{\cdot}[\eta_{ij}(t)]^{\beta(t)}}{\sum_{s\in allowed_k}[\tau_{is}(t)]^{\alpha(t)}{\cdot}[\eta_{is}(t)]^{\beta(t)}}, \text{ if } j \in allowed_k \\ 0, \text{ otherwise} \end{cases} \tag{10}$$



(a) Adaptive weight factor α **(b)** Random adaptive weight factor α

(c) Adaptive weight factor β **(d)** Random adaptive weight factor β

Figure 1. A graph showing the comparison of weighting factors.

3.4. Local Optimization Strategy

Due to the slow convergence speed of the ACO, long optimization time, and easy to premature stagnation defects, AACO-LST adopts a *k*-opt local optimization operator for its local optimization and is currently used in the local search of improved ant colony algorithm is generally a 2-opt or 3-opt. The 2-opt is the two-element optimization operator which chooses two cities of the original solution and exchanges them to try to get a new solution with a shorter path length, which takes the order of the new path of the new solution if it is good, and keeps the original solution if it is bad. Although experimental studies show that the solution quality of the 3-opt operator is higher than that of the 2-opt operator, AACO-LST adopts the 2-opt operator for local optimization after considering the computational quantity and time cost of the algorithm.

The time complexity of the 2-opt operator is $O(n^2)$, which is lower compared to the time complexity of the 3-opt operator. However, for solving the TSPs, it is obvious that 2-opt local optimization of the paths of all ants in each iteration would add too much computational effort, and the optimal paths are only a few. To increase the convergence speed of the proposed algorithm, the path lengths of ants are sorted in ascending order.

Only the top $\lambda \cdot m$ ranked ant paths are locally optimized, where λ is a control parameter for the number of ants obtained after multiple experiments. To further improve the solution quality of AACO-LST and avoid premature stagnation, a simplified 2-opt operator with a

time complexity of $O(n)$ is used in the proposed algorithm. Let C be $C_0, C_1, C_2, \ldots, C_{n-1}$ of the set, the algorithm begins with $i = 0$. If i is greater than n, then let $n = i\%n$. At this point, i is a positive value not greater than n, and the optimization rule is as follows:

$$\begin{cases} \text{replace}(C_{i+1}, C_{i+2}), \text{ if } L_{i+1} > L_{i+2} \\ \text{remain}(C_{i+1}, C_{i+2}), \text{ otherwise} \end{cases} \tag{11}$$

$$\begin{cases} L_{i+1} = D(i, i+1) + D(i+2, i+3) \\ L_{i+2} = D(i, i+2) + D(i+1, i+3) \end{cases} \tag{12}$$

The local optimization of this path ends when all n cities are traversed. According to the above optimization rules, the optimal solution of a certain iteration of example berlin52 is used as the original solution. The path nodes constructed by the ant colony algorithm, as shown in Figure 2a, exhibit intersections, resulting in a path length of 7639.31. By implementing local optimization, the cross-interference can be eliminated. The optimized result is shown in Figure 2b. The path length is currently 7544.37, which indicates that the above rules significantly enhance the precision in solving larger-scale TSP.

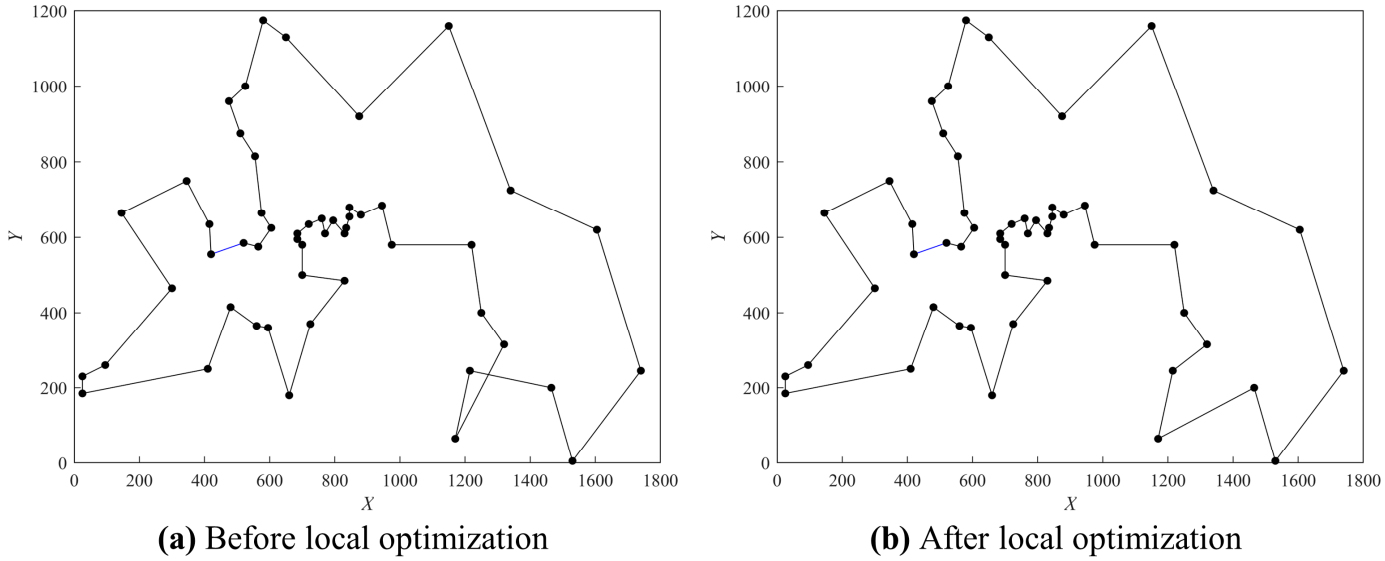


Figure 2. Comparison of local optimization results.

3.5. Adaptive Pheromone Update Rules

The updating rules of pheromones directly affects the overall performance of the ACO, and this article improves both the methods of calculation of local pheromones and global pheromones. First of all, to avoid too many residual pheromones affecting the possibility of the subsequent ants exploring other paths, when the ants pass through the path ij, the pheromone on this path will be updated locally as follows:

$$\tau_{ij}(t) = (1 - \varepsilon) \cdot \tau_{ij}(t) + \frac{\varepsilon}{m \cdot L_{nn}} \tag{13}$$

In the above equation, $\varepsilon \in (0, 1)$ is the volatilization factor of the local pheromone, m is the number of ants, and L_{nn} is the length of the path constructed by the greedy algorithm, the initial value of the pheromone obtained by using the inverse of the two has a better population diversity, which is conducive to obtaining the global optimal solution.

Second, to further expand the search space of the solution and improve the synergy between ants, AACO-LST combines with the ant system-based AS_{rank} to improve the pheromone updating rules when all the ants have completed the construction of paths within the current iteration. The ants are ranked in an ascending order based on the lengths

of their paths, and only some of the best $\lambda \cdot m$ ants are allowed to carry out the global pheromone update, the update rules are as follows:

$$\tau_{ij}(t+1) = (1-\rho)\cdot\tau_{ij}(t) + \rho\cdot\sum_{k=1}^{\lambda m}(\lambda m - rank(k) + 1)\cdot\Delta\tau_{ij}^{k} \tag{14}$$

$$\Delta\tau_{\mathrm{ij}}^{k}(t) = \begin{cases} \frac{Q}{L_k},\ rank(k) \le \lambda\cdot\mathrm{m} \\ 0,\ \text{otherwise} \end{cases} \tag{15}$$

In Equations (14) and (15), ρ is the volatilization factor of the global pheromone, $rank(k)$ is the rank of the ants after ascending order, and the magnitude of the weight of the pheromone released by the ants on L_k is $(\lambda \cdot m - k + 1)$. The more highly ranked ants release greater amounts of pheromone, and the weights acted as an amplifier of the differences in pheromone concentration across paths. To achieve a better balance between the global and local optimization abilities, an adaptive approach was used by AACO-LST to adjust ρ. When ρ is too large, the likelihood of previously searched paths being selected again is too high, too, which affects the randomization and global search ability of the proposed algorithm; Conversely, by decreasing ρ, although the stochastic performance and global search ability of the algorithm can be improved, the convergence speed of the algorithm will again be reduced. Therefore, ρ is set to the maximum value ρ_0 at the beginning of the iteration, while positive pheromone feedback dominates. If the algorithm obtains the same optimal result for a certain number of loops, it means that the algorithm is trapped in a local optimal solution, and the value of ρ should be reduced, which is calculated as follows:

$$\rho(nc+1) = \begin{cases} \rho_0,\ nc < \omega \cdot nc_{\max} \\ \gamma \cdot \rho(nc),\ nc > \omega \cdot nc_{\max}\ and\ s > s_0 \end{cases} \tag{16}$$

In the above equation, $\gamma \in (0, 1)$ is the decreasing factor of the global pheromone, ω is the control parameter obtained after multiple experiments, s is the number of times the optimal solutions of each generation are consecutively equal, and s_0 is a pre-set integer greater than 1.

3.6. The Algorithm Process of AACO-LST

The specific steps of the proposed algorithm are as follows. Correspondingly, the algorithm flowchart of AACO-LST is shown in Figure 3. The proposed algorithm will perform a path search according to the below process until the stop condition is reached. Based on the pseudocode of AACO-LST, it can be observed that the algorithm maintains a time complexity of $O(nc{\cdot}m{\cdot}n^2)$. Like the basic ACO, the local optimization strategy uses a simplified 2-opt operator, whose time complexity is just $O(n)$ and only partially optimizes the paths of the optimal ants locally. Therefore it will not increase the complexity of the algorithm and can effectively save the time cost. The space complexity of the algorithm is the same as the basic ACO, depending on the pheromone matrix, the path matrix stored by the ants, the space complexity of the pheromone matrix is $O(n^2)$ and the complexity of the path matrix stored by the ants is $O(m{\cdot}n)$, so the total space complexity of AACO-LST is $O(n^2 + m{\cdot}n)$. The algorithm process of AACO-LST is shown in Algorithm 1.

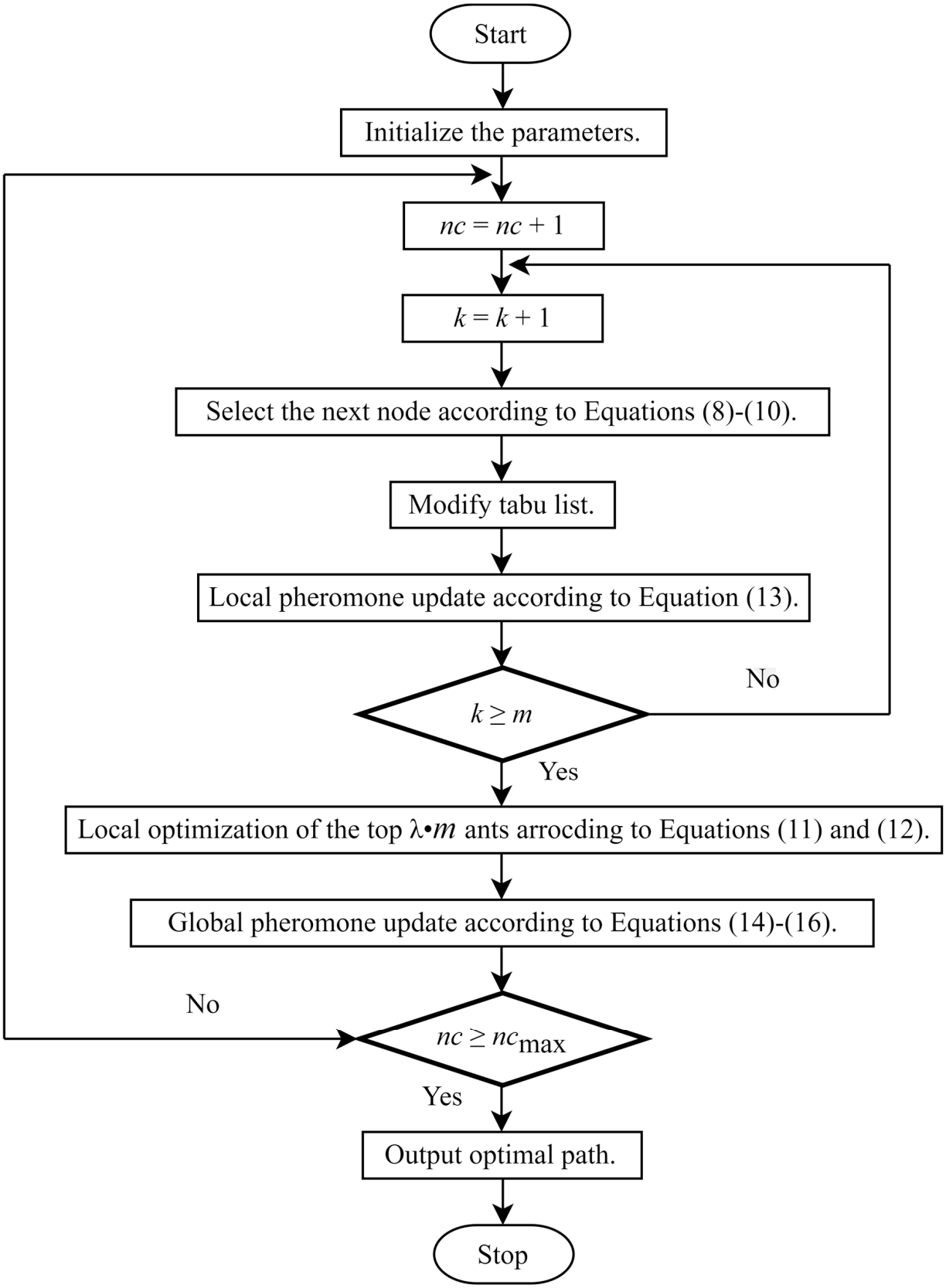


Figure 3. The algorithm flowchart of AACO-LST.

Algorithm 1: AACO-LST

Input : The maximum iteration number $nc_{\max}$, the volatility factor ε, λ, Q, $\rho 0$, ω.
Output : The obtained optimal path R_{best} and the optimal solution L_{best}.
Initialization : Initialize the pheromone matrix.
Place m ants in the set C consisting of n cities.
$nc \leftarrow 0$;
while $nc < nc_{\max}$ do
    for $k = 1$ to m do
        Ant chooses the next city according to the improved transfer probability and completes their respective tours.
        for $i = 1$ to n do
            Select the next node according to Equations (8)–(10);
            Modify the list of *tabu*;
            Local pheromone update according to Equation (13).
        end
    end
    Calculate the length L_k of the tour constructed by the k-th ant;
    Local optimization of the top $\lambda \cdot m$ ants according to Equations (11) and (12);
    for each edge do
        Global pheromone update according to Equations (14)–(16).
    end
    $nc \leftarrow nc + 1$;
end
Find the optimal path R_{best} and the optimal solution L_{best} generated in the iterative process;
return R_{best} and L_{best}

4. Simulation Experiments and Results

To verify the feasibility of AACO-LST, 45 instances of TSP using Euclidean distance were selected from the TSP standard test library (TSPLIB) for testing. The simulation experiments were programmed using the MATLAB software with the version of R2020b, MathWorks – Apple Hill Campus, MA 01760-2098, USA. Meanwhile, the operating system of Microsoft Windows 10 Pro 64-bit and 8GB of RAM were used in the experiments. All experiments were completed on the Intel (R) Core (TM) i7-10700 CPU in Jingdezhen, China, while the processor is produced by Intel.

4.1. Ablation Experiments

To fully demonstrate the impact of AACO-LST design and improved modules on the effectiveness of solving TSPs, this section conducts ablation experiments to validate the efficacy of each module. The settings of some of the key parameter parameters of the experiments, such as ant number m, pheromone enhancement factor Q, and pheromone volatilization factor ρ, are inspired by the work of Du, P.-Z. et al. in [45]. The algorithm proposed by them achieves good results in similar environments and shows that proper tuning of these parameters can improve the performance of the algorithm. Therefore, to adapt the algorithm of this article, the values of some parameters were optimized by grid search based on reference [45], and the finalized experimental parameter settings are shown in Table 1.

In Table 1, ACO refers to the standard algorithm of ACS, ACO-ISTR refers to the ACS with improved state transition rules, and ACO-LOS-AP refers to the ACS with added local optimization strategy and adaptive pheromone updating rules. In fact, adaptive rules for updating pheromones are introduced, which guides ants in updating their pheromones after on partial optimal ant paths. Therefore, the local optimization strategy and adaptive pheromone updating rules essentially function as a module and cannot be tested separately. The instances eil51, st70, eil76, and rat99 are selected for 10 experiments each in

the TSP standard test library TSPLIB. Each experiment consisted of 1000 iterations. The experimental results are presented in Table 2. In each column of Table 2, *BKS* represents the known optimal solution for each instance, *Best* denotes the algorithm's best solution obtained, *Avg* represents the average solution obtained from multiple algorithm tests, and *Std* is the standard deviation.

Table 1. Parameter list of comparison algorithms.

Algorithm	Parameter
ACO	$m = 1.5n$, $\alpha = 2$, $\beta = 4$, $\varepsilon = 0.1$, $\rho = 0.3$, $Q = 100$
ACO-ISTR	$m = 1.5n$, $\varepsilon = 0.1$, $\rho = 0.3$, $Q = 100$
ACO-LOS-AP	$m = 1.5n$, $\alpha = 2$, $\beta = 4$, $\varepsilon = 0.1$, $\lambda = 0.1$, $\rho_0 = 0.3$, $\omega = 0.7$, $s_0 = 30$, $\gamma = 0.8$, $Q = 100$
AACO-LST	$m = 1.5n$, $\varepsilon = 0.1$, $\lambda = 0.1$, $\rho_0 = 0.3$, $\omega = 0.7$, $s_0 = 30$, $\gamma = 0.8$, $Q = 100$

Table 2. Results of ablation experiments.

TSP (*BKS*)	Algorithm	*Best*	*Avg*	*Std*
eil51 (426)	ACO	429.53	433.62	4.30
	ACO-ISTR	428.98	431.70	3.41
	ACO-LOS-AP	428.98	430.25	2.78
	AACO-LST	428.87	429.88	1.83
st70 (675)	ACO	681.22	687.81	7.68
	ACO-ISTR	678.62	684.70	6.10
	ACO-LOS-AP	680.50	682.80	4.46
	AACO-LST	677.11	680.83	2.87
eil76 (538)	ACO	547.40	558.56	6.50
	ACO-ISTR	546.39	554.95	6.10
	ACO-LOS-AP	545.39	549.32	3.70
	AACO-LST	544.37	548.75	2.71
Rat99 (1211)	ACO	1238.48	1295.16	15.76
	ACO-ISTR	1223.80	1243.11	15.23
	ACO-LOS-AP	1220.36	1230.13	8.62
	AACO-LST	1219.24	1225.07	5.21

According to the results of the four TSP instances shown in Table 2, ACO-ISTR and ACO-LOS-AP outperform ACO in obtaining optimal and average solutions. This demonstrates that improved state transition rules can expand the search space for solutions, helping ants achieve a balance between exploration and exploitation. The combination of local optimization strategies and adaptive pheromone updating rules can effectively enhance the accuracy of the solution. Each improvement method can contribute to enhancing ACO performance. The *Avg* and *Std* of ACO-LOS-AP are superior to ACO-ISTR, indicating that combining a local optimization strategy with adaptive per-update rules has a significantly better effect on global search than improving state transition rules. The optimal and average solutions obtained by AACO-LST are higher than those of ACO-ISTR and ACO-LOS-AP indicating that the integration of various improvement methods leads to better results compared to individual enhancement approaches. In particular, AACO-LST demonstrates better stability in solving the standard deviation of various instances compared to individual improvement methods. From this perspective, it also demonstrates that AACO-LST exhibits superior overall performance and represents an effective enhancement of the standard ACO.

4.2. Comparison of Solution Quality

A series of TSP instances were selected for two reasons in current experiments. Firstly, TSP instances used in the experiment were adapted from reference [45], which investigated the application of an object-oriented multi-role ant colony optimization algorithm in solving

the TSPs. Therefore, these instances are reused in this article. Secondly, they cover a range of TSP instances with varying sizes, which helps to better demonstrate the effectiveness of solving TSP of different scales.

In this stage of the experiments, two algorithms, ACS and AACO-LST, are used respectively to solve 45 TSP instances 30 times for comparison, and each time 1000 iterations are performed. The experimental parameters are the same as those in Section 4.1. The experimental results are shown in Table 3. In the columns of Table 1, *Dev* represents the degree of deviation between the optimal solution obtained by the algorithm and the known optimal solution, which can be calculated by Equation (17). As can be seen from Table 1, among the 45 TSP instances solved by AACO-LST, except for the instance burma14, which is of a very small scale and achieves the same solution quality as ACS, the solution quality of the remaining 44 instances is significantly better than ACS. The average *Dev* of AACO-LST for solving 45 instances is 1.44%, while ACS achieves an average *Dev* of 7.02%. In other words, AACO-LST improves the solution quality over ACS by 79%. In comparison to instances such as oliver30, berlin52, st70, kroA100, etc., AACO-LST's obtained optimal solutions are slightly worse than the known optimal solutions. This is due to the limited precision of floating-point numbers stored in computers. For the same optimal path, calculating it from different starting points may result in slightly different lengths. Furthermore, some of the known optimal solutions in Table 1 are integers, slightly smaller than the floating-point solutions. Given these reasons, the solutions with *Dev* below 0.5% in some instances in Table 1 have reached the optimum. Out of the 45 test cases, AACO-LST achieves *Dev* below 0.5% in 21 sets, accounting for a high proportion of 47%; while ACS has only 2 sets of instances with *Dev* below 0.5%, accounting for 9%. If reaching the known optimal solution is taken as the criterion, the solution accuracy of AACO-LST is more than 5 times that of ACS. As shown in Figure 4, in all 45 instances, the *Dev* of AACO-LST is consistently lower than ACS. Specifically, the *Dev* of AACO-LST has more than 15% on instances gr229 and gr434, and the *Dev* of ACS has more than 27% on gr229 and gr431, which indicates that the solution accuracy of AACO-LST is significantly superior to ACS. To summarize, AACO-LST is far superior to ACS in terms of the quality of solutions obviously.

$$Dev = \left(\frac{Best - BKS}{BKS} \right) \times 100\% \tag{17}$$

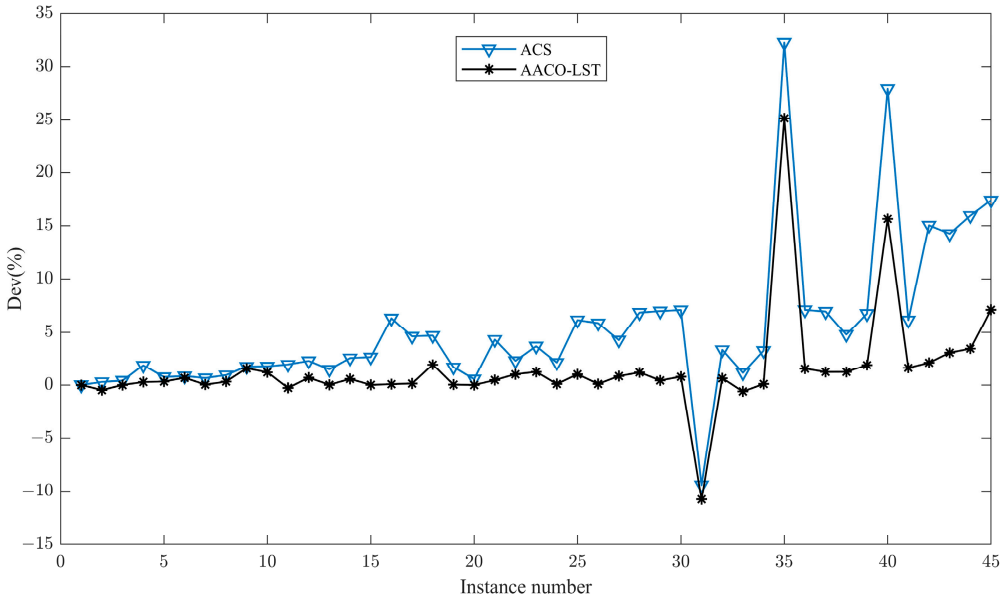


Figure 4. Comparison of solution quality deviation.

Table 3. Comparison of algorithmic solution.

Tag	TSP	*BKS*	*Best*		*Dev* (%)	
			ACS	AACO-LST	ACS	AACO-LST
1	burma14	30.88	30.88	30.88	0	0
2	ulysses22 *	75.67	75.88	75.31	0.28	−0.47
3	bays29	9074.15	9111.6	9074.15	0.41	0
4	oliver30	423.74	425.08	424.87	1.84	0.27
5	att48	33,523.71	33,772.05	33,523.71	0.74	0.32
6	eil51	426	429.53	428.87	0.83	0.67
7	berlin52	7542	7590.07	7544.37	0.64	0.03
8	st70	675	681.22	677.11	0.92	0.31
9	pr76	108,159.4	110,032.63	109,840.83	1.73	1.55
10	eil76	538	547.4	544.37	1.75	1.18
11	gr96 *	512.31	522.18	510.89	1.93	−0.28
12	rat99	1211	1238.48	1219.24	2.27	0.68
13	kroA100	21,282	21,593.58	21,285.44	1.46	0.02
14	kroB100	22,141	22,704.05	22,263.98	2.54	0.56
15	kroC100	20,749	21,294.39	20,750.76	2.63	0.01
16	kroD100	21,294	22,640.24	21,309.58	6.32	0.07
17	kroE100	22,068	23,085.56	22,098.13	4.61	0.14
18	eil101	629	658.44	640.98	4.68	1.9
19	lin105	14,379	14,618.75	14,383	1.67	0.03
20	pr107 *	44,303	44,541.26	44,301.68	0.54	−0.003
21	bier127	118,282	123,365.27	118,816.98	4.3	0.45
22	ch130	6110.86	6250.02	6171.96	2.28	1
23	gr137	698.53	724.14	707.02	3.67	1.22
24	ch150	6528	6665.42	6533.81	2.11	0.09
25	kroA150	26,524	28,155.86	26,787.78	6.15	0.99
26	kroB150	26,130	27,660.02	26,156.38	5.86	0.1
27	rat195	2323	2421.73	2342.13	4.25	0.82
28	d198	15,780	16,861.08	15,965.24	6.85	1.17
29	kroA200	29,368	31,419.63	29,491.02	6.99	0.42
30	kroB200	29,437	31,523.68	29,662.52	7.09	0.77
31	gr202 *	550	498.18	490.74	−9.42	−10.77
32	ts225	126,643	130,864.42	127,441.15	3.33	0.63
33	tsp225 *	3916	3960.36	3892.06	1.13	−0.61
34	pr226	80,369	82,983.87	80,440.1	3.25	0.09
35	gr229	1346.02	1779.42	1684.5	32.2	25.15
36	gil262	2378	2546.8	2414.13	7.1	1.52
37	lin318	42,029	44,949.7	42,371.29	6.95	1.21
38	rd400	15,281	16,011.51	15,467.56	4.78	1.22
39	fl417	11,861	12,661.42	12,081.41	6.75	1.86
40	gr431	1714.14	2192.29	1982.37	27.89	15.65
41	pr439	107,217	113,797.02	108,937.9	6.14	1.61
42	pcb442	50,778	58,410.35	51,837.94	15.03	2.09
43	d493	35,002	39,983.51	36,074.31	14.23	3.06
44	u574	36,905	42,789.92	38,169.74	15.95	3.43
45	vm1084	239,297	280,979.88	256,344.84	17.42	7.12

* AACO-LST has a negative *Dev* on those instances, which indicates that the optimal solution obtained by AACO-LST is not only better than that obtained by ACS but also better than that of the known optimal solution.

As shown in Figure 5, the optimal solutions derived from various sizes of TSPs are computed using AACO-LST, and the quality of these solutions can be evaluated based on the results, and the red dots represent cities, while the blue lines represent paths between cities. The flower pollination algorithm (FPA) is an optimization algorithm that simulates the pollination behavior of flowers in nature. On this basis, a new hybridization scheme based on FPA, ACO with local search, ant supervised by flower pollination with local search (ASFPA-Ls) proposed in the reference [46] incorporates a local search strategy in the algorithm as does AACO-LST to avoid falling into a local optimum, and this article

compares it with ASFPA-Ls alone, as shown in Table 4, where *Err* represents the relative error between the average solution and the known optimal solution, as shown in the Equation (18). It can be seen in Table 4 that the adaptive dynamic adjustment of AACO-LST is good, and *Avg* and *Err* are significantly better than the optimal solutions obtained from the comparison algorithm, which indicates that AACO-LST has good robustness.

$$Err = \left(\frac{Avg - BKS}{BKS}\right) \times 100\% \tag{18}$$

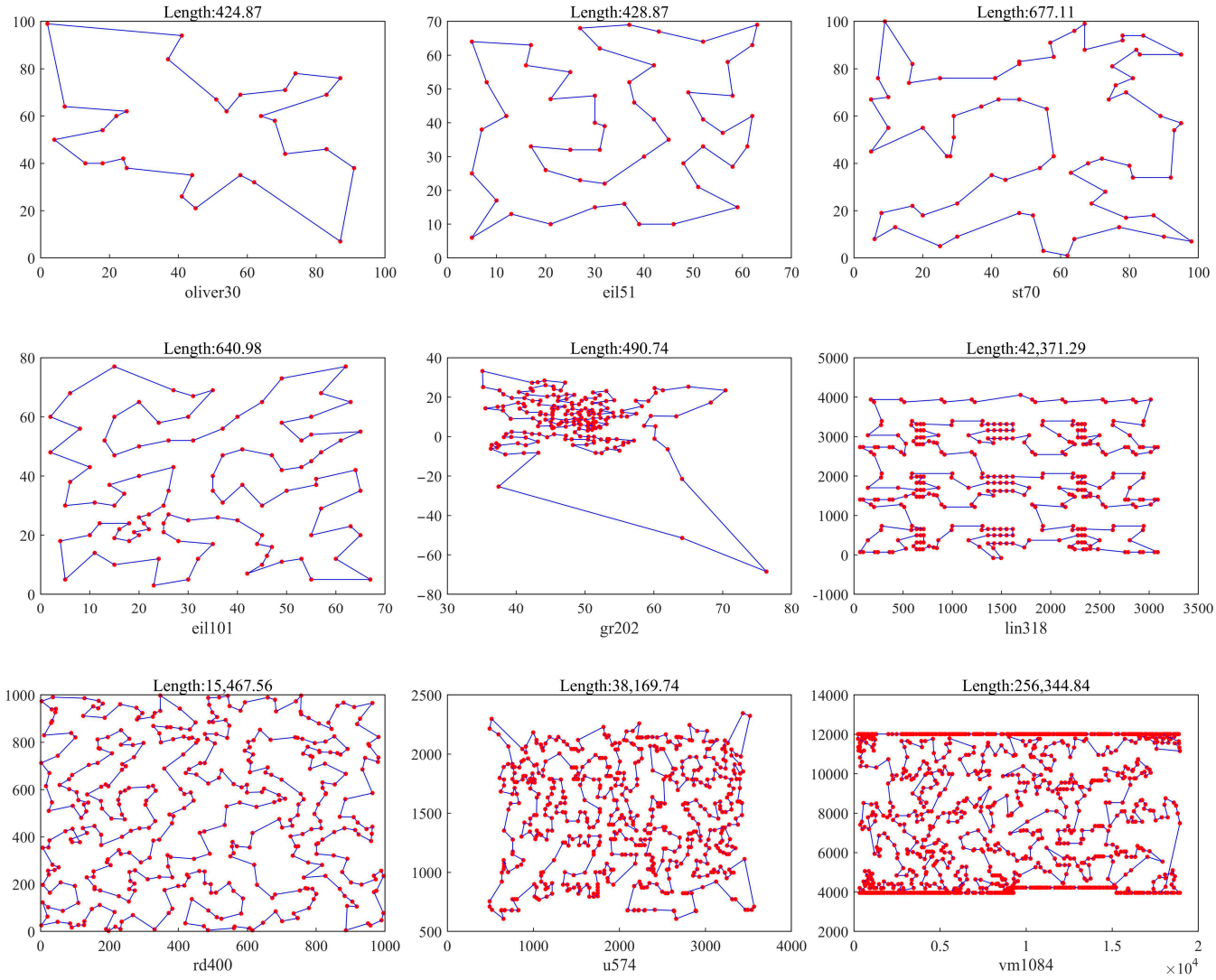


Figure 5. The optimal path obtained by AACO-LST in each instance.

The tree-seed algorithm (TSA) algorithm is an optimization algorithm that simulates the reproduction of trees by generating seeds. On this basis, the TSA is redesigned by integrating the swap, shift, and symmetry transformation operators (DTSA) [47] was proposed and applied in the comparative experiment of this article. Based on the comparison with ACS and ASFPA-Ls, AACO-LST is compared horizontally with multi-subdomain grouping-based particle swarm optimization (MSGPSO) [48], multi-domain inversion-based algorithm (MDIGA) [49], DTSA [47], improved greedy algorithm (IMGRA) [50], improved genetic algorithm based on soft actor-critic model (GA-SAC) [51], and simulated annealing algorithm with large neighborhood search (SALNS) [52] concerning solving the TSP of different sizes, and the solution quality results are shown in Tables 5 and 6. *PE* denotes the relative error between the optimal and average solutions obtained by each

algorithm, as shown in Equation (19) [26]. As can be seen from Tables 5 and 6, *Err* does not exceed 2% in all instances tested, which indicates that AACO-LST can effectively find high-quality solutions, and *PE* does not exceed 1%, which suggests that AACO-LST has high stability. In addition, for small-scale TSPs, AACO-LST has a slightly worse average than some algorithms (SALNS and GA-SAC), but as the size of the TSP increases, such as in examples tsp225 and lin318, AACO-LST has a better average than other algorithms, which suggests that AACO-LST has some advantages for solving large-scale TSPs.

$$PE = \left(\frac{Avg - Best}{Best} \right) \times 100\% \tag{19}$$

Table 4. Longitudinal comparison of ASFPA-Ls and AACO-LSTs.

TSP (*BKS*)	Algorithm	*Avg*	*Err* (%)
eil51 (426)	ASFPA-Ls	437.13	2.61
	AACO-LST	429.88	0.9
st70 (675)	ASFPA-Ls	768.00	2.84
	AACO-LST	683.78	1.3
eil76 (538)	ASFPA-Ls	620.00	5.26
	AACO-LST	549.40	2.1
Rat99 (538)	ASFPA-Ls	1271.22	4.95
	AACO-LST	1229.68	0.68
kroA100 (21,282)	ASFPA-Ls	22,003.405	3.53
	AACO-LST	21,416.6	0.63

Table 5. Horizontal comparison in kroA100.

TSP (*BKS*)	Algorithm	*Best*	*Avg*	*Err* (%)	*PE* (%)
kroC100 (20,749)	MSGPSO	21,161.51	22,657.77	1.99	7
	MDIGA	21,394.58	23,138.13	3.11	8
	DTSA	-	21,506.78	1.06	-
	IMGRA	22,506.83	-	-	-
	AACO-LST	20,750.76	20,927.37	0.86	0.85

Table 6. Horizontal comparison of algorithms.

TSP (*BKS*)	Algorithm	*Best*	*Avg*	*Err* (%)	*PE* (%)
eil51 (426)	SALNS	426	426	0	0
	GA-SAC	426	427.40	0.33	0.33
	IMGRA	429.53	-	-	-
	AACO-LST	428.87	430.80	1.13	0.45
st70 (675)	SALNS	675	675	0	0
	GA-SAC	675	678.90	0.58	0.58
	IMGRA	725.51	-	-	-
	AACO-LST	677.11	683.78	1.3	0.99
rat99 (1211)	SALNS	1211	1212	0.08	0.08
	GA-SAC	1211	1222.45	0.95	0.95
	IMGRA	1313.76	-	-	-
	AACO-LST	1219.24	1225.79	1.22	0.54
tsp225 (3916)	SALNS	3896	3910	−0.15	0.36
	GA-SAC	3900	3918.60	0.07	0.47
	IMGRA	4170	-	-	-
	AACO-LST	3892.06	3906.42	−0.24	0.37
lin318 (42,029)	SALNS	-	-	-	-
	GA-SAC	42,329	42,902.40	2.08	1.35
	IMGRA	45,036.36	-	-	-
	AACO-LST	42,371.29	42,726.78	1.66	0.84

Simulations based on 45 TSP instances show that the average *Dev* is 1.44% for AACO-LST and 7.02% for ACS, i.e., AACO-LST improves the solution quality by 79% compared to ACS. In the longitudinal comparison with ASFPA-Ls, the average *Err* is 3.84% for ASFPA-Ls, but 1.12% for AACO-LST, which is much lower than ASFPA-Ls. Further horizontal comparison with the algorithms, the *PE* of AACO-LST is not more than 1%. The *Err* is not more than 2%, which indicates that AACO-LST can find high-quality solutions with high stability. Finally, the convergence speed of the algorithm is tested, and the average convergence speed of the algorithm of AACO-LST has reached more than two times of ACS and particle swarm algorithm (PSO). This indicates that the proposed meta-heuristic algorithm, AACO-LST, could find a higher quality solution for large-scale TSP than other tested algorithms, with better stability and convergence.

4.3. Statistical Comparison of Solution Quality

In this section, instances eil51, eil101, gr202, and lin318 were selected from TSPLIB, the standard test library of TSPs, and 100 experiments were conducted on each of these four instances using AACO-LST and ACS for 1000 iterations each, with the same experimental parameters as in Section 4.1. The results of the 100 experiments are analyzed using the quartile method, where the values are first arranged from smallest to largest and divided into four equal parts. Then, take the positional values of the three divisions, including the lower, median, and upper quartile. The difference between the upper and lower quartiles is called the quartile spacing. The statistical results are shown in Figure 6. The solid line in the figure represents the median path length of the algorithm for each generation of optimization in 100 experiments, the upper dashed line represents the upper quartile, and the lower dashed line represents the lower quartile.

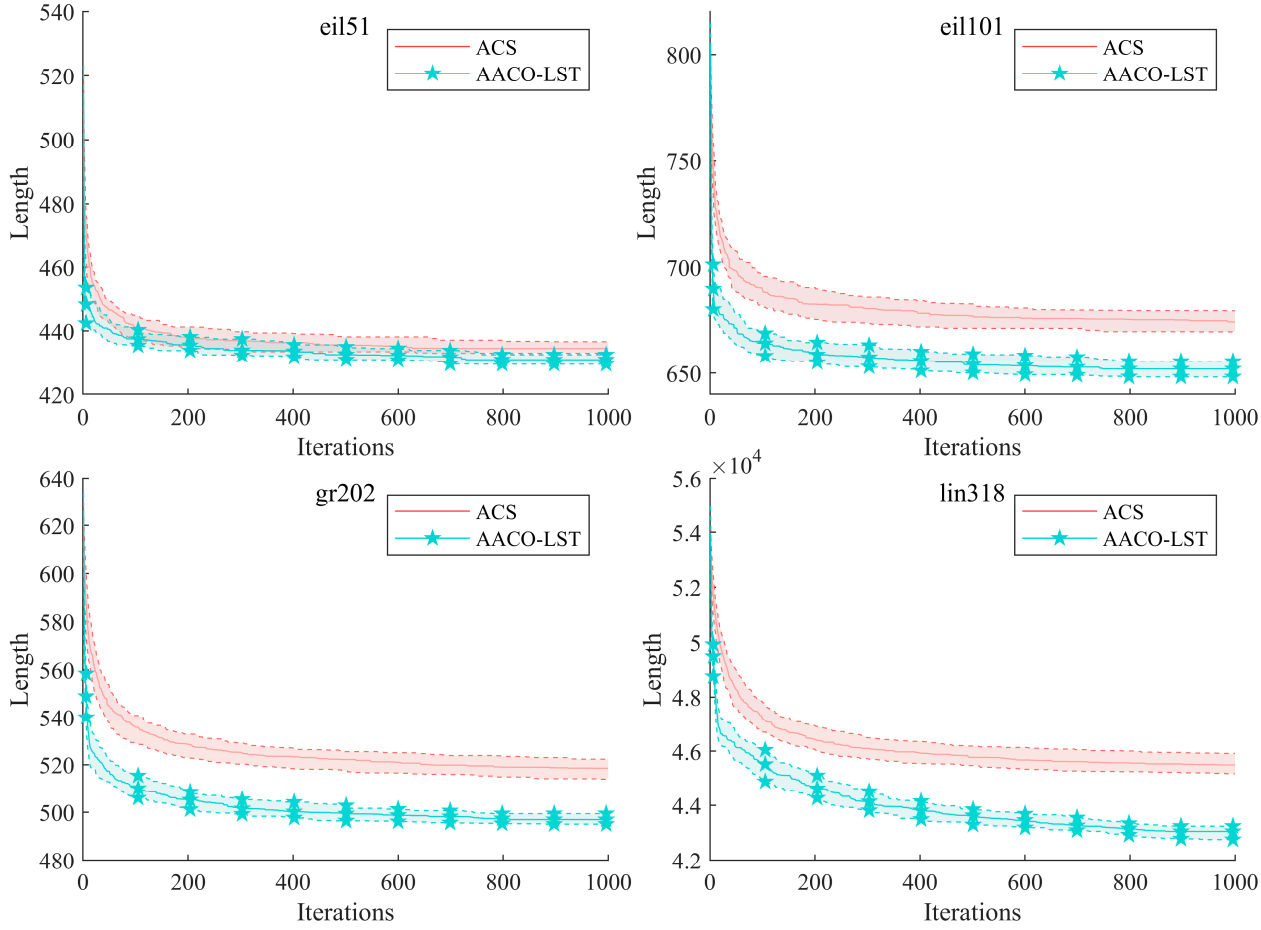


Figure 6. Statistical comparison of solution quality for each example.

As seen in Figure 5, for small-scale TSPs (e.g., eil51), the median path length of AACO-LST is only slightly better than that of ACS, and the interquartile distance is not much different. But for large-scale TSPs, it can be seen that the larger the problem size is, the better the median path length of AACO-LST is than that of ACS, which indicates that the solution quality of AACO-LST for large-scale problems is better than that of ACS, and the quartile distance is also significantly smaller than that of ACS, which indicates that the algorithm has better stability and robustness. The optimal paths of the four instances obtained are shown in Figure 6.

4.4. Convergence Speed Comparison

In this section, the classical PSO is selected to perform 100 experiments on instances eil51, eil101, gr202, and lin318 with 1000 iterations each. The convergence speed of PSO is then compared with that of ACS and AACO-LST, and a comparison of the optimal value curves for each algorithm is shown in Figure 7, which illustrates that the process of optimizing the path length with increasing iterations, and from the curve, then, the number of iterations when the algorithm reaches its optimal solution can be observed. For the eil51, PSO obtains the optimal solution 436.03 in the 554th iteration, ACS obtains the optimal solution 429.53 in the 555th iteration, and AACO-LST obtains the optimal solution 428.87 in the 189th iteration; for the eil101 example, PSO obtains the optimal solution 714.75 in the 950th iteration, and ACS obtains the optimal solution 658.44 in the 735th iteration. AACO-LST obtains the optimal solution 640.98 in the 290th iteration; for the gr202 example, PSO obtains the optimal solution 637.48 in the 829th iteration, ACS obtains the optimal solution 498.18 in the 893rd iteration, and AACO-LST obtains the optimal solution 490.74 in the 404th iteration; for the lin318 instance, PSO obtains the optimal solution 66,755.52 in the 996th iteration, ACS obtains the optimal solution 44,949.7 in the 966th iteration, and AACO-LST obtains the optimal solution 42,371.29 in the 493rd iteration; combining with Figure 5, it can be seen that the solution quality and robustness of AACO-LST are better than that of ACS, especially in these four instances. The average convergence speed of the AACO-LST algorithm has reached more than two times the convergence speed of ACS and PSO.

4.5. Population Diversity and Convergence Analysis of AACO-LST

In this section, the population diversity and convergence of the proposed algorithm will be further analyzed. That helps to understand that improving strategies have inherent impacts on the structure of simulation experiments. To analyze the level of population diversity in the AACO-LST during the process of optimization, we tracked and analyzed the population diversity, using the kroA100 instance as a representative case. At this moment, the number of cities is n = 100, the number of ants is m = 150, and the maximum iteration count of the experiment has been set to 1000. During the tracking process, the population will be saved every 200 iterations for analysis. To facilitate analysis, the population Pop_{Ite} that has undergone the *Ite*-th iteration is defined as:

$$\begin{cases} Pop_{Ite} = \bigcup_{AN=1}^{m} Dec_{AN} \\ s.t.\ Dec_{AN} = [x_1^{AN},\ y_1^{AN},\ x_2^{AN},\ y_2^{AN},\ \ldots\ ,x_n^{AN},\ y_n^{AN}] \end{cases} \tag{20}$$

In the Equation (20), m and n respectively represent the number of ants and cities, ($x_i{}^{AN}$, $y_i{}^{AN}$) represents the coordinate of the i-th city visited by the AN-th ant, and Dec_{AN} is the decision variable for the AN-th ant. To better track the extent of changes in diversity within a population, the decision variable of an individual is defined as a $2n$-dimensional vector formed by accessing the coordinates of all cities, instead of using a collection represented by the codes of the city number. To analyze the difference between individuals and the

population center, the calculation method for individual difference distance (*IDD*) is defined in the Equation (21):

$$\begin{cases} IDD_{Ite}^{AN} = \sqrt{\sum_{CN=1}^{n} \left(x_{CN}^{AN} - \overline{x_{CN}}\right)^2 + \sum_{CN=1}^{n} \left(y_{CN}^{AN} - \overline{y_{CN}}\right)^2} \\ s.t.\ PopCen_{Ite} = \text{mean}(Pop_{Ite}) = (\overline{x_1}, \overline{y_1}, \overline{x_2}, \overline{y_2}, \ldots, \overline{x_n}, \overline{y_n}) \\ \quad Pop_{Ite} = \bigcup_{AN=1}^{m} Dec_{AN} \\ \quad \overline{x_{CN}} = \frac{1}{m} \sum_{AN=1}^{m} x_{CN}^{AN} \\ \quad \overline{y_{CN}} = \frac{1}{m} \sum_{AN=1}^{m} y_{CN}^{AN} \end{cases} \tag{21}$$

In the above equation, *AN* and *CN* refer to the ant number and city number, respectively. Meanwhile, $IDD_{ite}{}^{AN}$ refers to the individual difference distance of the *AN*-th individual in the population that has undergone *Ite* iterations, ($x_{CN}{}^{AN}$, $y_{CN}{}^{AN}$) is the coordinate of the *CN*-th city visited by the *AN*-th ant, mean(·) is a calculation function that computes the mean value along columns, $PopCen_{Ite}$ is the population center at the *Ite*-th iteration. To measure the diversity of a population, the calculation process of population distribution diversity (PDD) is defined as follows:

$$PDD_{Ite} = \text{std}\left(\bigcup_{AN=1}^{m} IDD_{Ite}^{AN}\right) \tag{22}$$

In the Equation (22), PDD_{Ite} refers to the population distribution diversity at the *Ite*-th iteration, and std(·) refers to a function used for calculating standard deviation. The analysis results of the diversity within the AACO-LST population are displayed in Figure 8. In Figure 8, blue, purple, red, light yellow, and orange are used to represent the population states at 200, 400, 600, 800, and 1000 iterations, respectively. The left vertical axis refers to the value of *IDD* and a box plot is used to display the distribution of *IDD* among individuals in different iterations of the population. Meanwhile, the vertical axis on the right refers to the values of *PDD*, and a black line is used to depict the *PDD* values at different iteration counts. On one hand, during the first 80% of the iteration process, the distribution of *IDD* becomes slightly more centralized, which also indicates that the proposed algorithm accelerates convergence in the search process. In the final 20% of the process, the distribution of *IDD* becomes even more dispersed, surpassing its state at 200 iterations. This also indicates that the AACO-LST does not experience a significant decrease in population diversity as the number of iterations increases, unlike traditional PSO. On the other hand, the value of *PDD* slightly decreases during the process of evolution, but it quickly exceeds its initial state. This indicates that the AACO-LST is capable of effectively maintaining diversity levels at both the individual and population levels, leading to improved search efficiency.

In the previous experimental analysis, we tested the performance of the AACO-LST with several other comparison algorithms to verify that the proposed algorithm has a more excellent performance. In this section, we analyze the convergence of the AACO-LST in the following experiments to further explore its stability level. The optimal path lengths searched by the algorithm are analyzed under four different path lengths and the results are shown below. In the standard TSP test library, instances eil51, eil101, gr202, and lin318 are selected in this article to carry out further experiments. In Figure 9, the results of running 100 repetitions of experiments for each of these four instances using the AACO-LST are shown, where each instance is represented using a separate horizontal axis, and *OPL* denotes optimal path length. In addition, the median is labeled above each horizontal axis, while the upper and lower quartiles and the corresponding boxplots are plotted below the corresponding horizontal axis. In the three instances of eil51, eil101, and gr202, the distribution of their *OPL*s is relatively dense, and at the same time, the difference between the median and the upper and lower quartiles is closer in all three cases, which all show a close Gaussian distribution. With the increase in the number of cities, the difficulty of

the solution searching increases, at the same time, there is also a significant growth in the corresponding path length in the lin318 instance. Similarly, the distribution of its *OPL* is similar to the state of Gaussian distribution. The final results show that the distribution of path lengths in each instance shows a highly concentrated pattern, and the lengths of the intervals formed by the upper and lower quartiles are relatively short. For example, in the *OPL* set of eil51 instances, the interval composed of upper and lower quartiles is [429.48, 432.50], the median is 430.88, and the length of the interval is only 3.02. This also demonstrates that the AACO-LST has a high degree of convergence, and further proves that the algorithm has good search performance.

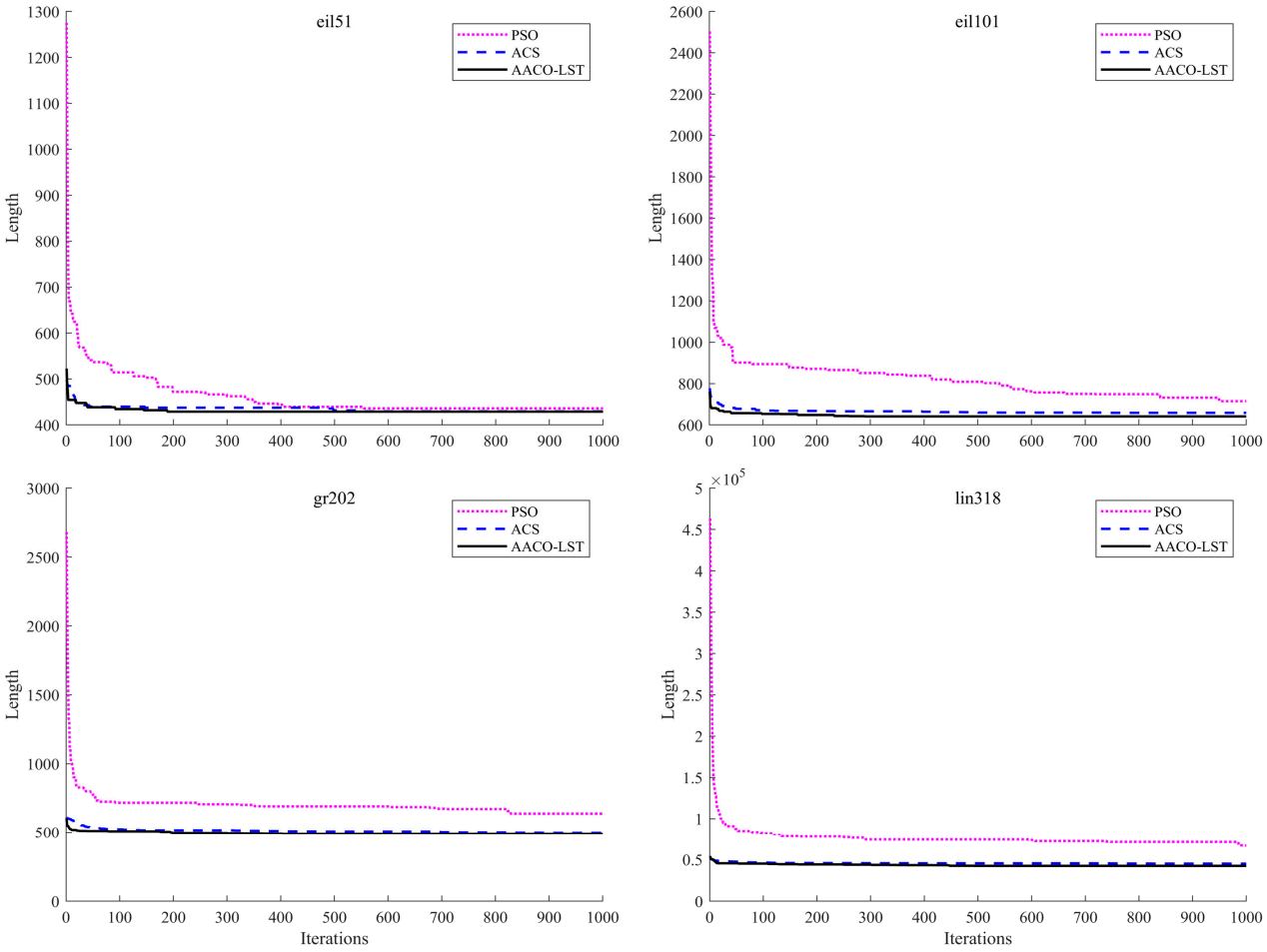


Figure 7. Comparison of optimal curves for each example.

Overall, the impact of different improvement modules on algorithm performance was tested through ablation experiments in Section 4.1. This demonstrates that each improved module has a positive impact on the performance. AACO-LST reduces the average variance of each instance by 60% compared to ACO, showcasing its excellent stability. In Section 4.2, the solution quality of ACS and AACO-LST was compared through 45 TSP instances from TSPLIB. The results show that on burma14, the smallest TSP instance, the optimal solutions of the two algorithms are the same. In addition, on the other 44 TSP instances, the optimal solutions of AACO-LST are superior to ACS. Among them, the optimal solution obtained by AACO-LST on 5 instances is even better than the known optimal solution, and the average Dev is less than 1.5%, far lower than the average Dev of ACS. In the comparison algorithms, ASFPA-Ls is an improved ant colony algorithm that integrates FPA and incorporates a local search strategy. Therefore, the ASFPA-Ls were chosen for conducting longitudinal comparative experiments in this article. The data shows that the average solution and error of AACO-LST are better than those of ASFPA-Ls. To further analyze the performance of

AACO-LST, algorithms such as MSGPSO, MDIGA, GA-SAC, IMGRA, SALNS, and DTSA were selected for comparison. Horizontal comparative experiments were conducted on TSP instances of different scales, and the results showed that as the problem size increased, the advantages of AACO-LST became more apparent. In Section 4.3, to further analyze the solution quality of selected algorithms, 100 repeated experiments were conducted on 4 TSP instances using AACO-LST and ACS. The results of this stage of experiments show that the median length of the AACO-LST path is smaller than ACS, and as the problem size increases, the difference in median length between them increases, indicating that the performance of AACO-LST is less affected by the problem size. In Section 4.4, AACO-LST was compared with PSO and ACS to analyze the convergence speed of the proposed algorithm. The results show that the convergence speed of AACO-LST has reached more than twice that of PSO and ACS. Multiple experimental results have shown that the improved state transition rules guide ants to choose the next city to arrive at, enhancing their adaptive search ability. At the same time, the simplified 2-opt operator also improves the accuracy of the algorithm. Finally, the constructed adaptive pheromone update rules also enhance the population diversity of the AACO-LST, and make the release and volatilization of pheromones more reasonable through adaptive calculation, enhancing the global search ability of AACO-LST at the same time. In summary, AACO-LST has significant advantages in solving quality, stability, and convergence speed compared to those comparative algorithms, and can be used to effectively solve large-scale TSPs

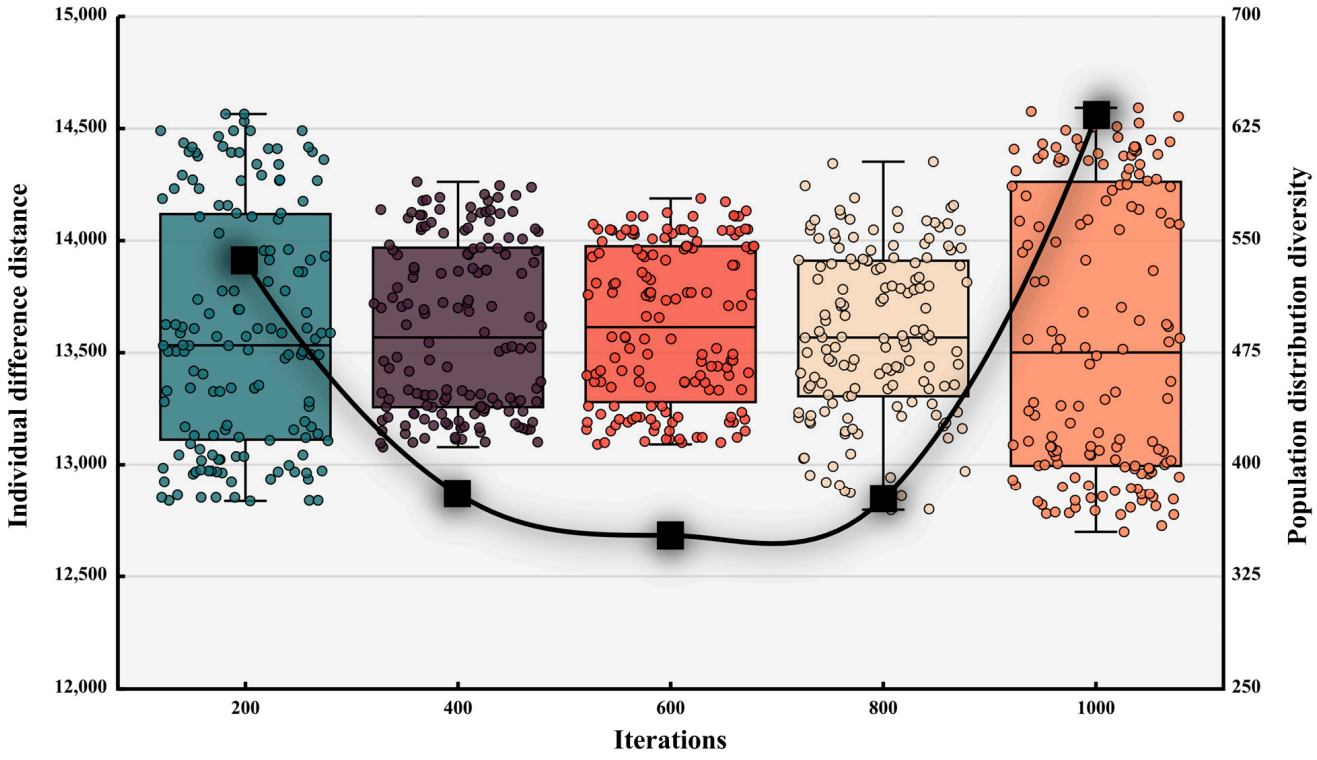


Figure 8. Analysis results of population diversity of AACO-LST.

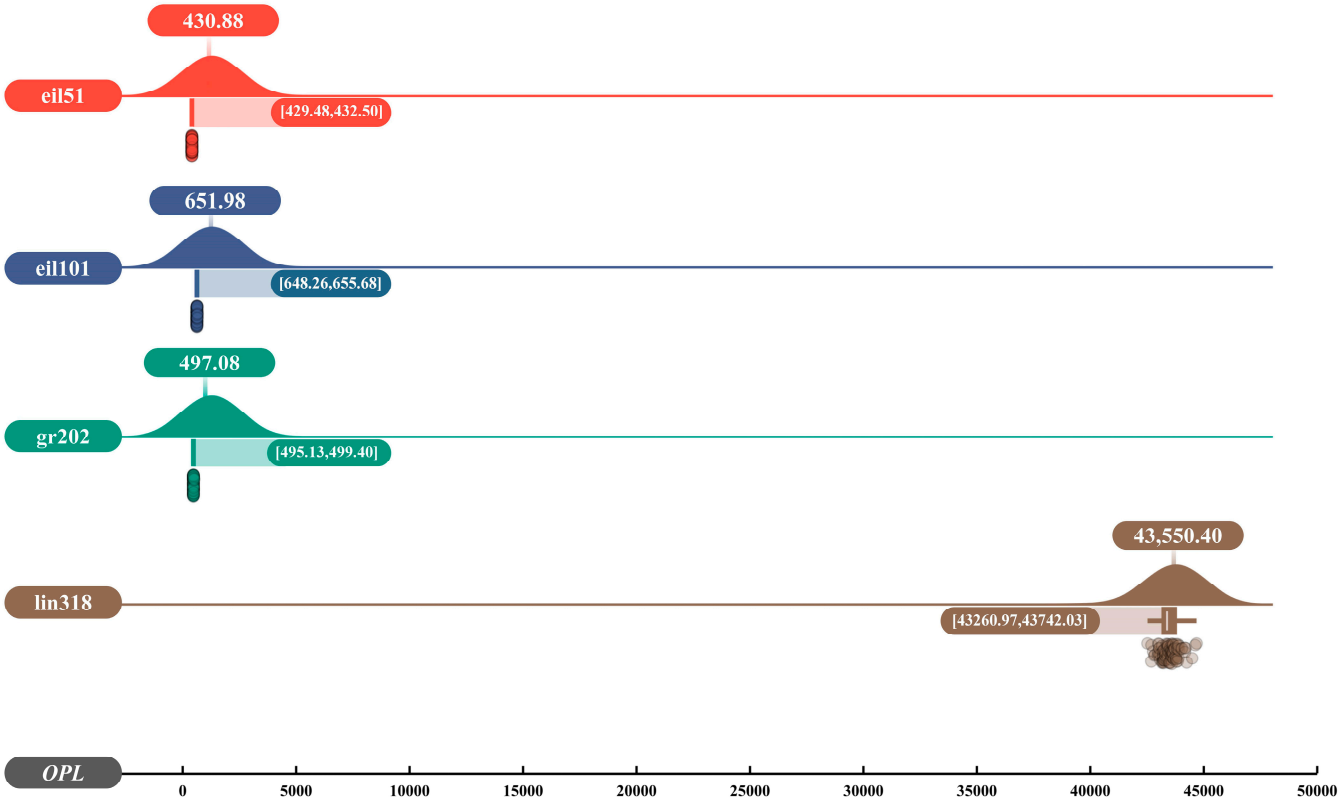


Figure 9. Convergence analysis of the AACO-LST.

5. Conclusions

In this work, we provide a comparison of the solution quality, average convergence speed, and related stability between the AACO-LST algorithm and other algorithms. The AACO-LST algorithm achieved an average *Dev* of 1.44% for solving all 45 instances selected from TSPLib, while ACS exhibited an average *Dev* of 7.02%. This indicates that AACO-LST has significantly enhanced the solution quality by 79% compared to ACS. Among them, there are 21 groups in AACO-LST with a *Dev* lower than 0.5%, accounting for a high proportion of 47%; whereas ACS only has 2 groups with a *Dev* lower than 0.5%, representing merely 9%. If the criterion is to achieve the known optimal solution, then AACO-LST has an accuracy that is over five times higher than ACS. In the statistical comparison graph of solving quality with ACS, it was observed that as the problem size increases, the gap between the median path length of AACO-LST and ACS also widens, while the interquartile range of AACO-LST gradually narrows. In a longitudinal comparison with ASFPA-Ls, AACO-LST has more than 50% less *Err* per TSP instance than ASFPA-Ls. In the comparison with six optimization algorithms of each type, (MSGPSO, MDIGA, DTSA, IMGRA, GA-SAC, and SALNS), there is no significant advantage of AACO-LST for small-scale TSPs such as eil51, st70, and rat99. For the large-scale TSP tsp225, AACO-LST reduces *Err* by 60% over SALNS, which is the best performer among the algorithms, with a PE difference of 0.03%, and both the optimal and average solutions exceed the SALNS. For larger instances like lin318, AACO-LST outperforms the comparison algorithms in terms of both optimal and average solutions, reducing *Err* by 20% and PE by 37%, surpassing even the best-performing GA-SAC algorithm. In convergence speed tests conducted on eil51, eil101, gr202, and lin318 instances, it was observed that the average convergence speed of AACO-LST was more than 50% higher than that of ACS and 142% higher than that of PSO. In the process of population diversity and convergence analysis, AACO-LST has also been shown to be better able to maintain the diversity level of the population to avoid falling into a local optimum. Meanwhile, the results obtained from multiple repetitive experiments also confirm the high degree of convergence that AACO-LST has, which indicates that

the proposed algorithm is more stable. In the future, the practical application areas of AACO-LST can be expanded, such as wireless communication routing, DNA sequencing, etc., to fulfill more practical needs. Of course, AACO-LST is not limited to solving the TSP; it can also be extended to other optimization problems in continuous spaces, which is the direction of our next research.

Author Contributions: Conceptualization, X.-F.W. and K.T.; Methodology, X.-F.W. and Y.-H.J.; Software, X.-F.W.; Writing—original draft preparation, X.-F.W., Y.-H.J. and Z.-W.C.; Supervision, L.Y.; Project administration, K.T.; Writing—review & editing, K.T.; Visualization, Z.-W.C. All authors have read and agreed to the published version of the manuscript.

Funding: This work was partly supported by the National Innovation Training Program for College Students (NO. 202210408015), and the Science and Technology Research Project of Jiangxi Provincial Department of Education (NO. GJJ211331).

Data Availability Statement: The data that support the findings of this study are available from the corresponding author, K.T., upon reasonable request.

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

References

1. Tostado-Véliz, M.; Matos, M.A.; Lopes, J.A.P.; Jurado, F. An improved version of the continuous Newton's method for efficiently solving the power-flow in ill-conditioned systems. *Int. J. Electr. Power* **2021**, *124*, 106389. [CrossRef]
2. Yuan, W.; Hu, F.; Lu, L. A new non-adaptive optimization method: Stochastic gradient descent with momentum and difference. *Appl. Intell.* **2022**, *52*, 3939–3953. [CrossRef]
3. Blum, C.; Roli, A. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.* **2003**, *35*, 268–308. [CrossRef]
4. Papadimitriou, C.H. On the complexity of integer programming. *J. ACM* **1981**, *28*, 765–768. [CrossRef]
5. Rao, M.; Zionts, S. Allocation of transportation units to alternative trips—A column generation scheme with out-of-kilter subproblems. *Oper. Res.* **1968**, *16*, 52–63. [CrossRef]
6. Lawler, E.L.; Wood, D.E. Branch-and-bound methods: A survey. *Oper. Res.* **1966**, *14*, 699–719. [CrossRef]
7. Lera-Romero, G.; Miranda Bront, J.J.; Soulignac, F.J. Dynamic programming for the time-dependent traveling salesman problem with time windows. *Informs. J. Comput.* **2022**, *34*, 3292–3308. [CrossRef]
8. Fisher, M.L. Optimal solution of vehicle routing problems using minimum k-trees. *Oper. Res.* **1994**, *42*, 626–642. [CrossRef]
9. Yin, Y.-H.; Shen, L.-C.; Jiang, Y.-H.; Gao, S.; Song, J.; Yu, D.-J. Improving the prediction of DNA-protein binding by integrating multi-scale dense convolutional network with fault-tolerant coding. *Anal. Biochem.* **2022**, *656*, 114878. [CrossRef]
10. Jiang, Y.-H.; Gao, S.; Yin, Y.-H.; Xu, Z.-F.; Wang, S.-Y. A control system of rail-guided vehicle assisted by transdifferentiation strategy of lower organisms. *Eng. Appl. Artif. Intel.* **2023**, *123*, 106353. [CrossRef]
11. Katoch, S.; Chauhan, S.S.; Kumar, V. A review on genetic algorithm: Past, present, and future. *Multimed. Tools. Appl.* **2021**, *80*, 8091–8126. [CrossRef] [PubMed]
12. George, T.; Amudha, T. Genetic algorithm based multi-objective optimization framework to solve traveling salesman problem. In *Proceedings of the Advances in Computing and Intelligent Systems: Proceedings of ICACM 2019*; Springer: Singapore, 2020; pp. 141–151. [CrossRef]
13. Gao, P.; Zhou, L.; Zhao, X.; Shao, B. Research on ship collision avoidance path planning based on modified potential field ant colony algorithm. *Ocean. Coast. Manag.* **2023**, *235*, 106482. [CrossRef]
14. Karimi, F.; Dowlatshahi, M.B.; Hashemi, A. SemiACO: A semi-supervised feature selection based on ant colony optimization. *Expert. Syst. Appl.* **2023**, *214*, 119130. [CrossRef]
15. Liu, D.; Hu, X.; Jiang, Q. Design and optimization of logistics distribution route based on improved ant colony algorithm. *Optik* **2023**, *273*, 170405. [CrossRef]
16. Ren, T.; Luo, T.; Jia, B.; Yang, B.; Wang, L.; Xing, L. Improved ant colony optimization for the vehicle routing problem with split pickup and split delivery. *Swarm. Evol. Comput.* **2023**, *77*, 101228. [CrossRef]
17. 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]
18. Minh, H.-L.; Khatir, S.; Rao, R.V.; Abdel Wahab, M.; Cuong-Le, T. A variable velocity strategy particle swarm optimization algorithm (VVS-PSO) for damage assessment in structures. *Eng. Comput.* **2023**, *39*, 1055–1084. [CrossRef]
19. Sedighizadeh, D.; Masehian, E.; Sedighizadeh, M.; Akbaripour, H. GEPSO: A new generalized particle swarm optimization algorithm. *Math. Comput. Simulat.* **2021**, *179*, 194–212. [CrossRef]
20. Wang, X.; Dong, F.; Liu, J.; Tan, Y.; Hu, S.; Zhao, H. The self-healing of Bacillus subtilis biofilms. *Arch. Microbiol.* **2021**, *203*, 5635–5645. [CrossRef]

21. Wu, X.; Han, J.; Wang, D.; Gao, P.; Cui, Q.; Chen, L.; Liang, Y.; Huang, H.; Lee, H.P.; Miao, C.; et al. Incorporating Surprisingly Popular Algorithm and Euclidean distance-based adaptive topology into PSO. *Swarm. Evol. Comput.* **2023**, *76*, 101222. [CrossRef]
22. Chai, Z.; Li, W.; Li, Y. Symmetric uncertainty based decomposition multi-objective immune algorithm for feature selection. *Swarm. Evol. Comput.* **2023**, *78*, 101286. [CrossRef]
23. Guo, F.; Han, W.; Su, X.-C.; Liu, Y.-J.; Cui, R.-W. A bi-population immune algorithm for weapon transportation support scheduling problem with pickup and delivery on aircraft carrier deck. *Def. Technol.* **2021**, *22*, 119–134. [CrossRef]
24. Lian, L. Reactive power optimization based on adaptive multi-objective optimization artificial immune algorithm. *Ain. Shams. Eng. J.* **2022**, *13*, 101677. [CrossRef]
25. Wei, W.; Chen, S.; Lin, Q.; Ji, J.; Chen, J. A multi-objective immune algorithm for intrusion feature selection. *Appl. Soft Comput.* **2020**, *95*, 106522. [CrossRef]
26. Wang, Y.; Han, Z. Ant colony optimization for traveling salesman problem based on parameters optimization. *Appl. Soft Comput.* **2021**, *107*, 107439. [CrossRef]
27. Yang, K.; You, X.; Liu, S.; Pan, H. A novel ant colony optimization based on game for traveling salesman problem. *Appl. Intell.* **2020**, *50*, 4529–4542. [CrossRef]
28. Dahan, F.; El Hindi, K.; Mathkour, H.; AlSalman, H. Dynamic flying ant colony optimization (DFACO) for solving the traveling salesman problem. *Sensors* **2019**, *19*, 1837. [CrossRef]
29. Zhang, Z.; Xu, Z.; Luan, S.; Li, X.; Sun, Y. Opposition-based ant colony optimization algorithm for the traveling salesman problem. *Mathematics* **2020**, *8*, 1650. [CrossRef]
30. Shahadat, A.S.B.; Akhand, M.; Kamal, M.A.S. Visibility adaptation in ant colony optimization for solving traveling salesman problem. *Mathematics* **2022**, *10*, 2448. [CrossRef]
31. Yu, X.; Yu, L.; Zheng, M.; Lu, J.; Zhang, L. Firefly algorithm and ant colony algorithm to optimize the traveling salesman problem. *J. Phys. Conf. Ser.* **2022**, *2253*, 012010. [CrossRef]
32. 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]
33. Pérez-Carabaza, S.; Gálvez, A.; Iglesias, A. Rank-Based Ant System with Originality Reinforcement and Pheromone Smoothing. *Appl. Sci.* **2022**, *12*, 11219. [CrossRef]
34. Miller, C.E.; Tucker, A.W.; Zemlin, R.A. Integer programming formulation of traveling salesman problems. *J. ACM.* **1960**, *7*, 326–329. [CrossRef]
35. Nikolaev, A.; Batsyn, M. Branch-and-bound algorithm for symmetric travelling salesman problem. In *Combinatorial Algorithms: 29th International Workshop*; Springer: New York, NY, USA, 2018; pp. 311–322. [CrossRef]
36. Held, M.; Karp, R.M. The traveling-salesman problem and minimum spanning trees: Part II. *Math Program.* **1971**, *1*, 6–25. [CrossRef]
37. Roy, A.; Manna, A.; Maity, S. A novel memetic genetic algorithm for solving traveling salesman problem based on multi-parent crossover technique. *Decis. Making Appl. Manag. Eng.* **2019**, *2*, 100–111. [CrossRef]
38. Emambocus, B.A.S.; Jasser, M.B.; Hamzah, M. An enhanced swap sequence-based particle swarm optimization algorithm to solve TSP. *IEEE Access.* **2021**, *9*, 164820–164836. [CrossRef]
39. İlhan, İ.; Gökmen, G. A list-based simulated annealing algorithm with crossover operator for the traveling salesman problem. *Neural Comput. Appl.* **2022**, *34*, 7627–7652. [CrossRef]
40. Ma, Q.; Ge, S.; He, D.; Thaker, D.; Drori, I. Combinatorial optimization by graph pointer networks and hierarchical reinforcement learning. *arXiv* **2019**, arXiv:1911.04936.
41. 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]
42. Dorigo, M.; Maniezzo, V.; Colorni, A. Ant system: Optimization by a colony of cooperating agents. *IEEE Trans. Syst. Man Cybern. Part B* **1996**, *26*, 29–41. [CrossRef]
43. Kumar, S.; Parhi, D.R.; Muni, M.K. Optimal path search and control of mobile robot us-ing hybridized sine-cosine algorithm and ant colony optimization technique. *Ind. Robot* **2020**, *47*, 535–545. [CrossRef]
44. 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]
45. Du, P.-Z.; Tang, Z.-M.; Sun, Y. An object-oriented multi-role ant colony optimization algorithm for solving TSP problem. *Control. Decis.* **2014**, *29*, 1729–1736. [CrossRef]
46. Rokbani, N.; Kumar, R.; Abraham, A.; Alimi, A.M.; Long, H.V.; Priyadarshini, I.; Son, L.H. Bi-heuristic ant colony optimization-based approaches for traveling salesman problem. *Soft Comput.* **2021**, *25*, 3775–3794. [CrossRef]
47. Cinar, A.C.; Korkmaz, S.; Kiran, M.S. A discrete tree-seed algorithm for solving symmetric traveling salesman problem. *Eng. Sci. Technol.* **2020**, *23*, 879–890. [CrossRef]
48. Cui, Y.; Zhong, J.; Yang, F.; Li, S.; Li, P. Multi-subdomain grouping-based particle swarm optimization for the traveling salesman problem. *IEEE Access.* **2020**, *8*, 227497–227510. [CrossRef]
49. Xin, J.; Zhong, J.; Yang, F.; Cui, Y.; Sheng, J. An improved genetic algorithm for path-planning of unmanned surface vehicle. *Sensors* **2019**, *19*, 2640. [CrossRef] [PubMed]

50. Rao, W.; Jin, C.; Lu, L. An Improved Greedy Algorithm with Information of Edges' Location for Solving the Euclidean Traveling Salesman Problem. *Chin. J. Comput.* **2013**, *36*, 836–850. [CrossRef]
51. Chen, B.; Liu, W. SAC Model Based Improved Genetic Algorithm for Solving TSP. *J. Front. Comput. Sci. Technol.* **2021**, *15*, 1680–1693. [CrossRef]
52. Sun, J.; Liu, S.; Wu, X. Simulated Annealing Algorithm Based on Large Neighborhood Search To Solve TSP. *Comput. Simul.* **2023**, *40*, 415–420. [CrossRef]