Hindawi
Journal of Advanced Transportation
Volume 2021, Article ID 6642009, 16 pages
https://doi.org/10.1155/2021/6642009

Research Article

An Improved Ant Colony Optimization Based on an Adaptive Heuristic Factor for the Traveling Salesman Problem

Pengzhen Du,[1] **Ning Liu**,[2] **Haofeng Zhang**,[1] **and Jianfeng Lu**[1]

[1]*School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, China*
[2]*Nanjing Customs, Nanjing, China*

Correspondence should be addressed to Pengzhen Du; dupengzhen@njust.edu.cn

Received 23 December 2020; Revised 2 September 2021; Accepted 30 September 2021; Published 12 October 2021

Academic Editor: Jose E. Naranjo

The traveling salesman problem (TSP) is a typical combinatorial optimization problem, which is often applied to sensor placement, path planning, etc. In this paper, an improved ACO algorithm based on an adaptive heuristic factor (AHACO) is proposed to deal with the TSP. In the AHACO, three main improvements are proposed to improve the performance of the algorithm. First, the k-means algorithm is introduced to classify cities. The AHACO provides different movement strategies for different city classes, which improves the diversity of the population and improves the search ability of the algorithm. A modified 2-opt local optimizer is proposed to further tune the solution. Finally, a mechanism to jump out of the local optimum is introduced to avoid the stagnation of the algorithm. The proposed algorithm is tested in numerical experiments using 39 TSP instances, and results shows that the solution quality of the AHACO is 83.33% higher than that of the comparison algorithms on average. For large-scale TSP instances, the algorithm is also far better than the comparison algorithms.

1. Introduction

1.1. General Perspective. The TSP is an optimization problem in which a traveler needs to pass through all cities once and only once in order to obtain the shortest path. The TSP is a typical combinatorial optimization problem that is highly relevant and is widely used in a variety of practical problems, such as computer networks, sensor placement, vehicle routing, logistics and distribution, circuit board wiring, and intelligent transportation [1]. The TSP is an NP-hard problem, and the traditional optimization algorithm is not effective, so a metaheuristic algorithm is often used to optimize the TSP, and the TSP is often used as the benchmark of the metaheuristic algorithm.

A metaheuristic algorithm is an implementation on a specific problem guided by a set of guidelines or strategies [2]. Metaheuristic algorithms do not require the problem itself to have precise mathematical characteristics and have good optimization performance and strong robustness. Compared with a traditional algorithm, metaheuristic algorithms do not guarantee an optimal solution, but they can obtain a satisfactory solution within a certain amount of time or computation [3]; at the same time, they are very adaptable and are used in almost all scientific fields and engineering applications.

1.2. Literature Review and Motivation. For the reasons mentioned above, in recent years, many metaheuristic algorithms have been used to solve the TSP, such as the genetic algorithm (GA) [4–7], ant colony optimization (ACO) algorithm [8–11], particle swarm optimization (PSO) [12–15], artificial bee colony (ABC) algorithm [16, 17], and spider monkey optimization (SMO) [18]. Most metaheuristic algorithms essentially obtain a feasible solution first, improve the feasible solution by using mechanisms such as movement, exchange, mutation, and cooperative perception, repeat the improvement process many times to gradually approach the optimal solution, and finally finish the optimization through certain conditions. Metaheuristic algorithms can generally obtain better results when solving small-scale optimization problems, but when addressing

some large-scale optimization problems, such as large-scale TSP problems, there are often two concerns: (1) the convergence speed is too slow, resulting in a long computation time; (2) it is easy to fall into a local optimum and stagnate. These two problems are interrelated: the former is due to the low speed of approximation, which is usually improved by improving some parameters or introducing some mechanisms, while the latter is mainly due to the lack of population diversity, which is directly linked to the final solution quality and can usually be improved by enhancing population diversity. Much research has been conducted on the shortcomings of metaheuristic algorithms, and improvements to the standard algorithm itself or mixing the standard algorithm with a variety of mechanisms have achieved good results.

In [4], the authors proposed a hybrid method of GA and PSO, while using the cross-mutation of GA and the position update of PSO, which greatly improved the convergence speed of the algorithm. Wang et al. [6] proposed multi-offspring genetic algorithm (MO-GA), which improved the probability of producing excellent individuals and made the population competitive. Compared with the basic genetic algorithm, MO-GA has a greater improvement. In [7], the authors proposed a hybrid genetic algorithm with variable neighborhood search. The core idea in this study is the dual-chromosome solutions, and the deleting and reinserting operator on them. In [15], the speed calculation method of the particles was improved. The calculation method in the standard PSO was not adopted immediately. Instead, a number of tentative tours were evaluated, and the most adaptable one was used. This significantly improved the accuracy of the algorithm, but the calculation time increased. In [16], the authors applied ABC to the TSP problem through the defined swap sequence and swap operator. On this basis, they used the 3-opt algorithm combined with the mechanism of jumping out of the local optimum to avoid the stagnation of the algorithm. In [17], an improved ABC algorithm based a novel neighborhood selection mechanism was proposed to solve the TSP problem, which enhanced the solution quality. In [18], the authors discretized the SMO and made it suitable for TSP problems. Under the inherent grouping and regrouping strategy of SMO, they proposed a method of exchanging experiences between random individuals, global leaders, and local leaders, which improved the search ability of the algorithm. Ali et al. [19] improved the differential evolution (DE) algorithm by using k-means algorithm to classify nodes in TSP. It enhanced the quality of the initial solution, and they also used a mutation strategy with better search performance to substantially improve the solution quality.

ACO is a metaheuristic algorithm proposed by the Italian scholar Dorigo in the 1990s, inspired by the foraging behavior of an ant colony [20]. In ACO, a solution of the solution space is represented as an individual ant, which will leave pheromones on the path it passes when searching for food, and other individuals will prefer the path with more pheromones. At the same time, pheromones evaporate over time, and the longer the path from the nest to the food is, the less pheromones the ant will retain on the path, due to the evaporation mechanism. The ACO algorithm actually leverages the simple behavior of an individual ant to form a complete optimization system through intergroup collaboration. Unlike many continuous domain optimization algorithms, ACO has the feature of building paths step by step, which is comparatively more applied to discrete and combinatorial optimization problems, and many intensive studies have been conducted on ACO for solving TSP problems.

Sudipta et al. [8] introduced a transfer strategy based on a large neighborhood search in the standard ACO, which maintains population diversity and can more quickly reach the global optimal region, improves the search efficiency of the basic ACO, and reduces the search time for the optimal solution. DEACO [10] classifies cities based on the standard ACO, and the classification results are used to select the starting city. At the same time, it replaces the pheromone evaporation method in the standard ACO with a dynamic pheromone evaporation method that changes with the number of iterations. The proposed method enhances the evolution speed and avoids local optimum. Mahi et al. [14] proposed a hybrid algorithm using PSO to optimize the optimal parameters of the ACO and 3-opt algorithm to jump out of the local optimum, thus boosting the solution accuracy and robustness of the algorithm. In [21], the authors proposed an improved ACO algorithm, which introduces a strategy of dynamically adjusting pheromone evaporation and preserves a small amount of pheromones in the iterative initial path. It improves the search capability of the solution space, progressively increases the pheromones, and enhances the convergence speed of the algorithm to achieve a better balance between solution quality and convergence speed. Escario et al. [22] extended the ACO algorithm, which divides different class of colonies, which perform different actions, and proposed a transformation strategy, which can be dynamically transformed between individuals in different colonies, enriching the diversity of the population and improving the quality of the solution. Delévacq et al. [23] improved the roulette in the ACO by parallelizing optimization and implemented it on a CUDA-based GPU platform, which significantly reduced the running time of the algorithm. In [24], an improved ACO introduced a pheromone updating strategy for the best path, and metaheuristic information and pheromone density conversion rules were applied to find the best path. Chaos is the seemingly random irregular movement that occurs in a deterministic system, and many algorithms use it to provide better search capabilities than random. Xu et al. [25] introduced a dynamic moving method based on unidimensional chaotic mapping to improve the solution accuracy and overall efficiency of the algorithm. Lei and Wang [26] introduced an elite strategy and a max-min ant system in order to increase the search capability of the solution space and enhance the convergence speed, which ensures high quality solutions while preventing the algorithm from falling into stagnation.

Most current research has focused on certain improvements, e.g., improving the quality of the initial solution by some mechanism to cover the entire solution space at the initial stage, controlling pheromones such that they jump

out of the local optimum to avoid degradation of the solution quality due to stagnation, introducing elite strategies to improve convergence speed, using a local search operator for tuning, providing multiple population strategies to maximize the diversity of populations, dynamically adjusting algorithm parameters with the number of iterations or other criteria to achieve a balance between solution speed and solution quality, and performing targeted parallel optimization based on parallel processors to improve the speed of the algorithm. However, the aforementioned algorithm improvements are mostly improvements to the algorithm itself, which fails to take full advantage of the spatial information of the TSP and performs poorly in some TSP instances. In view of this, this paper proposes an improved ACO algorithm (AHACO) that makes full use of the spatial information of the TSP with an adaptive heuristic factor. The specific description of AHACO is given in Section 3.

1.3. Outline of This Paper. The remainder of the paper is organized as follows: Section 2 gives the mathematical model of the TSP problem and describes the standard ant colony algorithm for solving the TSP problem; Section 3 describes the proposed AHACO algorithm for solving the TSP problem; Section 4 performs experiments on the standard TSPLIB [27] dataset and discusses the results; and Section 5 provides the conclusion of the work in this paper.

2. Principle and Methodology of ACO to Solve the TSP

2.1. The Traveling Salesmen Problem. Any solution of the TSP problem can be regarded as a Hamiltonian path $H = \{V, E\}$. Among them, $V = \{v_1, v_2, v_3, \ldots, v_n\}$ denotes the city set in the TSP problem. $E = \{(v_i, v_j)\colon v_i \in V, v_j \in V, i \neq j\}$ indicates a response to a city edge set, so the optimization problem of TSP can be described as follows.

$$\text{Minimize} \quad C(H \in \Pi), \tag{1}$$

where $C(H) = \sum_{i=1}^{n-1} \|v_i - v_{i+1}\| + \|v_n - v_1\|\colon v_i \in V, v_j \in V, i \neq j$, $\|\cdot\|$. indicates Euclidean distance, and Π indicates the solution space of the TSP.

2.2. Principle and Methodology of ACO to Solve the TSP. The solution of the TSP problem by ACO is a process in which a number of ants search for paths in parallel. When an ant completes one search of the path, the path on which it travels is a feasible solution to the TSP problem. Using ACS [20] as an example, the algorithm starts by putting m ants randomly into n cities. So, the k-th ant at moment t uses a roulette strategy with probability p_{ij}^k to determine the next city.

$$p_{ij}^k = \begin{cases} \dfrac{\tau_{ij}^{\alpha}(t)\eta_{ij}^{\beta}}{\sum_{u \in \text{allowed}_k} \tau_{iu}^{\alpha}(t)\eta_{iu}^{\beta}}, & j \in \text{allowed}_k, \\ 0, & \text{otherwise}, \end{cases} \tag{2}$$

$$\eta_{iu} = \frac{1}{d_{iu}}, \tag{3}$$

where $\tau_{iu}(t)$ denotes the amount of pheromones between city i and city u at time t, α denotes the relative importance of the pheromones, η_{iu} stands for the heuristic factor from city i to city u, d_{iu} is the distance between city i and city u, β indicates the importance of the heuristic factor, and allowed_k represents the set of cities not traversed by the k-th ant, i.e., the set of allowed cities.

The ant completes the solution path search under the influence of pheromones and heuristic factors. During the search process, the ants release pheromones to enhance the positive feedback of the whole system, which helps the algorithm to converge faster. The pheromone updating strategy is divided into three models: an ant quality model, ant density model, and ant cycle model. The ant cycle model is more biased towards global information and is more widely used [8, 10, 23, 24]. The formula for updating pheromones using the ant cycle model is as follows:

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \rho\Delta\tau_{ij}(t), \tag{4}$$

$$\Delta\tau_{ij}(t) = \sum \Delta\tau_{ij}^k(t), \tag{5}$$

$$\Delta\tau_{ij}^k(t) = \begin{cases} \dfrac{Q}{L_k}, & (i, j) \text{ tour done by ant } k, \\ 0, & \text{otherwise}, \end{cases} \tag{6}$$

where ρ is the evaporation coefficient. If ρ increases, it will enhance the randomness of the algorithm, and if it decreases, it will speed up the algorithm's convergence, but the algorithm is more likely to fall into a local optimum. Q is the total amount of pheromones. (i, j) represents the path from city i to city j. L_k is the length of the path traveled by the k-th ant in the current iteration.

Guided by pheromones τ and distance η, the ant colony progressively improves the quality of the solution, and the algorithm is completed when a specified number of iterations is reached or the current optimal solution satisfies the expectation.

3. Problem Mapping of AHACO for TSP

Based on the standard ACO, AHACO has the following three main improvements. (1) It fully considers the information contained in the solution space, uses k-means to classify the TSP cities, and sets a special colony from the original colony. It applies a reward-punish factor. When the special colony selects the next city, it will prefer interclass or

intraclass cities according to the reward-punish factor, and the reward-punish factor changes dynamically by iteration. (2) A simplified 2-opt optimizer is implemented. (3) The scout bee mechanism in ABC is introduced, which can make the algorithm jump out of the local optimum. Improvement 1 fully enhances the diversity of the population and is mainly used to enhance the search ability of the algorithm. The ant colony algorithm is a method with a feedback mechanism (pheromone). If there is an error in the order of two adjacent cities on an obtained better path, then this path will be strengthened by pheromone accumulation. It is difficult for the ant colony algorithm to optimize this situation through its own mechanism. Thus, improvement 2 is introduced to tune the optimal solution. Finally, the ant colony algorithm has a widely criticized shortcoming: it is easy to fall into a local optimum. Therefore, improvement 3 is introduced, which is used to jump out of the local optimum. Figure 1 shows a simplified flowchart of the AHACO, in which the thick border indicates the improved part. The specific algorithm flow is given by Algorithm 1.

3.1. Classification of TSP Cities. Existing algorithms treat TSP cities indiscriminately, ignoring the spatial information contained in the TSP itself. This algorithm uses k-means to classify the cities and introduces confidence intervals to separate the classless cities from the already classified classes. Prerequisites are provided for the execution of different search strategies for multi-role ant colonies.

3.1.1. K-Means Clustering. K-means clustering is a very classical clustering algorithm, which is simple, efficient, easy to understand, and widely used in reality. For the TSP problem, its basic procedure is as follows:

Step 1: randomly select k cities as class centers from the set of cities cities $= \{c_1, c_2, \ldots, c_n\}$ to construct the set center $= \{s_1, s_2, \ldots, s_k\}$.

Step 2: $\forall c_i \in$ cities classified in class j as

$$j = \arg\min\left(d_{iu}\right), \quad u = 1, 2, \ldots, k, \tag{7}$$

$$d_{iu} = \left\|c_i - s_u\right\|, \quad s_u \in \text{center}, \tag{8}$$

where $\|\cdot\|$. indicates Euclidean distance.

Step 3: update the class center according to equation (9), $\text{class}_j(t)$ is the set of cities of the class j at moment t, and $n_j(t)$ is the number of cities in $\text{class}_j(t)$.

$$s_j(t+1) = \frac{\sum_{c_i \in \text{class}_j(t)} c_i}{n_j(t)}, \quad s_j \in \text{center}. \tag{9}$$

Step 4: if center is updated, go to Step 2 and Step 3; otherwise, end the algorithm.

If the number of city classes is too large, the algorithm will converge quickly and be trapped in a local optimum; if it is too small, the algorithm will take too long to search and the number of optimal iterations will increase. In the following, some discussions about the number of city classes are carried out. The number of city classes should be determined according to the following formula:

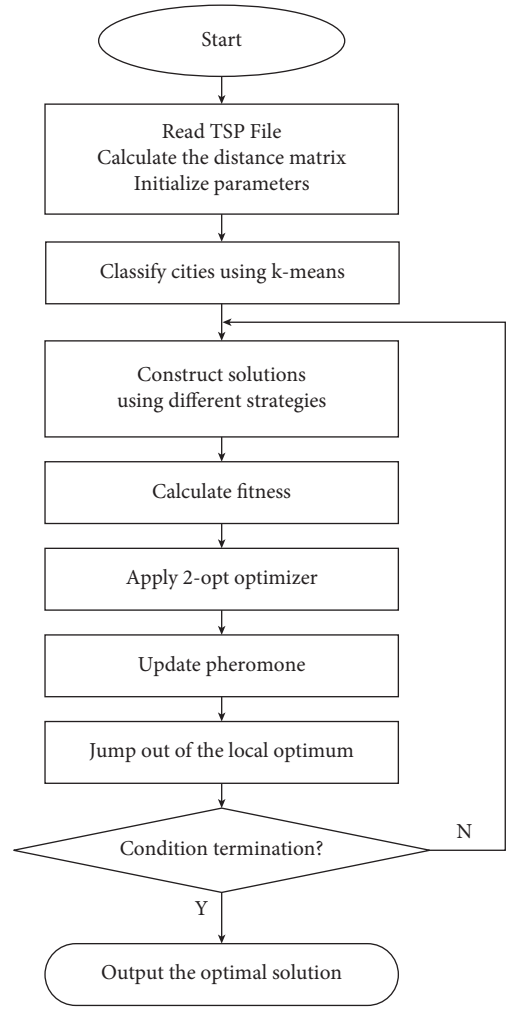

Figure 1: The flowchart of AHACO.

$$k = \begin{cases} \text{floor}\left(\dfrac{n}{25}\right), & n \geq 125, \\ 4, & \text{otherwise}, \end{cases} \tag{10}$$

where k is the number of city classes to be determined, n is the number of TSP cities, and floorrepresents the largest integer less than or equal to the parameter.

The population diversity in the ACO mainly depends on the way the city is selected. First, when the value of k is 1 or the number of cities, AHACO degenerates into ACO because all cities are intraclass cities or interclass cities, and the reward-punish factor will become invalid (related descriptions are given in Section 3.2). When k is the square root of the number of cities, the probability of each class being selected is equal. At the same time, once the ant chooses an interclass city, the class of the interclass city becomes the class of the ant, and the interclass city within a class can be regarded as a whole. When the value of k is the square root of

Input: t_{max}, popsize, tries, ρ, β, ϵ, ξ_{max}, η, τ_{init}, Q
Output: optimal solution
(1) Calculate Euclidean distance for the cities and get the number of cities as citynum
(2) Initialize pheromone on all path with τ_{init}
(3) Apply k-means clustering for the cities (equations (7)–(10))
(4) Separate non-classified cities (equation (11))
(5) times = 0
(6) $\Delta\xi = 2(\xi_{max} - 1)/t_{max}$
(7) **for** iter = 1: t_{max} **do**
(8) **if** iter $< t_{max}/2$ **then**
(9) $\xi = \xi - \Delta\xi$
(10) $\gamma = -1$
(11) else
(12) $\xi = \xi + \Delta\xi$
(13) $\gamma = 1$
(14) end
(15) **for** i = 1: popsize **do**
(16) **for** j = 1: citynum **do**
(17) **if** i%2 == 1 **then**
(18) ant i select next city using γ, ξ (equations (12)–(13)); //special ant
(19) else
(20) ant i select next city by equation (2); //normal ant
(21) **end**
(22) Set tabu table for ant I
(23) end
(24) Calculate fitness of the corresponding solution obtained by ant i (equation (1))
(25) end
(26) Select the best solutions for all ants (include normal ants and special ants)
(27) Update global best solution (optimal solution)
(28) Apply improved 2-opt algorithm to optimal solution (Section 3.3)
(29) Update pheromone on normal and special best solution separately (equations (14)–(16))
(30) **if** global best solution not be updated **then**
(31) times = times + 1
(32) **if** times ≥ tries **then**
(33) Re-initialize the pheromone on the global best solution with τ_{init}
(34) times = 0
(35) end
(36) else
(37) times = 0
(38) end
(39) end

Algorithm 1: AHACO.

the number of cities, the number of cities in the class is equal to the number of classes. In this case, the diversity of the population is expected to be the highest. However, the distribution of cities is not always uniform. K-means is very sensitive to the choice of centroid, and its anti-noise ability is relatively poor. Therefore, equation (10) is only an approximate formula defined from experience.

3.1.2. Classless Cities. In the process of selecting the next city, the algorithm encourages or penalizes the ant colony based on the city class relationship. By introducing confidence intervals and separating the classless cities, the barriers between city classes can be buffered, further increasing the diversity of the algorithm.

For the TSP problem of n cities, the distance of each city to the center of its respective class constructs the set. The cities satisfying equation (11) are then separated into classless cities.

$$d_i - \mu \geq \varepsilon\sigma \quad i = 1, 2, \ldots, n, \tag{11}$$

where μ is the distance mean, σ is the distance standard deviation, and $\varepsilon \in [1, 2]$ is the separation factor; taking its values too small will increase the number of unclassified cities, and making its values too large will decrease the number of unclassified cities. The value taken in this experiment is 1.5. Figure 2 shows a classification map of TSPLIB [27] examples eil51, kroA100, kroB150, and gil262, where "+" denotes classless cities, circles denote cities with classes, and cities with classes are distinguished by different colors.

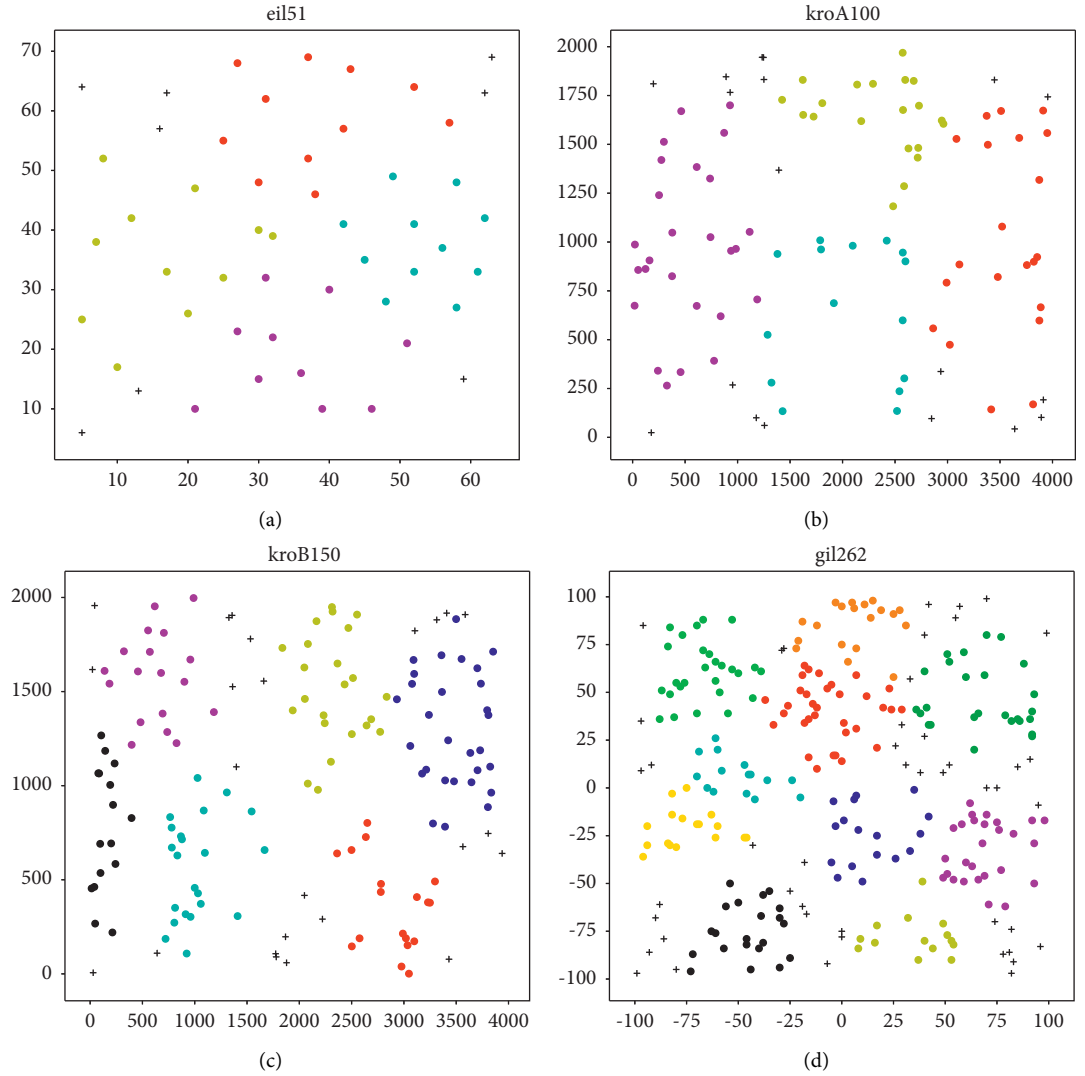

Figure 2: TSP city classification. (a) eil51. (b) kroA100. (c) kroB150. (d) gil262.

3.2. Multi-Role Colony and Corresponding Strategies. AHACO divides the ant colony into two roles: a normal ant colony and a special ant colony. At the beginning of the algorithm, the two colonies perform a path search simultaneously with different strategies, and the difference is reflected in the different city selection strategies.

The ant selection strategy is based on city class relations, and to represent the differences between city classes, a class operator is first introduced:

$$\operatorname{sgn}(i,j)=\begin{cases}1, & i\neq j,\ i\in \text{class}_m,\ j\in \text{class}_m,\\ -1, & i\in \text{class}_l,\ j\in \text{class}_m,\ l\neq m,\\ 0, & \text{otherwise},\end{cases} \tag{12}$$

where i and j are both city numbers, class_l denotes cities of class l, and class_m denotes cities of class m. When $\operatorname{sgn}(i,j)$ returns 1, city i and city j are in the same class; when $\operatorname{sgn}(i,j)$ returns -1, city i and city j are in different classes; when $\operatorname{sgn}(i,j)$ returns 0, city i and city j are in at least one of the uncategorized cities.

For different class relationships, this paper proposes three search strategies, as follows.

Strategy 1. Take into account intraclass, interclass, class, and no-class searches in order to achieve a balance between global optimality and convergence speed.

Strategy 2. Emphasize an interclass city search with a focus on the global optimum.

Strategy 3. Emphasize the search for similar cities and focus on local path tuning to accelerate convergence.

Equation (13) is used in AHACO to determine the probability of moving to the next city:

$$p_{ij}^{k}=\begin{cases}\dfrac{\tau_{ij}(t)\eta_{ij}^{3}\xi^{\gamma\operatorname{sgn}(i,j)}}{\sum_{u\in \text{allowed}_k}\tau_{iu}(t)\eta_{iu}^{3}\xi^{\gamma\operatorname{sgn}(i,u)}}, & j\in \text{allowed}_k,\\ 0, & \text{otherwise},\end{cases} \tag{13}$$

where $\gamma \in \{-1, 0, 1\}$ determines the choice of strategy.

A normal colony uses Strategy 1, $\gamma = 0$, which treats the various city class relationships indiscriminately. In fact, in this case, equation (13) is equivalent to equation (2).

Special ant colonies at $t \le t_{\max}/2$, $\gamma = -1$, improve the selection probability of interclass cities and shrink the selection probability of intraclass cities.

Special ant colonies at $t > t_{\max}/2$, $\gamma = 1$, improve the probability of selection for intraclass cities and narrow the probability of selection for interclass cities.

$\xi \in [1, \xi_{\max}]$ is the reward-punish factor, which indicates the ant colony's preference for class relationships.

ξ decreases linearly from $\xi_{\max}$ to 1 with $\Delta\xi = 2(\xi_{\max} - 1)/t_{\max}$ when $t \le t_{\max}/2$.

ξ increases linearly from 1 to $\xi_{\max}$ with $\Delta\xi = 2(\xi_{\max} - 1)/t_{\max}$ when $t > t_{\max}/2$.

Different search strategies directly influence the formation of the final path. In Figure 3, all ants in TSPLIB instances eil51, kroA100, kroB150, and gil262 start from the same city under $\xi_{\max} = 8$. Figure 3 shows the path diagram for 10 iterations at $\gamma \in \{-1, 0, 1\}$, with all instances starting from the city numbered 1 that is indicated by a square. The results show that, although there is a random factor, the focus of individual ants is significantly different under different strategies.

The diversity of the population is directly related to the search ability of the algorithm. At present, many algorithms use various mechanisms to improve the diversity of the population, but most of them are improvements of the algorithms themselves, and fail to use the characteristics of the problem. The standard ACO uses a set of parameters to perform spatial search successively, but AHACO divides ants into different roles. One type of ants treats intraclass and interclass cities indiscriminately, and another type of ant prefers intraclass cities or interclass cities based on dynamically changing parameters. The innovation is that different ants perform different searches based on the information of the TSP instance itself, which is the maximum use of TSP spatial information. From the perspective of population diversity, this is a problem-oriented adaptive diversity enhancement mechanism. AHACO divides the population into normal ants and special ants, each of which accounts for half of the population size. In order to facilitate the implementation on the computer, during the operation of the algorithm, odd-numbered ants are classified as special ants, and even-numbered ants are classified as normal ants. The pseudocode of the division process can be seen in lines 17–21 of Algorithm 1.

3.3. Local Optimization. The ant colony algorithm obtains the final solution by a stepwise search, which is prone to form two adjacent points reversed and is difficult to optimize further when approaching the optimal solution, as shown in Figure 4.

The 2-opt [9] is a local optimization operator, and the time complexity is $O(n^2)$. Local tuning in AHACO uses simplified 2-opt with a time complexity of $O(n)$. route $= \{c_0, c_1, \ldots, c_{n-1}\}$, n is the number of cities in the route, and $d(i, j)$ is the distance between c_i and c_j. To facilitate the description of the algorithm flow, in the case of $\forall i \in Z$, when $i \ge n$, c_i equals $c_{i\%n}$. The algorithm starts with $i = 0$.

Step 1: if $d(i, i+1) + d(i+2, i+3) > d(i, i+2) + d(i+1, i+3)$, swap c_i and c_{i+1}.

Step 2: if $i < n$, then $i \longleftarrow i + 1$ and go to Step 1. Otherwise, the algorithm is complete.

The optimized results are shown in Figure 4.

3.4. Avoiding Stagnation. AHACO draws on the behavior of scout bees in the ABC [16] to jump out of the local optimum. Whenever an optimal solution cannot be updated in a certain number of times in ABC, the solution is discarded. AHACO uses a combination of an elite strategy and jumping out of the local optimum. If the optimal path obtained by the population has not been updated for more than a certain number of times, on the one hand, it shows that the path is a sufficiently good solution; on the other hand, it shows that there is a large amount of pheromones on the path. AHACO will retain this path as an elite to speed up the convergence of the algorithm. At the same time, AHACO reinitializes the pheromone on this path because a large amount of pheromones easily causes the algorithm to fall into a local optimum. In this paper, the parameter "a certain number" is defined as tries and the value is set to $t_{\max}/10$. If tries is too small, normal optimization will be interrupted, and if it is too large, stagnation will not be prevented. This not only retains the higher quality solutions but also avoids stagnation and enhances the search capability of the algorithm. The pseudocode of this process can be seen in lines 30–38 of Algorithm 1.

3.5. Modalities for Updating Pheromones. The traditional way of pheromone updating is shown in equations (4)–(6), where all ants perform pheromone updates after one iteration. This update method enhances the positive feedback of the system and helps the algorithm to converge faster, but it affects the diversity of the algorithm and leads to the degradation of the solution quality. AHACO only selects the shortest path for pheromone update from the two roles of ants. It improves the diversity of the algorithm while avoiding the degradation of the algorithm to random greedy search. The update formula is as follows:

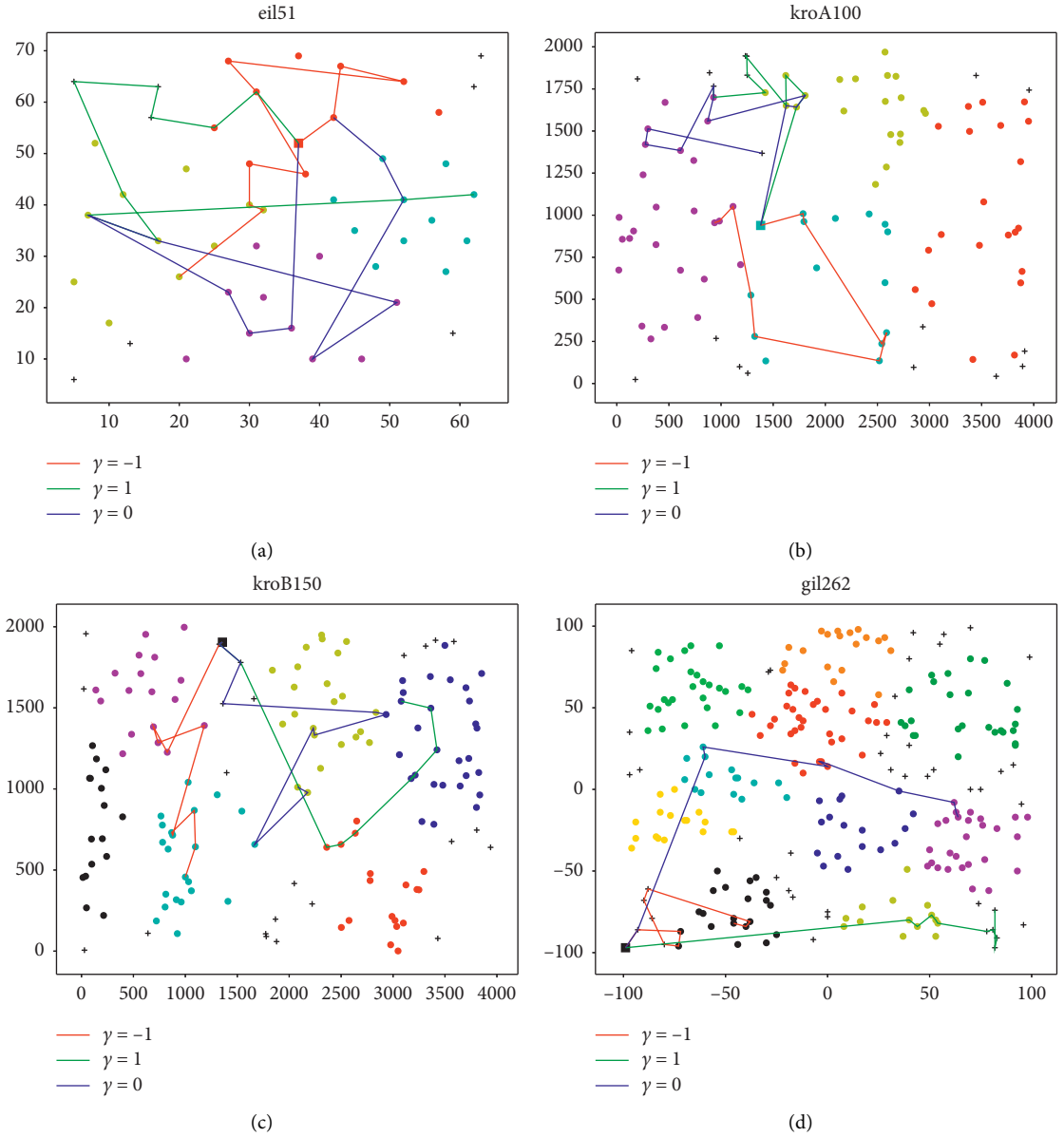

Figure 3: The effect of search strategy on (a) eil51, (b) kroA100, (c) kroB150, and (d) gil262.

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \Delta\tau_{ij}(t), \tag{14}$$

$$\Delta\tau_{ij}(t) = \Delta\tau_{ij}^{\text{normal}}(t) + \Delta\tau_{ij}^{\text{special}}(t), \tag{15}$$

$$\Delta\tau_{ij}^{s}(t) = \begin{cases} \dfrac{Q}{L_{\text{best}}^{s}(t)}, & (i,j) \text{ on tour done by best ant, } s \in \{\text{normal, special}\}, \\ 0, & \text{otherwise,} \end{cases} \tag{16}$$

where $\Delta\tau_{ij}^{\text{normal}}$ represents the pheromone increment of the normal ant, $\Delta\tau_{ij}^{\text{special}}$ represents a pheromone increment for the special ant, Q is the total amount of pheromones, (i, j) represents the path from city i to city j, and L_{best}^{s} denotes the current optimal path length of the ant colony for role s.

3.6. Pseudocode of the AHACO. AHACO primarily classifies cities at the beginning and subsequently rewards or penalizes them according to different city relationships to maintain population diversity during roulette. The specific pseudocode is given in Algorithm 1.

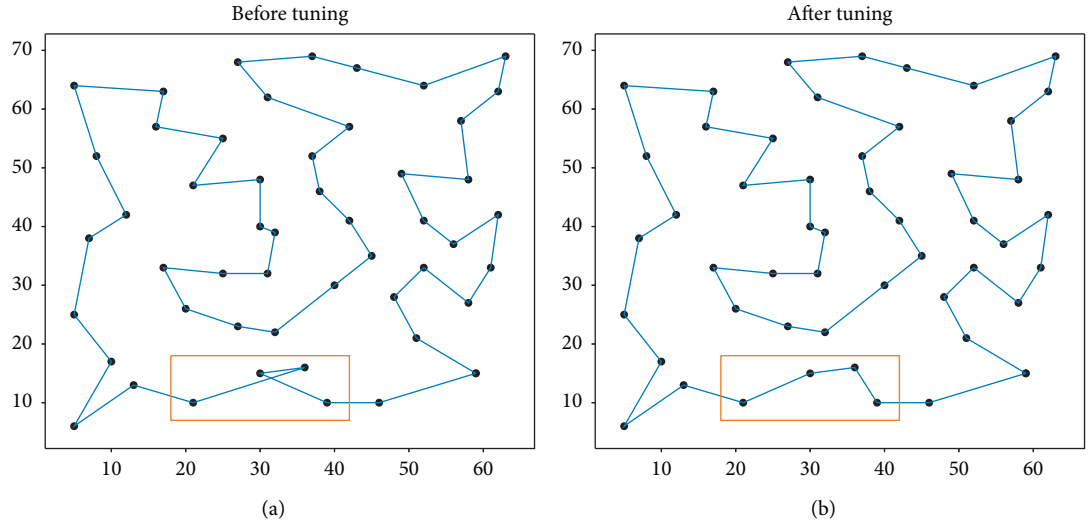

Figure 4: Local optimization. (a) Before tuning. (b) After tuning.

4. Experimental Results

To verify the validity of AHACO, 39 classical examples from TSPLIB [27] are selected in this paper for experiments. It is worth noting that full floating-point operations are used in all experiments without any additional numerical manipulation. TSP instance names are composed of letters and digits, where the digits represent the number of cities corresponding to the instance, that is, its scale. The language used for the implementation is C/C++, the IDE is Visual Studio 2013, the computer CPU is i7-4790K, the memory is 32GB, and the Windows 7 64-bit operating system is used.

4.1. Comparisons. To compare the optimal solution, ACO, DMSO [18], VTPSO [15], and ABCSS [16] were used. The following was implemented: 20 independent runs, respectively, 500 iterations of ACO, DMSO, and VTPSO, and 1000 iterations of ABCSS, with a population size of 300. All other parameters are based on the literature [18]. AHACO uses the following parameters: $\alpha = 1.0$, $\beta = 3.0$, $Q = 120$, $\rho = 0.9$, $\varepsilon = 1.5$, and $\xi \in [1, 8]$. Since AHACO uses ABC scout bees to jump out of the local optimum, the convergence rate will decrease slightly and the number of iterations will have little impact on the final result [18]. The number of iterations in this experiment was chosen to be 1000. The experimental results are shown in Table 1.

Table 1 shows the shortest paths obtained from 20 independent runs of each algorithm. The values are two decimal places, and the optimal terms are in bold. The experiment was conducted to verify the quality of the algorithm's solution, and the values in parentheses in the table are the deviations, in percentages, between the solution obtained by the corresponding algorithm in the corresponding TSP instance and the optimal solution obtained in that instance.

From the data in Table 1, it can be seen that the proposed AHACO algorithm is substantially ahead of the comparison, with 36 out of 39 groups of examples obtaining the optimal solution, accounting for 92.31% of the examples, followed by DSMO, which obtained 7 groups of optimal solutions, accounting for 17.95%. Comparing purely in terms of solution quality, AHACO is 74.36% higher than DSMO, and 76.92%, 82.05%, and 100% higher than ABCSS, VTPSO, and ACO, respectively. The amount of deviation is proportional to the optimization capability of the algorithm. From the data, the AHACO's deviation was 0.09 and the mean deviation was 0.002, compared to 1.53, 2.07, 12.45, and 11.79 for DSMO, VTPSO, ABCSS, and ACO, respectively, which were much higher than that of the AHACO. Overall, almost all algorithms can obtain the optimal solution when the TSP size is less than 70; however, as the TSP size increases, the optimization ability is insufficient and the results obtained are not comprehensible. ABCSS obtains the optimal solution six times, but the average deviation is 12.45%, indicating that the algorithm is unstable and the solution quality varies greatly for different TSP instances. From the data, ABCSS performs very well in solving TSP instances with a scale of less than 300, while it performs poorly for TSP instances with a scale of more than 300. The average deviations of both VTPSO and DSMO are low, and accordingly the algorithm is more stable. AHACO classifies cities and enhances population diversity from the location relationship of cities, which makes it easier for the algorithm to enter the optimal solution region by maximizing the search solution space during the iterative process. At the same time, AHACO uses the method of jumping out of the local optimal solution in the ABC algorithm to better control the precocity of the algorithm. From the data, both the number of optimal solutions and the average deviation of AHACO are significantly ahead of the comparison algorithm, which can obtain the optimal solution more reliably with better overall performance. Figure 5 shows the path diagram of the optimal solution obtained by AHACO on some TSP instances. A more intuitive comparison is shown in Figure 6.

Table 1: Performance comparison among ACO, VTPSO, DSMO, and AHACO on TSPs.

SL	Instance	Best solution (deviation %)				
		ACO	VTPSO	ABCSS	DSMO	AHACO
1	burma14	31.21 (1.1)	**30.87 (0)**	**30.87 (0)**	**30.87 (0)**	**30.87 (0)**
2	ulysses16	77.13 (4.24)	**73.99 (0)**	**73.99 (0)**	**73.99 (0)**	**73.99 (0)**
3	ulysses22	84.78 (12.57)	**75.31 (0)**	**75.31 (0)**	**75.31 (0)**	**75.31 (0)**
4	eil51	499.92 (16.57)	429.51 (0.15)	428.98 (0.03)	**428.86 (0)**	428.87 (0)
5	berlin52	7870.45 (4.32)	**7544.37 (0)**	**7544.37 (0)**	**7544.37 (0)**	**7544.37 (0)**
6	st70	734.19 (8.43)	682.57 (0.81)	682.57 (0.81)	**677.11 (0)**	**677.11 (0)**
7	eil76	581.42 (6.61)	559.25 (2.54)	550.24 (0.89)	558.68 (2.44)	**545.39 (0)**
8	pr76	127025.9 (17.44)	109586.1 (1.32)	108879.7 (0.67)	**108159.4 (0)**	**108159 (0)**
9	gr96	563.77 (10.31)	515.27 (0.82)	512.2 (0.22)	518.38 (1.43)	**511.06 (0)**
10	rat99	1366.3 (12.06)	1256.25 (3.04)	1242.32 (1.89)	1225.56 (0.52)	**1219.24 (0)**
11	kroA100	24504.9 (15.13)	21307.44 (0.1)	21299 (0.06)	21298.21 (0.06)	**21285.4 (0)**
12	kroB100	24664.13 (10.95)	22475.67 (1.11)	**22229.71 (0)**	22308 (0.35)	22234.5 (0.02)
13	rd100	9120.92 (15.3)	8094.75 (2.33)	7944.32 (0.43)	8041.3 (1.65)	**7910.39 (0)**
14	eil101	715.35 (11.22)	653.16 (1.55)	646.05 (0.45)	648.66 (0.85)	**643.17 (0)**
15	lin105	15364.58 (6.82)	14581.58 (1.38)	14406.12 (0.16)	**14383 (0)**	**14383 (0)**
16	pr107	46317.71 (4.37)	44436.25 (0.13)	44525.68 (0.33)	44385.86 (0.02)	**44379.1 (0)**
17	pr124	65145.25 (10.36)	61076.73 (3.47)	**59030.74 (0)**	60285.21 (2.13)	59074.8 (0.07)
18	pr136	110872.2 (14.39)	99247.01 (2.4)	97853.91 (0.96)	97538.68 (0.63)	**96925.5 (0)**
19	gr137	896.07 (26.42)	714.18 (0.76)	713.91 (0.72)	709.48 (0.1)	**708.79 (0)**
20	kroA150	30546.06 (14.12)	27232.1 (1.74)	26981.98 (0.81)	27591.44 (3.09)	**26765.7 (0)**
21	kroB150	29124.59 (11.24)	26579.73 (1.52)	26760.79 (2.21)	26601.94 (1.6)	**26175.4 (0)**
22	pr152	79153.02 (7.42)	74414.17 (0.99)	74337.62 (0.89)	74243.91 (0.76)	**73683.6 (0)**
23	u159	47514.43 (12.93)	43579.82 (3.57)	42862.51 (1.87)	42598.3 (1.24)	**42075.7 (0)**
24	rat195	2534.83 (8.09)	2452.92 (4.6)	2469.31 (5.3)	2372.89 (1.19)	**2345.09 (0)**
25	d198	17301.47 (8.86)	16066.44 (1.09)	16270.22 (2.38)	15978.13 (0.54)	**15892.6 (0)**
26	kroA200	34547.69 (17.27)	30602.81 (3.88)	30701.86 (4.22)	30481.35 (3.47)	**29460 (0)**
27	kroB200	34207.79 (15.04)	30767.52 (3.47)	31508.85 (5.97)	30716.5 (3.3)	**29734.7 (0)**
28	gr202	545.33 (11.36)	497.02 (1.5)	507.27 (3.59)	501.83 (2.48)	**489.69 (0)**
29	tsp225	4396.39 (13.01)	4095.01 (5.26)	4140.24 (6.42)	4013.68 (3.17)	**3890.32 (0)**
30	pr226	90501.46 (12.59)	81050.23 (0.83)	82266 (2.34)	83587.98 (3.99)	**80382.5 (0)**
31	gr229	1865.63 (11.87)	1676.46 (0.53)	1713.54 (2.75)	1683.45 (0.95)	**1667.62 (0)**
32	gil262	2730.52 (13.22)	2547.16 (5.61)	2713.75 (12.52)	2543.15 (5.45)	**2406.84 (0)**
33	pr299	56700.65 (16.99)	50571.83 (4.34)	64464.76 (33.01)	50579.82 (4.36)	**48466 (0)**
34	lin318	47442.95 (12.29)	44724.38 (5.85)	55744.52 (31.94)	44118.66 (4.42)	**42250.7 (0)**
35	linhp318	47577.77 (12.61)	44337.02 (4.94)	56834.19 (34.52)	43831.44 (3.74)	**42250.7 (0)**
36	fl417	13296.85 (10.47)	12376.53 (2.82)	22253.99 (84.89)	12218.98 (1.52)	**12036.6 (0)**
37	gr431	2334.63 (17.76)	2021.95 (1.99)	3284.99 (65.7)	1993.15 (0.54)	**1982.46 (0)**
38	pr439	127228 (17.05)	112088 (3.12)	206233.14 (89.74)	112105.2 (3.14)	**108679 (0)**
39	d493	39254 (6.99)	37132.09 (1.21)	68556.68 (86.86)	36844.63 (0.43)	**36440.6 (0)**
Optimal number/ratio		0/0	4/10.26	6/15.38	7/17.95	**36/92.31**
Deviation/average		459.79/11.79	80.77/2.07	485.55/12.45	59.56/1.53	**0.09/0.002**

4.1.1. Comparison with GA-Based Algorithm. Genetic algorithm is a very classic intelligent algorithm. At present, many studies have applied it to solve the TSP problem and achieved good results. In order to reflect the effectiveness of the proposed AHACO, tests compared with the MO-GA [6] are introduced. In this test, the parameters selected by MO-GA are the same as those in [6], and the population size and number of iterations of AHACO are also set to the same values. From the data in Table 2, it can be seen that the proposed AHACO algorithm can converge to a better solution faster, and the performance is stronger than that of MO-GA. It is worth noting that on the instance kroB100, the results obtained by AHACO are worse than those of MO-GA. This is because the proposed AHACO algorithm uses k-means to classify cities, and the distribution of cities in kroB100 is uniform. Relatively, the classification result is greatly affected by the initial center. On this basis, although this improvement speeds up the convergence of the algorithm, it makes the algorithm easier to fall into the local optimum to a certain extent.

4.2. Statistical Comparison of Convergence. In the comparison described in Section 4.1, the solution quality and stability of the proposed algorithm AHACO are compared, and this section carries out the comparison of convergence speed. AHACO is an improvement on ACO, while VTPSO and DSMO are non-ACO class algorithms with very different mechanisms; in order to reflect the fairness of the experiment, ACS [20] is chosen as the comparison for the

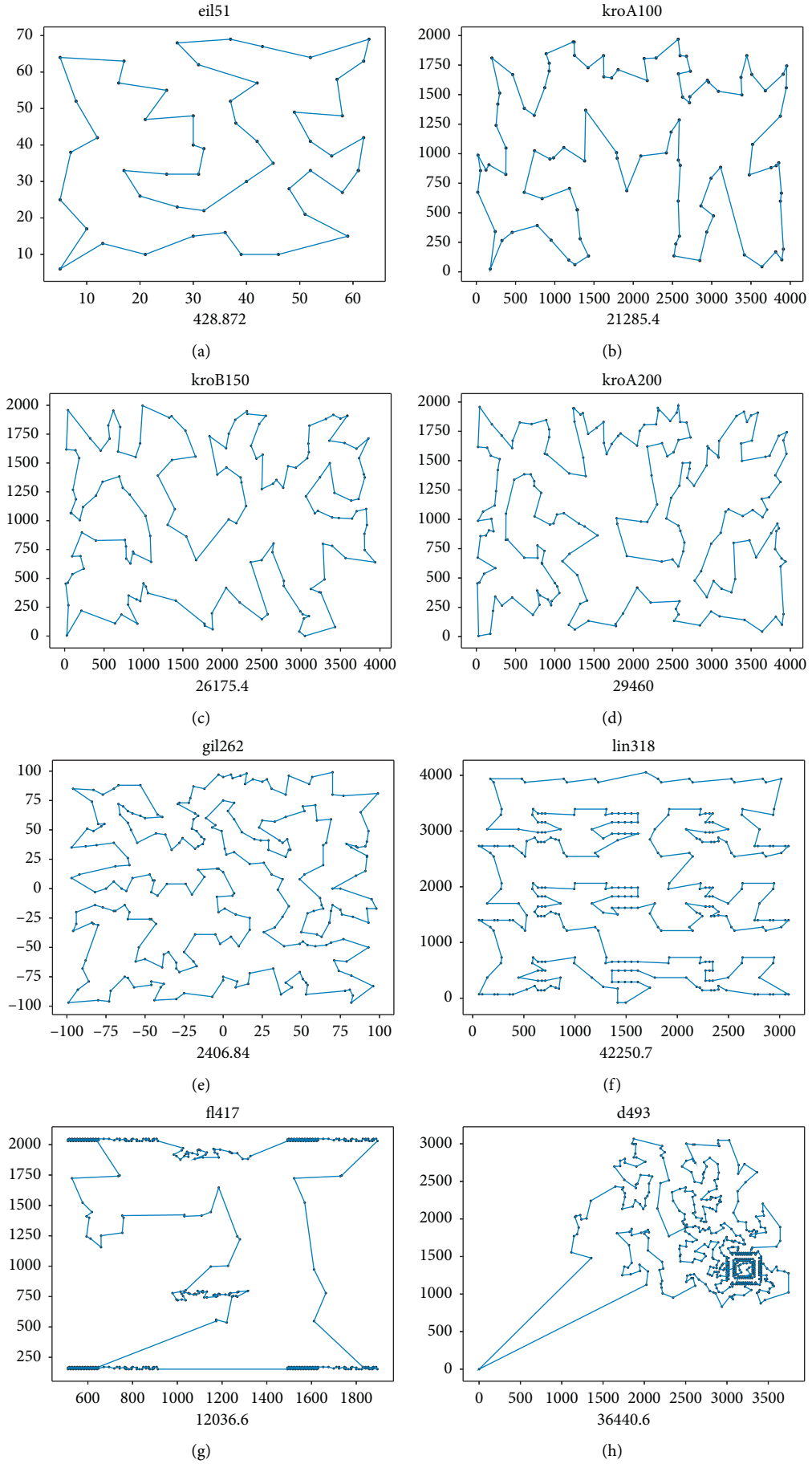

Figure 5: Optimal path. (a) eil51. (b) kroA100. (c) kroB150. (d) kroA200. (e) gil262. (f) lin318. (g) fl417. (h) d493.

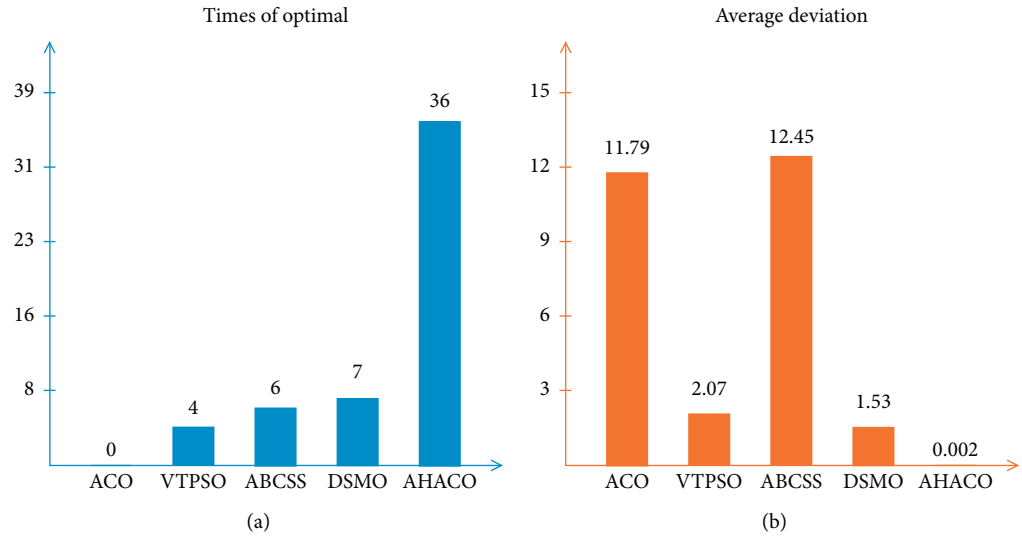

Figure 6: Performance comparison. (a) Times of optimal obtained. (b) Average deviation of solutions.

Table 2: Comparison with MO-GA.

		burma14	eil51	kroB100
Average optimal solution	MO-GA	30.87	442.19	22141.27
	AHACO	30.87	428.87	22297.58
Average number of iterations	MO-GA	9	268	389
	AHACO	12	52	117

convergence speed in this paper. ACS is an improved and widely used algorithm proposed by the proponents of the ant colony algorithm, which is more balanced in terms of convergence speed, solution quality, and stability.

In this experiment, examples of eil51, krob150, lin318, and d493 were selected. The reason for selecting these TSP instances is that the number of cities in these four TSP instances ranges from 51 to 493, covering different scales, which can better test the convergence speed of the proposed algorithm. AHACO and ACS use the same parameters as in Section 4.1. Four TSP instances were tested 100 times, with 1000 iterations performed each time. The statistical distribution is shown in Figure 7.

The horizontal coordinate in Figure 7 is the number of iteration generations when the optimal solution is obtained, and the vertical coordinate is the number of iterations. It can be seen from the figure that the number of iteration generations for AHACO to reach the optimal is significantly smaller than that of ACS and relatively concentrated, indicating that the AHACO is very stable and has a strong convergence rate. In contrast, the ACS graph is more scattered, indicating that the algorithm converges at a low rate and is less stable. Meanwhile, the number of iterative generations for AHACO to reach the optimum increases steadily with the problem size, indicating that the algorithm is highly robust. In addition, in combination with the solution quality of Table 1, the high quality of the solution is obtained in fewer iterations, indicating that the algorithm has a good ability to jump out of the local optimum. Compared with ACS, AHACO mainly has three additional operations: (1) k-means at the beginning of the algorithm; (2) an exponential operation every time the next city is selected; and (3) a 2-opt operation with $O(n)$ time complexity for the optimal solution. Considering the computing power of a modern computer, the sum of these operations is far smaller than the operations required for one iteration, so this can be ignored to some extent. On the whole, AHACO has a high convergence speed and a strong ability to jump out of the local optimum.

4.3. Discussion of Parameter ξ. The ξ parameter has a large influence on the algorithm and is the core improvement point of AHACO. This section takes $\xi_{\min} = 1$, $\xi_{\max} \in \{2, 4, 8, 16\}$, respectively. Experiments on TSP instances eil51, kroA100, gil262, and pr439 use the same parameters as in Section 4.1. Figure 8 shows a graph of the experimental results. The vertical axis for the logarithm of the obtained value minus the difference between the known optimal value plus 1 is $\log_{10}(\text{value} - \text{optimal} + 1)$, and the horizontal and vertical axes for the number of iterations correspond to the value obtained. Among them, the initial population optimal solutions of the improved algorithm are all much better than ACS ($\xi = 1$), which means that when $\xi > 1$, the population diversity is higher, and the algorithm searches the solution space to a greater extent. In both instances, stagnation occurred at $\xi = 1$, while at $\xi > 1$, although the quality of the solution was different, both were

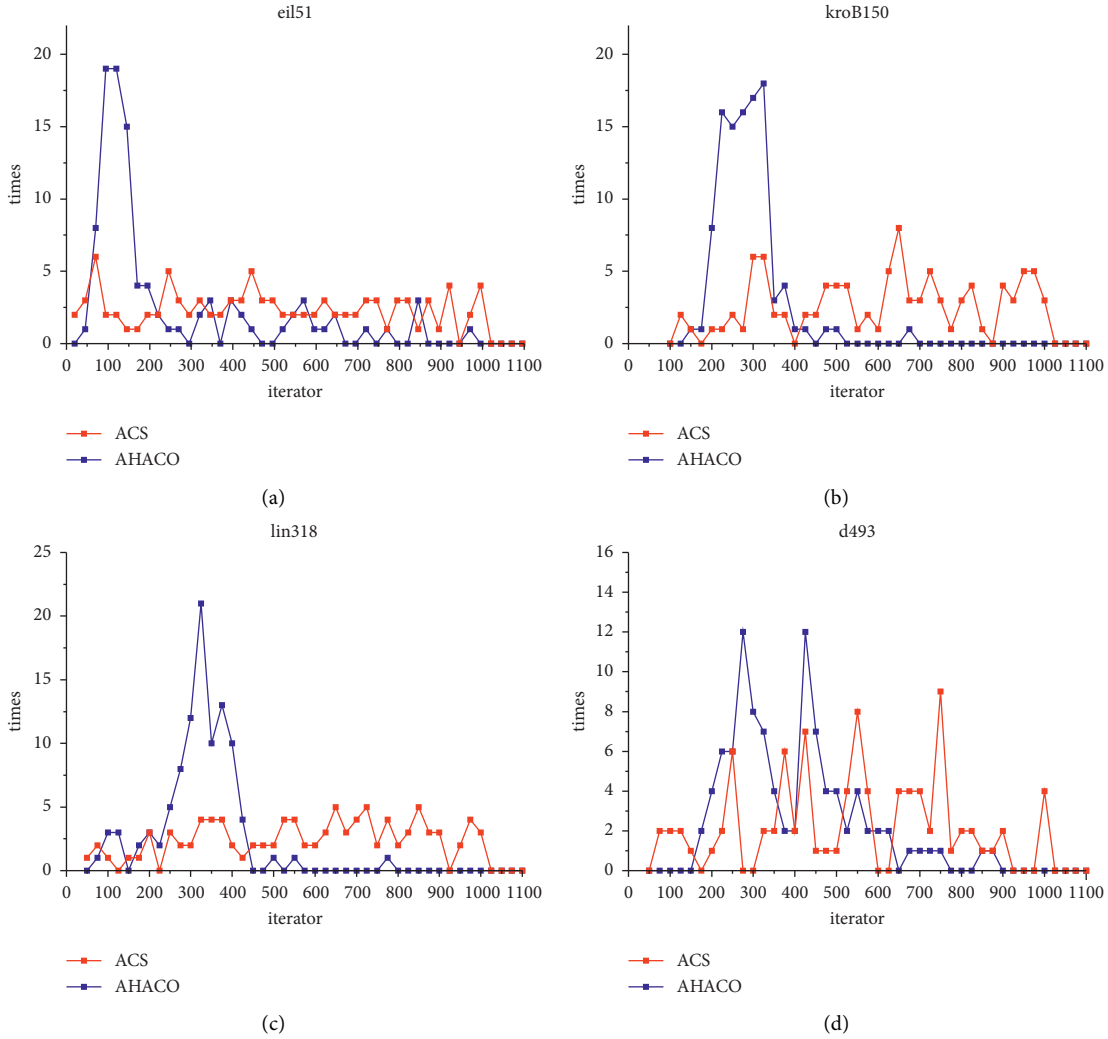

Figure 7: Optimal solution statistical distribution. (a) eil51. (b) kroB150. (c) lin318. (d) d493.

continuously optimized and did not stagnate, which clearly benefited from a better population diversity. For the four cases where $\xi > 1$, ξ_{max} works best when it is 8, followed by 4, and there is little difference in effect between values 2 and 16. When $\xi_{max} = 16$, individual ants are more likely to choose interclass cities, the randomness of the population is enhanced, which is not conducive to convergence, and the overall effect is poor. When $\xi_{max} = 2$, individual ants select interclass cities with less probability, the diversity of the population is not sufficiently enhanced, which is not conducive to searching the solution space, and the overall effect is, again, poor. The main reason why $\xi_{max} = 4$ is not as effective as $\xi_{max} = 8$ is that in the second half of the iteration, the convergence speed is insufficient, resulting in $\xi_{max} = 4$ consuming more iterations for the same solution mass. From the present experiments, the setting of the parameter effectively improves the population diversity of the algorithm and substantially increases the convergence speed of the algorithm, which achieves the desired effect.

4.4. Ablation Experiment. In order to verify the effectiveness of the improvements proposed in this paper, this experiment introduces two intermediate algorithms, corresponding to each improvement, defined as follows: (1) ACS + k-means (A1) and (2) A1 + 2-opt (A2). With the basic ACS and AHACO (A2 + scout-bees), a total of four algorithms are compared. In this experiment, four examples of eil51, kroA100, gil262, and lin318 are selected, and each algorithm is run separately on the example 20 times, with 1000 iterations performed each time, and the average value is calculated for comparison. Other parameters of algorithms are the same as in Section 4.1.

It can be seen in Table 3 that the result of A1 is significantly better than that of the ACS, which shows that the algorithm uses the spatial information of the TSP instance, and the population diversity is greatly improved. The result of A2 is slightly better than that of A1, which shows that the proposed 2-opt algorithm is tuning. AHACO is better than (except eil51) A2, which shows that the algorithm can jump

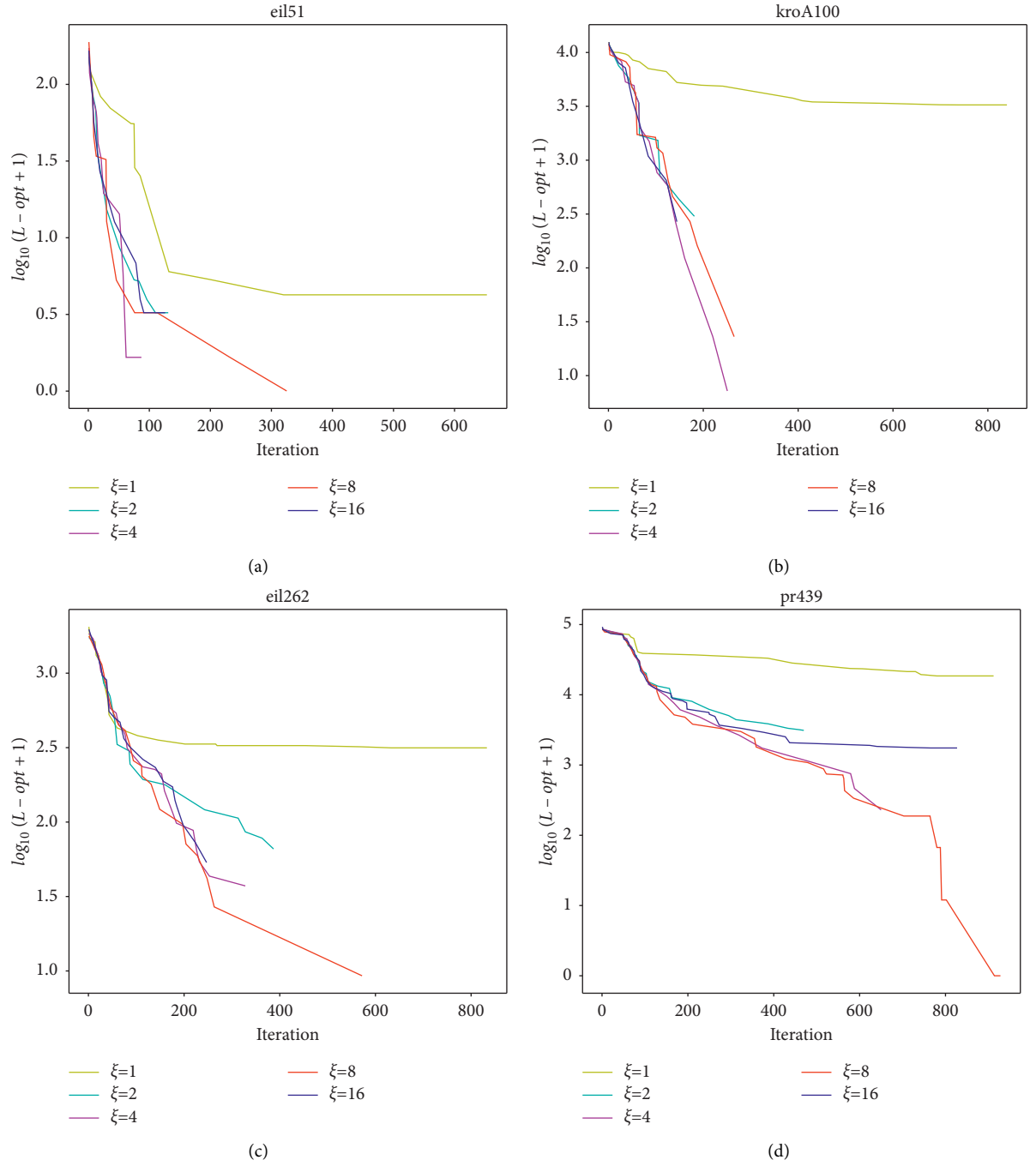

Figure 8: ξ's impact. (a) eil51. (b) kroA100. (c) eil262. (d) pr439.

Table 3: Ablation experiment.

	ACS	A1	A2	AHACO
eil51	433.88	430.56	429.11	429.48
kroA100	21445.8	21292.3	21285.4	21285.4
gil262	2439.61	2424.31	2418.64	2411.85
lin318	45024.7	42407	42360.9	42267.2

out of the local optimum. Due to the small scale of instance eil51, the results are somewhat random, but the overall results are not affected. The idea of this paper is to use spatial information to improve the search ability of the algorithm to a large extent, introduce 2-opt for tuning, and introduce the scout bees mechanism to avoid stagnation. This experiment shows the effectiveness of the proposed improvement.

4.5. Analysis of Time Consumption. In order to verify the running time of the proposed algorithm, this section conducts experiments on the TSP instances eil51, kroA100, kroB200, lin318, fl417, and d493 with the population sizes of 20, 40, 100, and 200. All parameters of the algorithm are the

Table 4: Time consumption (millisecond per iteration).

	eil51	kroA100	kroB200	lin318	fl417	d493
popsize = 20	0.36	0.84	3.14	7.63	12.22	17.44
popsize = 40	0.74	1.86	6.26	13.26	23.61	33.07
popsize = 100	1.5	4.22	14.8	32.25	56.48	80.91
popsize = 200	2.71	8.65	29.06	64.09	112.11	162.07

same as in Section 4.1. The algorithm implementation does not use a parallel mechanism and is implemented in a single thread. The program only uses the "/O2" parameter during the compilation process, without other optimization parameters. The experimental results are shown in Table 4.

Since AHACO has not changed the framework of ACO, this will not affect the analysis of time complexity, so the following analysis of ACO is carried out. Without considering system scheduling and other interference factors, it can be seen in Table 4 that the time consumption of ACO has a linear relationship with the population size and a square relationship with the number of cities. For the convenience of expression, we set m as the number of cities. The path generation part of the ACO is the most time-consuming process. This part includes m selections, each time selecting $m/2$ cities on average. Therefore, its time complexity is $O(n^2)$. Each individual in the population performs the same calculation, so its time complexity is obviously $O(n)$. Iteration is a process of repeated actions in a loop, and the time complexity is also $O(n)$. The data show a great advantage of the ACO algorithm, that is, it has very strong robustness. In addition, the ACO will not increase rapidly in complexity as the scale of the problem becomes larger. From an implementation point of view, ACO is naturally easy to parallelize. A highly parallelized implementation, depending on the degree of parallelization, can almost reduce the time consumption of the algorithm proportionally. Compared with the precise algorithm, under the adaptive setting, ACO can obtain an acceptable solution within an acceptable time. In fact, this is a fundamental feature of metaheuristic algorithms.

5. Conclusions

Based on the study of the ant colony algorithm and the use of the information contained in the problem itself, an improved ant colony algorithm based on an adaptive heuristic factor (AHACO) is proposed in this paper. The traditional improved AHACO takes less consideration of the diversity of ant colonies, and its search ability is limited. Based on the TSP city classification, AHACO classifies the roles of ants and introduces the adaptive parameter ξ, which greatly improves the diversity of ants and improves the solution accuracy and convergence speed. AHACO is not limited to solving TSP problems. In the future, AHACO will be applied in other contexts.

Data Availability

The data that support the findings of this study are openly available at https://doi.org/10.6084/m9.figshare.13492158.v1.

Conflicts of Interest

The authors declare that there are no conflicts of interest.

Acknowledgments

This research was funded by the National Natural Science Foundation of China (NSFC) under grant no. 61872187.

References

[1] S. S. Rajesh Matai and M. L. Mittal, *Traveling Salesman Problem: An Overview of Applications, Formulations, and Solution Approaches*, InTech, West Palm Beach, FL, USA, 2011.

[2] K. Sörensen, "Metaheuristics-the metaphor exposed," *International Transactions in Operational Research*, vol. 22, no. 1, pp. 3–18, 2015.

[3] K. Sörensen, M. Sevaux, and F. Glover, "A history of metaheuristics," in *Handbook of Heuristics*, pp. 791–808, Springer, Cham, Berlin, 2018.

[4] I. K. Gupta, S. Shakil, and S. Shakil, "A hybrid GA-PSO algorithm to solve traveling salesman problem," *Computational Intelligence: Theories, Applications and Future Directions—Volume I*, vol. 798, pp. 453–462, 2019.

[5] X. Dong and Y. Cai, "A novel genetic algorithm for large scale colored balanced traveling salesman problem," *Future Generation Computer Systems*, vol. 95, pp. 727–742, 2019.

[6] J. Wang, O. K. Ersoy, M. He, and F. Wang, "Multi-offspring genetic algorithm and its application to the traveling salesman problem," *Applied Soft Computing*, vol. 43, pp. 415–423, 2016.

[7] X. Dong, H. Zhang, M. Xu, and F. Shen, "Hybrid genetic algorithm with variable neighborhood search for multi-scale multiple bottleneck traveling salesmen problem," *Future Generation Computer Systems*, vol. 114, pp. 229–242, 2021.

[8] C. Sudipta, M. Mohammad, T. Huseyin, L. Bian, and B. William, "A modified ant colony optimization algorithm to solve a dynamic traveling salesman problem: a case study with drones for wildlife surveillance," *Journal of Computational Design and Engineering*, vol. 6, pp. 368–386, 2019.

[9] Ş. Gülcü, M. Mahi, Ö. K. Baykan, and H. Kodaz, "A parallel cooperative hybrid method based on ant colony optimization and 3-opt algorithm for solving traveling salesman problem," *Soft Computing*, vol. 22, no. 5, pp. 1669–1685, 2018.

[10] S. Ebadinezhad, "DEACO: adopting dynamic evaporation strategy to enhance ACO algorithm for the traveling salesman problem," *Engineering Applications of Artificial Intelligence*, vol. 92, Article ID 103649, 2020.

[11] F. Dahan, K. El Hindi, H. Mathkour, and H. AlSalman, "Dynamic flying ant colony optimization (DFACO) for solving the traveling salesman problem," *Sensors*, vol. 19, 2019.

[12] Y. Zhong, J. Lin, L. Wang, and H. Zhang, "Discrete comprehensive learning particle swarm optimization algorithm with metropolis acceptance criterion for traveling salesman problem," *Swarm and Evolutionary Computation*, vol. 42, pp. 77–88, 2018.

[13] W. N. Chen, J. Zhang, H. S. H. Chung, W. L. Zhong, W. G. Wu, and Y. H. Shi, "A novel set-based particle swarm optimization method for discrete optimization problems," *IEEE Transactions on Evolutionary Computation*, vol. 14, no. 2, pp. 278–300, 2010.

[14] M. Mahi, Ö. K. Baykan, and H. Kodaz, "A new hybrid method based on particle swarm optimization, ant colony

optimization and 3-opt algorithms for traveling salesman problem," *Applied Soft Computing*, vol. 30, pp. 484–490, 2015.

[15] M. A. H. Akhand, Z. Jannat, and K. Murase, "Capacitated vehicle routing problem solving using adaptive sweep and velocity tentative PSO," *International Journal of Advanced Computer Science and Applications*, vol. 8, 2017.

[16] I. Khan and M. K. Maiti, "A swap sequence based artificial bee colony algorithm for traveling salesman problem," *Swarm and Evolutionary Computation*, vol. 44, pp. 428–438, 2018.

[17] C. S. Siang, W. Li-Pei, and L. C. Peng, "An artificial bee colony algorithm with a modified choice function for the traveling salesman problem," *Swarm and Evolutionary Computation*, vol. 44, pp. 622–635, 2019.

[18] M. A. H. Akhand, S. I. Ayon, S. A. Shahriyar, N. Siddique, and H. Adeli, "Discrete spider monkey optimization for travelling salesman problem," *Applied Soft Computing*, vol. 86, Article ID 105887, 2020.

[19] I. M. Ali, D. Essam, and K. Kasmarik, "A novel design of differential evolution for solving discrete traveling salesman problems," *Swarm and Evolutionary Computation*, vol. 52, Article ID 100607, 2020.

[20] M. Dorigo and L. M. Gambardella, "Ant colony system: a cooperative learning approach to the traveling salesman problem," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 53–66, 1997.

[21] Y. Yan, H.-S. Sohn, and G. Reyes, "A modified ant system to achieve better balance between intensification and diversification for the traveling salesman problem," *Applied Soft Computing*, vol. 60, pp. 256–267, 2017.

[22] J. B. Escario, J. F. Jimenez, and J. M. Giron-Sierra, "Ant colony extended: experiments on the travelling salesman problem," *Expert Systems with Applications*, vol. 42, no. 1, pp. 390–410, 2015.

[23] A. Delévacq, P. Delisle, M. Gravel, and M. Krajecki, "Parallel ant colony optimization on graphics processing units," *Journal of Parallel and Distributed Computing*, vol. 73, no. 1, pp. 52–61, 2013.

[24] Y. Liu, "Research on the algorithm optimization of improved ant colony algorithm-LSACA," *International Journal of Signal Processing, Image Processing and Pattern Recognition*, vol. 9, no. 3, pp. 143–154, 2016.

[25] H. Xu, X. Qian, and L. Zhang, "Study of ACO algorithm optimization based on improved tent chaotic mapping," *Journal of Information and Computational Science*, vol. 9, pp. 1653–1660, 2012.

[26] W. Lei and F. Wang, "Research on an improved ant colony optimization algorithm for solving traveling salesmen problem," *International Journal of Database Theory and Application*, vol. 9, no. 9, pp. 25–36, 2016.

[27] G. Reinelt, "TSPLIB—a traveling salesman problem library," *ORSA Journal on Computing*, vol. 3, no. 4, pp. 376–384, 1991.