Computers, Materials & Continua

Tech Science Press

https://doi.org/10.32604/cmc.2026.087350

ARTICLE

A Discrete Crested Porcupine Optimizer for the Spherical Asymmetric Traveling Salesman Problem

Honglei Ma[1], Yingxuan Luo[2], Jie Li[3,*] and Jia Chen[1]

[1]School of Computer Application, Guilin University of Technology, Guilin, China
[2]Department of Mechanical Engineering, The University of Hong Kong, Hong Kong, China
[3]School of Artificial Intelligence, South China Normal University, Foshan, China

*Corresponding Author: Jie Li. Email: lijie@m.scnu.edu.cn

Received: 15 June 2026; Accepted: 06 August 2026; Published: 15 September 2026

ABSTRACT: The Spherical Asymmetric Traveling Salesman Problem (SATSP), characterized by spherical geometry and direction-dependent travel costs, is a challenging combinatorial optimization problem, particularly in large-dimensional scenarios. Although the recently proposed Crested Porcupine Optimizer (CPO) has shown promising performance in continuous optimization, its applicability to discrete asymmetric routing problems remains largely unexplored. To address this limitation, we propose a Discrete Crested Porcupine Optimizer (DCPO), which integrates a discrete solution representation with dual crossover operators, namely order crossover and partially mapped crossover, as well as a multi-strategy mutation mechanism including inversion mutation and swap mutation. A 2-opt local search strategy is embedded to enhance local refinement while maintaining global exploration. DCPO is applied to SATSP instances to validate the effectiveness and stability of the proposed algorithm. Experimental comparisons with several representative metaheuristic algorithms, including the Discrete Mayfly Algorithm (DMA), Flower Pollination Algorithm (FPA), Discrete Sparrow Search Algorithm (DSSA), Genetic Algorithm (GA), Randomized Bias Genetic Algorithm (RBGA), and Discrete Marine Predators Algorithm (DMPA), demonstrate that DCPO obtains the best mean and best-found objective values in 11 out of 12 SATSP instances of different scales, corresponding to 91.7%, and achieves the lowest standard deviation in 8 instances, corresponding to 66.7%, indicating strong solution quality and stability. For large-dimensional scenarios with 1000 to 1200 cities, DCPO achieves improvements of 13.09%–19.46% in mean objective values over the best competing algorithms, further confirming its effectiveness and scalability for complex SATSP optimization.

KEYWORDS: Spherical asymmetric traveling salesman problem; discrete crested porcupine optimizer; metaheuristic; 2-opt local search; discrete optimization

1 Introduction

The Traveling Salesman Problem (TSP) is a classical NP-hard combinatorial optimization problem with significant theoretical and practical value [1]. Its objective is to find a minimum-cost tour that visits each city exactly once and returns to the starting city. Owing to its computational complexity, the TSP has become a benchmark problem in combinatorial optimization and has been widely applied in logistics [2], manufacturing [3], transportation [4], route planning [5], task scheduling [6], and supply chain management [7]. However, many practical routing problems are not confined to planar space. In global-scale applications, spherical geometry and asymmetric travel costs must be considered, leading to the Spherical Asymmetric Traveling Salesman Problem (SATSP). Compared with the classical Euclidean TSP, SATSP

introduces additional challenges in distance calculation, route representation, and neighborhood search, thereby reducing the effectiveness of many existing optimization algorithms.

To address the challenges introduced by spherical geometry and asymmetric travel costs, various methods have been developed for spherical TSP, SATSP, and related complex TSP variants. Existing studies can be broadly classified into three categories. The first category focuses on spherical routing problems. Symmetric spherical TSP has been addressed using Ant Colony Optimization (ACO) [8] and the Discrete Greedy Flower Pollination Algorithm (DGFPA) [9], whereas asymmetric spherical variants have been studied using the Discrete Mayfly Algorithm (DMA) for SATSP [10] and Spherical Vector-Based Artificial Gorilla Troops Optimization (SGTO) for the spherical asymmetric multiple TSP [11]. The second category comprises problem-specific discrete and hybrid methods developed for conventional or constrained TSP variants. Discrete approaches include the Discrete Marine Predators Algorithm (DMPA) for symmetric TSP [12], an improved discrete Bat Algorithm (IDBA) for symmetric and asymmetric TSP [13], a discrete Water Cycle Algorithm (DWCA) for the symmetric TSP (STSP) and asymmetric TSP (ATSP) [14], a discrete Differential Evolution (DDE) method for TSP [15], and discrete Pigeon-Inspired Optimization (DPIO) for large-scale TSP [16]. Hybrid metaheuristics include Growth Optimization with local improvement [17], discrete search with greedy-beam-search initialization and edge-based operators [18], elite-guided memetic framework for a constrained colored TSP variant with time windows [19], and a memetic algorithm with optimal recombination for ATSP [20]. The third category includes deterministic search, learning-based models, and alternative encoding strategies, such as probe-based search [21], topology-aware attention networks [22], Keplerian TSP modeling [23], and Bloch sphere encoding [24]. Beyond these three categories, metaphor-based algorithms have also been adapted to discrete combinatorial problems, including a modified Red Deer Algorithm (RDA) for truck scheduling with time windows [25], a Social Engineering Optimizer (SEO)-based method for vehicle routing [26], and a Genetic Engineering Algorithm (GEA) for the generalized quadratic assignment problem [27]. Overall, these studies show that effective TSP optimization requires problem-specific discrete representations and search mechanisms. However, some discrete methods rely on continuous-to-discrete decoding, limiting direct search in the permutation space. Existing SATSP-oriented algorithms may also incur high computational costs due to intensive local search, while fixed search settings provide limited adaptation to population diversity and solution dissimilarity. These limitations motivate a SATSP-specific framework that directly preserves feasible permutations, adaptively controls local refinement, and evaluates complete directed tour costs.

Against this background, the Crested Porcupine Optimizer (CPO), proposed by Abdel-Basset et al. in 2023 [28], has shown strong potential in continuous optimization tasks. However, the original CPO is designed for continuous-valued search spaces, where solution updates rely on arithmetic operations, making it unsuitable for direct application to permutation-based SATSP. Therefore, this study develops a SATSP-specific discrete reformulation of CPO rather than another general-purpose metaheuristic. In DCPO, each individual is directly represented as a feasible city permutation, and the exploration–exploitation organization of CPO is transformed into permutation-based crossover and mutation processes. Compared with existing metaheuristics, DCPO preserves route feasibility without continuous-to-discrete decoding, adaptively controls local-search intensity through Hamming-distance-guided 2-opt, and evaluates complete directed tour costs to account for the asymmetric costs of reversed internal and boundary arcs. These features make DCPO well suited to SATSP while providing a favorable balance between solution quality and computational cost.

The main contributions of this study are summarized as follows:

1. A SATSP-specific discrete CPO framework is developed to transform the continuous search behavior of CPO into a feasible permutation-based optimization process for spherical asymmetric routing.

2. The four defense mechanisms of CPO are functionally reformulated as complementary permutation-based exploration and exploitation operators, while preserving route feasibility throughout the search.
3. A Hamming-distance-guided 2-opt strategy is introduced to adaptively control local-search intensity, and complete directed tour-cost evaluation is employed to account for the asymmetric costs of reversed internal and boundary arcs.
4. Extensive experiments against representative metaheuristic algorithms verify the competitiveness, robustness, and scalability of DCPO on SATSP instances of different scales.

The remainder of this paper is organized as follows: Section 2 introduces the formulation and characteristics of the SATSP. Section 3 provides an overview of the original Crested Porcupine Optimizer (CPO). Section 4 presents the proposed DCPO. Section 5 presents the experimental setup, performance comparisons, and result analysis. Finally, conclusions and future work are discussed in Section 6.

2 SATSP Problem Formulation

The SATSP formulation in this section is adapted from the spherical ATSP model proposed by Zhang et al. [10], with modifications to the asymmetric distance definition and subtour-elimination constraints. Let $P_1\,(x_1, y_1, z_1)$ and $P_2\,(x_2, y_2, z_2)$ be two points on a sphere of radius r. As shown in Fig. 1, their geodesic distance is determined by the central angle θ:

$$d\,(P_1, P_2) = r \cdot \theta, \tag{1}$$

the central angle is obtained from the dot product of the position vectors:

$$\cos\,(\theta) = \frac{x_1 \cdot x_2 + y_1 \cdot y_2 + z_1 \cdot z_2}{r^2}, \tag{2}$$

$$\theta = \arccos\left(\frac{x_1 \cdot x_2 + y_1 \cdot y_2 + z_1 \cdot z_2}{r^2}\right), \tag{3}$$

to simplify the problem, we assume a unit sphere ($r = 1$). Thus, the geodesic distance can be expressed as:

$$d\,(P_1, P_2) = \arccos\,(x_1 \cdot x_2 + y_1 \cdot y_2 + z_1 \cdot z_2)\,. \tag{4}$$

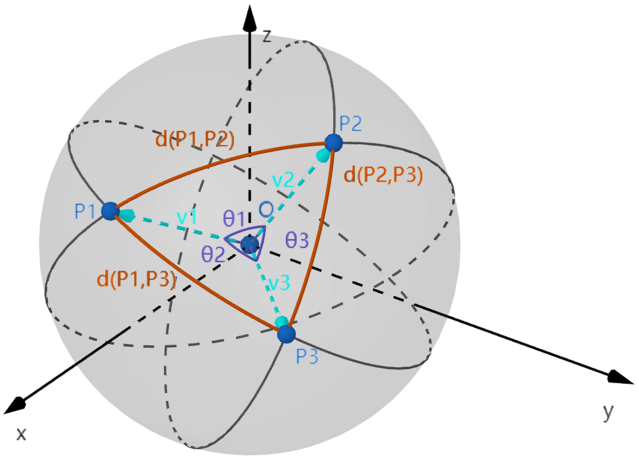


Figure 1: Illustration of geodesic distance on a sphere.

For the symmetric TSP, the travel costs between P_i to P_j equals that from P_j to P_i. In practical routing scenarios, however, factors such as wind, terrain, traffic flow, and operational constraints may produce direction-dependent costs, such that $d_{ij} \neq d_{ji}$. To model this asymmetry, the geodesic distances obtained from Eq. (4) are stored in a symmetric matrix D_s, after which independent perturbations within ±5% are applied to the off-diagonal entries. The diagonal entries remain zero. The perturbation terms ε_{ij} and ε_{ji} are independently generated, so the travel costs in opposite directions are generally different. The entries of the asymmetric distance matrix are defined as follows:

$$d_{ij}^{a} = \begin{cases} d_{ij}^{s}\left(1+\varepsilon_{ij}\right), & i \neq j, \\ 0, & i = j, \end{cases} \quad \varepsilon_{ij} \sim U\left[-0.05, 0.05\right], \tag{5}$$

where d_{ij}^{s} and d_{ij}^{a} denote the (i, j)-th entries of D_s and D_a, respectively. For simplicity, $d_{ij} = d_{ij}^{a}$ is used hereafter.

Let $G = (V, E)$ be a directed complete graph, where $I = \{1, 2, \ldots, n\}$ denotes the index set of cities, $V = \{v_i \mid i \in I\}$ is the set of vertices representing cities on the sphere and $E = \left\{\left(v_i, v_j\right) \mid i, j \in I, i \neq j\right\}$ is the set of directed arcs. Each arc $\left(v_i, v_j\right)$ is assigned an asymmetric spherical travel cost d_{ij}. The SATSP is formulated as follows:

$$Minimize \quad \sum_{i=1}^{n}\sum_{j\neq i}^{n} d_{ij}x_{ij}, \tag{6}$$

$$\sum_{\substack{j=1 \\ j\neq i}}^{n} x_{ij} = 1, \quad \forall i \in I, \tag{7}$$

$$\sum_{\substack{i=1 \\ i\neq j}}^{n} x_{ij} = 1, \quad \forall j \in I, \tag{8}$$

$$u_i - u_j + nx_{ij} \leq n-1, \quad \forall i, j \in I\backslash\{1\}, \; i \neq j, \tag{9}$$

$$1 \leq u_i \leq n-1, \quad \forall i \in I\backslash\{1\}, \quad u_1 = 0, \tag{10}$$

$$x_{ij} \in \{0, 1\}, \quad \forall i, j \in I, \quad i \neq j. \tag{11}$$

In this formulation, the binary variable x_{ij} equals 1 if the directed arc from city i to city j is selected and 0 otherwise. The auxiliary variable u_i represents the visiting order of city i, with $u_1 = 0$ for the reference city. Eq. (6) minimizes the total asymmetric tour cost. Eqs. (7) and (8) ensure that each city has exactly one outgoing and one incoming arc, respectively. Eqs. (9) and (10) eliminate disconnected subtours, while Eq. (11) defines the binary domain of the decision variables. Together, these constraints construct a single closed tour that visits every city exactly once.

3 Overview of the Crested Porcupine Optimizer

The Crested Porcupine Optimizer (CPO), proposed by Abdel-Basset et al. [28], is inspired by four defensive behaviors of crested porcupines: visual deterrence, auditory warning, scent emission, and physical attack. The first two mechanisms support exploration, whereas the latter two promote exploitation.

3.1 The First Defensive Strategy

When a crested porcupine detects a predator, it raises its quills as a visual deterrent. In CPO, a normally distributed random variable determines whether the simulated predator moves toward or away from the porcupine, enabling exploration at different scales. The position of the i-th individual is updated as follows:

$$\vec{x}_i^{t+1} = \vec{x}_i^t + \tau_1 \left| 2\tau_2 \vec{x}_{CP}^t - \vec{y}_i^t \right|, \tag{12}$$

where τ_1 is a normally distributed random number, $\tau_2 \in [0,1]$, and $\overrightarrow{x_{CP}^t}$ denotes the best solution obtained at iteration t. The formula of it is described below:

$$\vec{y}_i^t = \frac{\vec{x}_i^t + \vec{x}_r^t}{2}, \tag{13}$$

where r is a randomly selected integer index from $\{1, 2, \ldots, N\}$, and N denotes the population size.

3.2 The Second Defensive Strategy

The second defensive strategy models the auditory warning behavior of the crested porcupine. In CPO, this mechanism combines the simulated predator position with the positional difference between two randomly selected individuals to promote exploration. The position is updated as follows:

$$\vec{x}_i^{t+1} = \left(1 - \vec{U}_1\right) \vec{x}_i^t + \vec{U}_1 \left(\vec{y}_i^t + \tau_3 \left(\vec{x}_{r_1}^t - \vec{x}_{r_2}^t \right) \right), \tag{14}$$

where, $\vec{U}_1$ is designed as a random binary vector having 0 and 1. τ_3 is a random number between [0, 1], r_1 and r_2 are randomly selected integer indices from $\{1, 2, \ldots, N\}$.

3.3 The Third Defensive Strategy

The third defensive strategy models scent emission and is used to exploit promising regions. The position of the i-th individual is updated as follows:

$$\vec{x}_i^{t+1} = (1 - \vec{U}_1)\vec{x}_i^t + \vec{U}_1 \left[\vec{x}_{r_1}^t + S_i^t \left(\vec{x}_{r_2}^t - \vec{x}_{r_3}^t \right) - \tau_3 \vec{\delta} \gamma_t S_i^t \right], \tag{15}$$

where S_i^t is the odor diffusion factor, $\vec{\delta}$ controls the search direction, γ_t is the defense factor, and $\tau_3 \in [0,1]$. The indices r_1, r_2 and r_3 is randomly selected integer indices from $\{1, 2, \ldots, N\}$.

3.4 The Forth Defensive Strategy

The fourth defensive strategy models the physical attack behavior of the crested porcupine and strengthens exploitation around the current best solution. The position of the i-th individual is updated as follows:

$$\vec{x}_i^{t+1} = \vec{x}_{CP}^t + [\alpha(1 - \tau_4) + \tau_4](\vec{\delta}\vec{x}_{CP}^t - \vec{x}_i^t) - \tau_5 \vec{\delta} \gamma_t \vec{F}_i^t, \tag{16}$$

where $\vec{x}_{CP}^t$ is the best solution obtained at iteration t, α is the convergence-speed factor, $\tau_4, \tau_5 \in [0,1]$, and $\vec{F}_i^t$ represents the average force acting on the i-th individual. The terms $\vec{\delta}$ and γ_t denote the search-direction and defense factors, respectively.

4 The Proposed Discrete Crested Porcupine Optimizer

The original CPO is designed for continuous optimization, and its arithmetic position-update equations cannot be directly applied to permutation-based SATSP. Therefore, this study proposes a Discrete Crested Porcupine Optimizer (DCPO) that represents each solution as a feasible city permutation and reformulates the four defensive mechanisms of CPO using discrete crossover and mutation operators. The correspondence between CPO and SATSP is summarized in Table 1.

Table 1: Correspondence between DCPO and SATSP.

DCPO	SATSP
Position of a porcupine	Visiting order of cities
Dimension of a porcupine's position	Number of cities
Fitness value of a porcupine	Length of the visiting path
Best porcupine position	Optimal path
Best fitness value	Length of the optimal path

DCPO retains the population-based cooperation, probabilistic switching, elitist preservation, and global-best guidance of the original CPO. The first and second defense mechanisms preserve their exploration roles and are transformed into OX and PMX, respectively, which recombine route information while maintaining feasible permutations. The third and fourth mechanisms preserve their exploitation roles and are transformed into inversion and swap mutation, providing segment-level and position-level perturbations, respectively. In addition, Hamming-distance-guided 2-opt is introduced to adaptively control the intensity of local refinement. The complete procedure is presented in Algorithm 1.

Algorithm 1: The Pseudocode of DCPO

Require: N_0: initial population size, N_{min}: minimum population size, T_{max}: maximum iterations, T_f: defense threshold

Ensure: G_b: global best path, F^*: cost of the global best path

Initialize the population and define related parameters
Calculate the initial fitness and obtain the current best solution
$t \leftarrow 1$
while $t < T_{max}$ **do**
  for $i = 1$ **to** N_t **do**
    Generate random numbers $\tau_1, \tau_2 \sim U(0, 1)$
    if $\tau_1 < \tau_2$ then ▷ **Exploration phase**
      if $\tau_1 < 0.8$ **then**
        Perform Order Crossover: $X_i' \leftarrow OX(X_i, X_{rand})$ ▷ First Defense Mechanism
      else
        Perform PMC Crossover: $X_i' \leftarrow PMX(X_i, X_{rand})$ ▷ Second Defense Mechanism
      end if
    else ▷ **Exploitation phase**
      Compute Hamming distance: $H \leftarrow H(X_i, G_b)$
      if $\tau_2 < T_f$ **then**
        Perform Inversion Mutation Operator: $X_i' \leftarrow Reverse(X_i, G_b)$ ▷ Third Defense Mechanism
      else
        Perform Swap Mutation Operator: $X_i' \leftarrow Swap(X_i)$ ▷ Fourth Defense Mechanism
      end if
      Apply 2-opt local optimization: $X_i' \leftarrow 2 - opt(X_i', H)$
    end if
    if $L(\pi') < L(\pi)$ **then**
      $X_i \leftarrow X_i'$ ▷ Elitism preservation

(Continued)

Algorithm 1 (continued)

Update personal best: $X^{i}_{\text{pbest}} \leftarrow X_i$
end if
end for
Update the global best path G_b and its corresponding tour cost $F^{\star}$
$t \leftarrow t+1$
end while

4.1 Crossover Operators in the Exploration Phase

4.1.1 Order Crossover Operator

The core mechanism of the Order Crossover (OX) operator is to generate valid offspring by inheriting the city sequence characteristics from the parents. This operator preserves relative ordering information of the selected path segment from one parent to ensure the feasibility of the offspring, while expanding the search space through the reordering of the remaining cities. Such a design enhances global exploration capability and improves search efficiency, aligning with the porcupine's early-stage defensive behavior associated with global search.

Let the parent paths be denoted as $P_1 = (a_1, a_2, \ldots, a_n)$ and $P_2 = (b_1, b_2, \ldots, b_n)$. Two crossover points k_1 and k_2, satisfying $1 \le k_1 < k_2 \le n$, are randomly selected. The segment between these points in P_1 is copied to the offspring. The remaining cities are then inserted in their order of appearance in P_2, excluding those already contained in the copied segment. The resulting offspring is expressed as:

$$C = OX\left(P_1, P_2; k_1, k_2\right), \tag{17}$$

where C denotes the offspring route. The OX process is illustrated in Fig. 2.

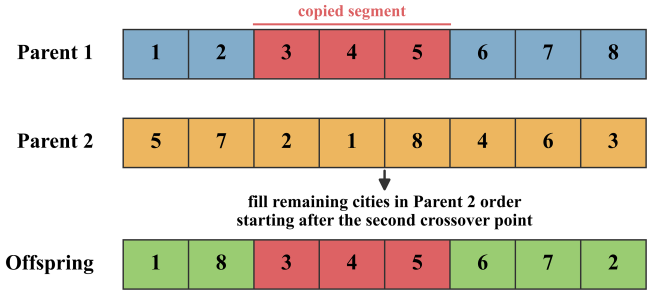


Figure 2: Visualization of the order crossover process in DCPO.

4.1.2 Partially Mapped Crossover Operator

The Partially Mapped Crossover (PMX) operator generates offspring by exchanging selected path segments between two parents and resolving gene conflicts through a bidirectional mapping mechanism. This process preserves the feasibility of the offspring permutation while increasing population diversity. The conflict nodes generated during mapping can be regarded as potential activation points of the second defense mechanism, which is consistent with the global search behavior in the exploration phase.

The PMX operator is illustrated in Fig. 3. Given two parent paths P_1 and P_2, two crossover points are randomly selected to define two equal-length mapping segments, $S_1 = (a_i, a_{i+1}, \ldots, a_{i+l}) \subseteq P_1$ from P_1 and $S_2 = \left(b_j, b_{j+1}, \ldots, b_{j+l}\right) \subseteq P_2$ from P_2. A bidirectional mapping is then established between the corresponding elements of the two segments as follows:

$$\mathcal{M}: a_{i+k} \leftrightarrow b_{j+k}, \quad k = 0, 1, \ldots, l, \tag{18}$$

for a candidate *offspring* solution $C_{draft} = (c_1, c_2, \ldots, c_n)$ the gene c_k is updated according to the following rule:

$$c_k = \begin{cases} b_{j+m} & if\ c_k = a_{i+m} \in S_1 \\ a_{i+m} & if\ c_k = b_{j+m} \in S_2 \\ c_k & otherwise \end{cases} \quad m = 0, 1, \ldots, l, \tag{19}$$

where $k \in \{1, 2, \ldots, n\}$ denotes the position of a gene in the offspring. If the mapped element still conflicts with the copied segment, the mapping procedure is repeated until a nonconflicting element is obtained.

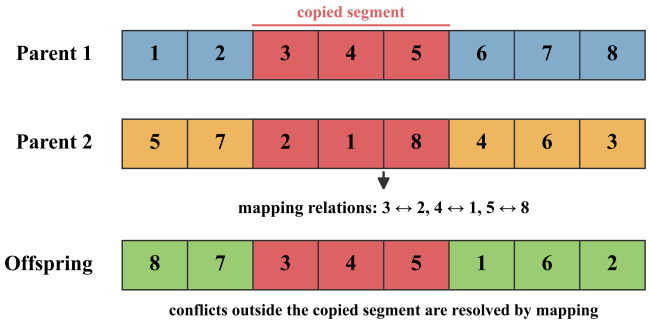


Figure 3: Visualization of the partially mapped crossover process in DCPO.

4.2 Mutation Operators in the Exploitation Phase

4.2.1 Inversion Mutation Operator

The inversion mutation operator generates new candidate solutions by reversing a selected segment within the path. This operation perturbs the original route structure, thereby increasing population diversity and improving local search potential. In DCPO, it is used to simulate the scent-based defensive behavior of the crested porcupine, corresponding to an exploitation-oriented search process. The inversion mutation operator is illustrated in Fig. 4.

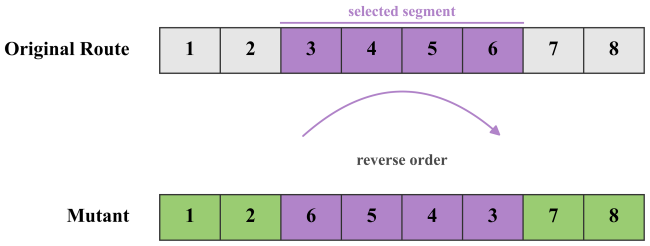


Figure 4: Illustration of the inversion mutation operator.

Let the current path be denoted as $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$. The operation is defined as follows:

$$\pi' = \left(\pi_1, \ldots, \pi_{i-1}, \underbrace{\pi_j, \pi_{j-1}, \ldots, \pi_i}_{reversed\ segment}, \pi_{j+1}, \ldots, \pi_n \right), \tag{20}$$

where i and j are randomly selected index positions satisfying $1 \le i < j \le n$. In SATSP, reversing a segment changes the directions and costs of the internal directed arcs. Therefore, the complete directed tour cost must be recalculated after the inversion mutation. The change in distance is calculated as:

$$\Delta d = \sum_{k=1}^{n} \left(d_{\pi'_k, \pi'_{k+1}} - d_{\pi_k, \pi_{k+1}} \right), \qquad \pi_{n+1} = \pi_1, \quad \pi'_{n+1} = \pi'_1. \tag{21}$$

4.2.2 Swap Mutation Operator

The swap mutation operator introduces controlled perturbations by exchanging two randomly selected elements in the solution sequence. This operation helps increase population diversity and may further improve solution quality. In DCPO, it is used to simulate the crested porcupine's final confrontation-based defensive behavior, corresponding to an intensified local search process. The swap mutation operator is illustrated in Fig. 5. Specifically, two distinct positions k and l are randomly selected, where $k, l \in \{1, \ldots, n\}$ and $k \neq l$, and the mutated path is generated as follows:

$$\pi'_m = \begin{cases} \pi_l & m = k \\ \pi_k & m = l \\ \pi_m & otherwise \end{cases} \quad \forall m \in \{1, \ldots, n\}, \quad k, l \in \{1, \ldots, n\}, \quad k \neq l. \tag{22}$$

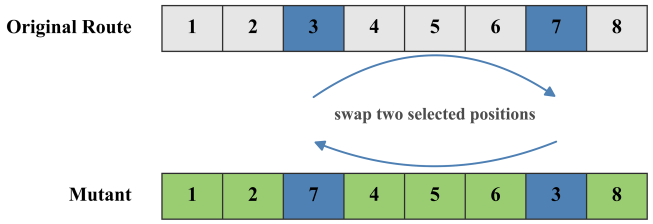


Figure 5: Illustration of the swap mutation operator.

4.3 2-opt Local Optimization

4.3.1 2-opt Optimization Procedure

The 2-opt local search is applied after the mutation operation to improve solution quality and accelerate convergence. As shown in Fig. 6, the 2-opt operator selects two nonadjacent positions in a complete route and reverses the segment between them to generate a candidate route. For the SATSP, reversing a segment changes not only the two boundary arcs but also the directions and costs of all internal arcs. Therefore, the complete directed tour cost of the candidate solution must be recalculated using the asymmetric distance matrix. Given the current path $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$, the optimization proceeds as follows:

Step1: Randomly select two positions i, j such that $1 \le i < j \le n$ and $j - i \ge 2$.

Step2: Reverse the sub-path from position i to position j to generate a candidate solution. This operation is identical to the inversion operation defined in Eq. (20), but is used here for greedy local refinement.

Step3: Recalculate the complete directed tour costs of the current and candidate routes, and apply the following greedy acceptance criterion:

$$\pi_{new} = \begin{cases} \pi' & if\ L(\pi') < L(\pi) \\ \pi & otherwise, \end{cases} \tag{23}$$

where $L(\pi) = \sum_{k=1}^{n} d_{\pi_k,\pi_{k+1}}, \pi_{n+1} = \pi_1$ and $L(\pi') = \sum_{k=1}^{n} d_{\pi'_k,\pi'_{k+1}}, \pi'_{n+1} = \pi'_1$. This acceptance criterion ensures that all reversed internal arcs, the two boundary arcs, and the closing arc are evaluated according to the asymmetric distance matrix.

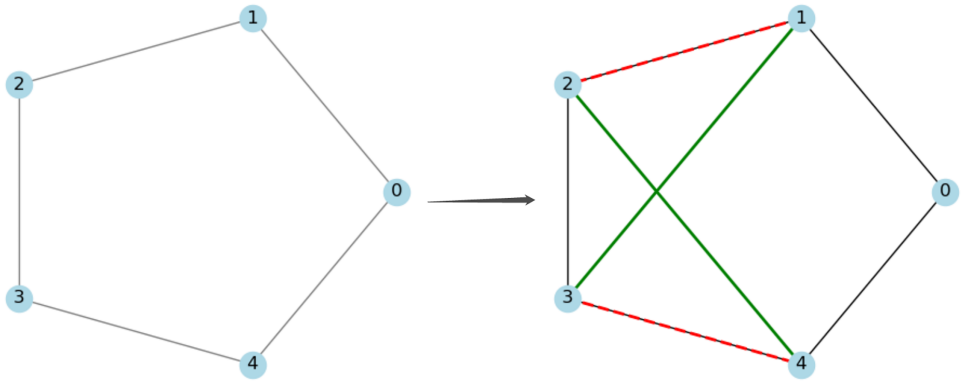


Figure 6: Illustration of the 2-opt operation.

4.3.2 Hamming Distance Association

Hamming distance is a metric used to measure the number of differing characters at corresponding positions between two strings of equal length. The core idea is to compare elements position by position and count the mismatches, thereby quantifying the degree of difference between them. In this study, the Hamming distance is used to define the dissimilarity between the current solution and the global best solution G_b:

$$H(\pi, G_b) = \sum_{k=1}^{n} I\left(\pi_k \neq (G_b)_k\right), \tag{24}$$

where $I(\cdot)$ denotes the indicator function. For each individual, the algorithm performs $H(\pi, G_b)$ iterations of the 2-opt operation. After each 2-opt attempt, the candidate route is retained only if its complete directed tour cost is lower than that of the current route.

5 Experimental Evaluation

5.1 Experimental Settings

To evaluate the performance of DCPO on the SATSP, 12 spherical test scenarios with different problem scales are designed. All instances are generated on a unit sphere and divided into low-dimensional scenarios (n = 25, 50, 75, 100), medium-dimensional scenarios (n = 200, 300, 400, 500), and large-dimensional scenarios (n = 600, 800, 1000, 1200), which are used to assess convergence behavior, computational stability, and scalability across different problem sizes.

All experiments are conducted on the same workstation equipped with an AMD Ryzen 7 5800H processor and 16 GB RAM, running the 64-bit Windows 10 operating system. All algorithms are implemented and executed in MATLAB R2019a under the same software environment. No GPU acceleration or parallel computing is used. DCPO is compared with six representative metaheuristic algorithms, namely the Discrete Mayfly Algorithm (DMA) [10], the Flower Pollination Algorithm (FPA) [9], the Discrete Sparrow Search Algorithm (DSSA) [29], the Randomized Bias Genetic Algorithm (RBGA) [30], the Genetic Algorithm (GA) [31] and the Discrete Marine Predators Algorithm (DMPA) [12]. These algorithms are selected

to represent different categories of metaheuristic methods. GA and RBGA represent evolutionary and biased genetic search, DMA, DSSA, and DMPA represent discrete population-based optimization, and FPA provides a problem-relevant benchmark because it has been applied to spherical TSP problems. For the iteration-based comparison, all algorithms use a population size of 50 and a maximum number of 2000 iterations. Each algorithm is independently executed 20 times using independent random initializations. Algorithm-specific parameters are set according to their original references or recommended settings. Since the exact optimal solutions of the generated SATSP instances are unavailable, the best value obtained across all algorithms and independent runs for each instance is used as the empirical best-known solution (*BKS*) for calculating the relative percentage deviation (*RPD*). The *RPD* is calculated as $RPD = (Mean - BKS)/BKS \times 100\%$, where *BKS* denotes the empirical best-known solution, and a lower *RPD* indicates better performance. For all test scenarios, the rank is determined by the mean objective value in ascending order. Table 2 displays the key control parameters for each algorithm.

Table 2: Parameter settings of the algorithms.

Algorithm	Parameter Settings
DCPO	Defense threshold T_f = 0.4 OX probability = 0.8 PMX probability = 0.2 Inversion mutation probability = 0.4 Swap mutation probability = 0.6
DMA	Defense threshold T_f = 0.4 Population size = 50, including 25 male and 25 female mayflies Parameter u linearly increases from 0.1 to 0.9
FPA	Switching probability p = 0.95
GA	Crossover probability p_c = 0.8
DSSA	Mutation probability p_m = 0.8 The proportion of producers = 0.4 The proportion of scouters = 0.2 Alarm value = 0.8
RBGA	Crossover rate = 0.7 Mutation rate = 0.01
DMPA	Step factor = 0.5

5.2 Performance Evaluation

5.2.1 Performance Evaluation in Low-Dimensional Scenarios

The proposed DCPO is compared with six representative algorithms in low-dimensional SATSP scenarios involving 25, 50, 75, and 100 cities.

As shown in Table 3, DCPO achieves the best overall performance across all low-dimensional scenarios. It achieves the lowest mean objective values for all four instances. DCPO also obtains the lowest RPD values across all low-dimensional scenarios, confirming that its average solutions remain closest to the empirical best-known solutions. It demonstrates strong stability, particularly in the 25-city case, where the standard

deviation is zero. For the 50- and 75-city instances, DCPO still maintains the lowest mean objective values with relatively low standard deviations, while FPA and DMA generally show competitive performance among the compared algorithms, although their rankings vary across different instance sizes. Although FPA achieves the lowest standard deviation in the 100-city case, DCPO remains superior in terms of both best-found and mean objective values. In contrast, DMPA performs the worst overall. As the problem size increases, DMPA shows larger mean objective values, while its standard deviations remain relatively high but do not increase monotonically.

Table 3: Statistical performance comparison of algorithms in low-dimensional SATSP scenarios.

Cities	Algorithm	Best	Worst	Mean	Std	RPD (%)	Rank
25	DCPO	**12.5989**	**12.5989**	**12.5989**	**0.0000**	**0.0000**	1
	DMA	**12.5989**	13.2956	12.8218	0.2336	1.7629	4
	FPA	**12.5989**	13.0893	12.7111	0.1876	0.8906	2
	GA	12.6755	14.7664	13.5644	0.6092	7.6634	5
	DMPA	21.7904	24.2605	23.6232	1.2933	87.5021	7
	DSSA	**12.5989**	13.1482	12.7873	0.1746	1.4954	3
	RBGA	12.8776	16.4015	14.6460	0.8919	16.2482	6
50	DCPO	**18.0247**	**18.7597**	**18.1711**	**0.1871**	**0.8122**	1
	DMA	18.5173	19.8939	18.8044	0.3009	4.3257	2
	FPA	19.1711	19.8893	19.3852	0.2140	7.5480	3
	GA	19.0702	23.1922	20.4483	1.1684	13.4460	4
	DMPA	51.4716	60.1774	55.3223	2.2943	206.9249	7
	DSSA	20.4860	22.2279	21.1622	0.4321	17.4067	5
	RBGA	28.1027	37.3011	32.4335	2.8058	79.9392	6
75	DCPO	**23.6937**	**24.4767**	**24.0953**	**0.2287**	**1.6950**	1
	DMA	25.1657	26.6556	25.8299	0.3639	9.0159	3
	FPA	25.0006	26.7825	25.4061	0.5446	7.2272	2
	GA	25.7010	29.0137	27.3647	0.8154	15.4936	4
	DMPA	89.4720	96.4809	92.4060	1.8635	290.0024	7
	DSSA	31.9520	33.6132	32.6952	0.4397	37.9911	5
	RBGA	48.6889	56.1305	52.6953	2.2673	122.4022	6
100	DCPO	**26.3608**	**27.7603**	**27.4029**	0.3629	**3.9532**	1
	DMA	28.0892	29.6108	28.8978	0.3700	9.6241	3
	FPA	27.7881	29.0767	28.3012	**0.2812**	7.3609	2
	GA	31.1702	35.6352	33.1808	1.2943	25.8717	4
	DMPA	122.0226	128.7148	125.4253	1.9979	375.8023	7
	DSSA	43.7101	46.7779	45.3659	0.9478	72.0961	5
	RBGA	65.9530	77.0832	72.3245	2.7888	174.3638	6

Note: Values in bold denote the best results among all compared algorithms.

As illustrated in Fig. 7, DCPO consistently exhibits the fastest convergence across different problem sizes. In the 50-city case (Fig. 7b), it rapidly reaches near-optimal solutions, significantly ahead of competing algorithms, while DMPA stagnates in the early stages. In the 75-city instance, DCPO converges to the

optimum within just 200 generations, again outperforming others. Similar patterns are observed in the 100-city case, where DCPO maintains strong convergence behavior.

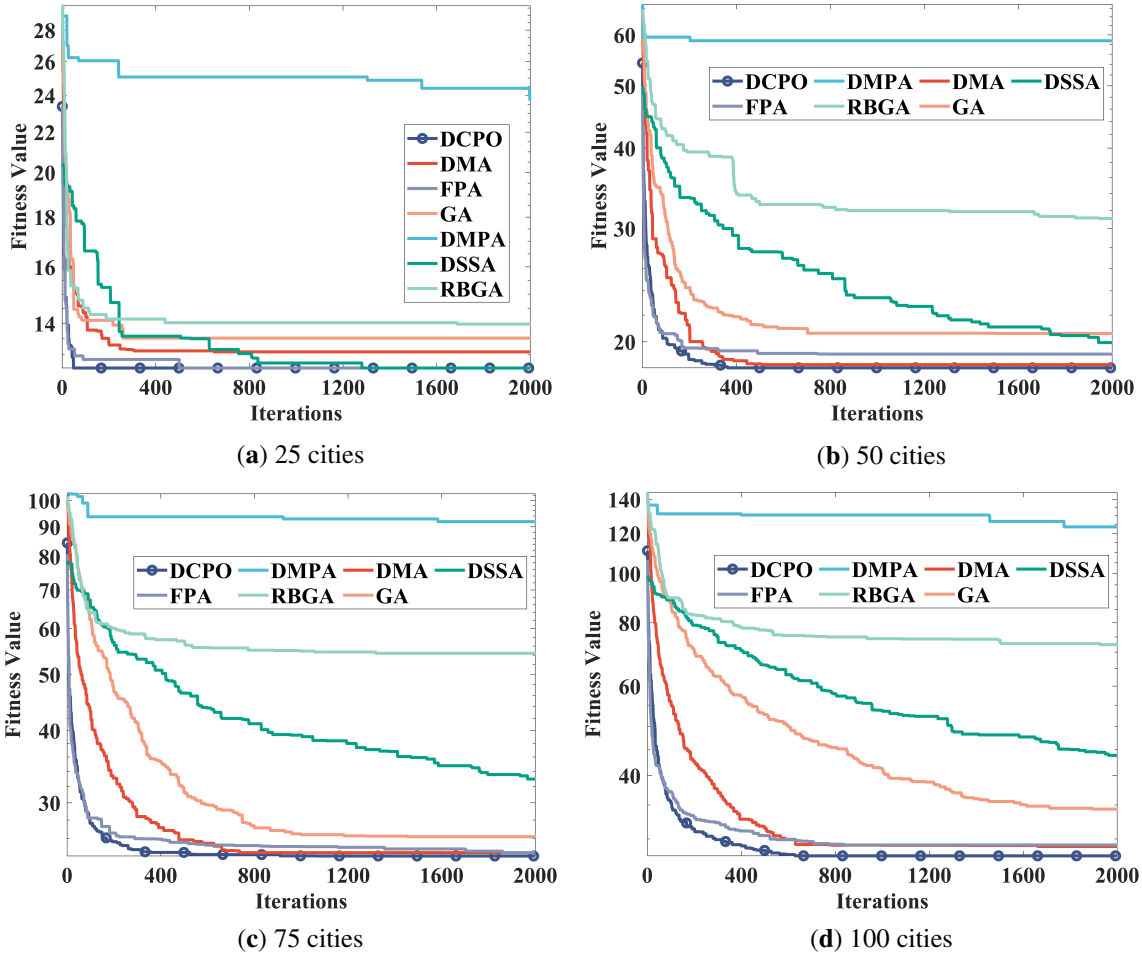


Figure 7: Convergence curves comparison in low-dimensional SATSP scenarios.

Fig. 8 compares the average runtime of the algorithms in the low-dimensional SATSP scenarios. The results confirm that the same population size and iteration limit do not lead to identical computational costs. DSSA consistently achieves the shortest runtime, whereas FPA and DMPA require substantially more computation across all instances. Although DCPO incorporates Hamming-distance-guided 2-opt local search, its runtime remains lower than those of FPA and DMPA and is comparable to that of DMA in the 50- and 100-city instances. However, DCPO is slower than simpler algorithms such as DSSA and RBGA, reflecting the additional cost introduced by local refinement. Therefore, the results indicate that DCPO achieves improved solution quality at a moderate additional computational cost, rather than under an identical computational budget.

As shown in Table 4, all pairwise comparisons between DCPO and the other algorithms yield p-values below 0.05 across the four low-dimensional scenarios, indicating statistically significant performance differences. Table 5 further shows that DCPO achieves the lowest Friedman mean rank of 1.00 and ranks first overall in the low-dimensional SATSP scenarios.

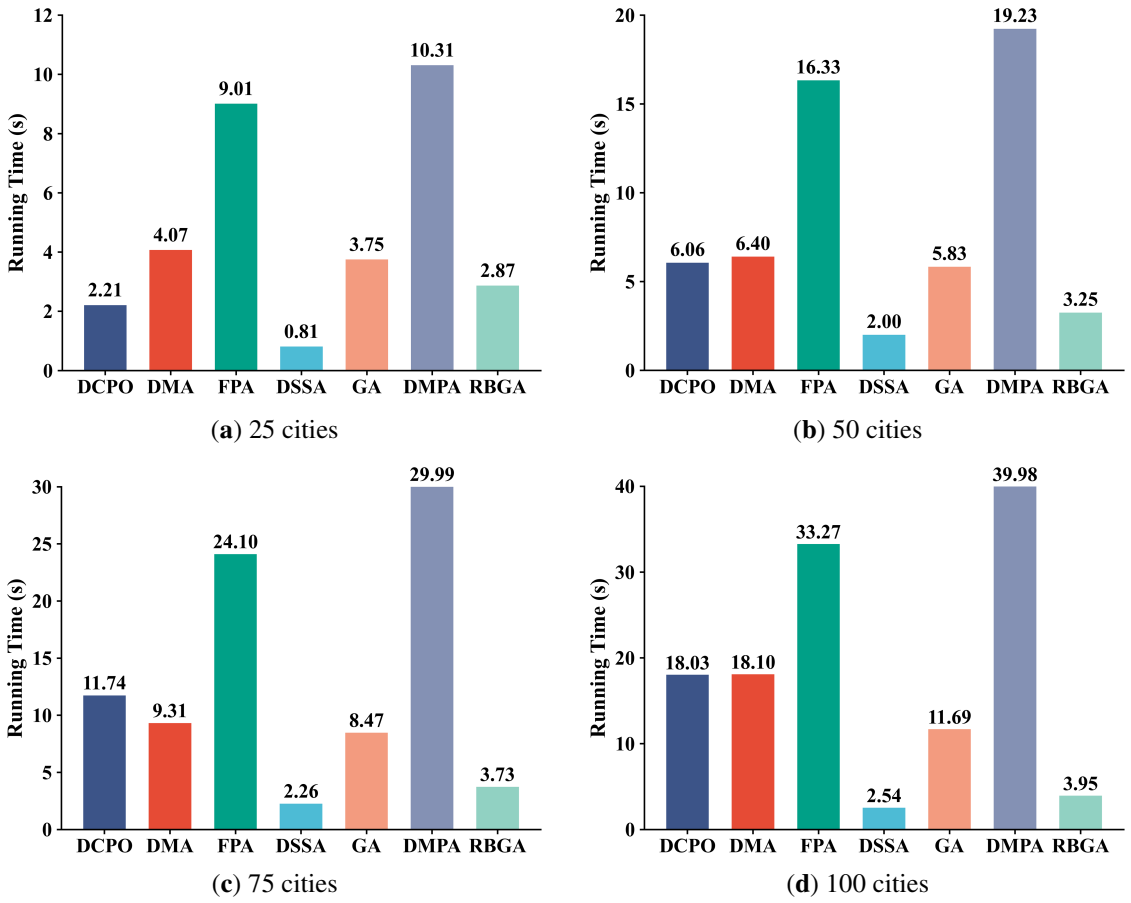


Figure 8: Average runtime comparison in low-dimensional SATSP scenarios.

Table 4: Wilcoxon rank-sum test results for low-dimensional SATSP scenarios.

Cities	DMA	FPA	GA	DMPA	DSSA	RBGA
25	1.6134e−04	9.5000e−03	8.0065e−09	7.8898e−09	6.6188e−05	8.0065e−09
50	7.4003e−06	6.4307e−08	6.5876e−08	6.5876e−08	6.5876e−08	6.5876e−08
75	6.7956e−08	6.7956e−08	6.7956e−08	6.7956e−08	6.7956e−08	6.7956e−08
100	1.0646e−07	2.6898e−06	6.7956e−08	6.7956e−08	6.7956e−08	6.7956e−08

Table 5: Friedman rank test results for low-dimensional SATSP scenarios.

Algorithms	DCPO	DMA	FPA	GA	DMPA	DSSA	RBGA
Friedman Mean Rank	1.00	3.00	2.25	4.25	7.00	4.50	6.00
Rank	1	3	2	4	7	5	6

5.2.2 Performance Evaluation in Medium-Dimensional Scenarios

To further evaluate the performance of DCPO, experiments are conducted in medium-dimensional SATSP scenarios involving 200, 300, 400, and 500 cities.

As reported in Table 6, DCPO achieves the best overall performance on the 200-city and 300-city instances, obtaining superior results in terms of best-found, worst, and mean objective values, as well as the standard deviation. It also achieves the lowest RPD values of 2.7728% and 1.7621%, respectively, indicating that its average solutions are closest to the empirical best-known solutions. However, the performance gap between DCPO and the second-best algorithm becomes smaller than that observed in low-dimensional scenarios. For the 400-city instance, DMA obtains the lowest best-found and mean objective values, and the lowest RPD of 2.2213%, compared with 2.7087% for DCPO, indicating a competitive advantage in this specific case. In the 500-city instance, although the standard deviation of DCPO is approximately 0.2 higher than that of DMA, DCPO again achieves lower best-found, worst, and mean objective values, together with the lowest RPD of 2.1061%, suggesting that its clear advantage in solution accuracy is maintained. In contrast, GA and RBGA exhibit relatively large standard deviations and RPD values, indicating weaker solution stability and accuracy.

Table 6: Statistical performance comparison of algorithms in medium-dimensional SATSP scenarios.

Cities	Algorithm	Best	Worst	Mean	Std	RPD (%)	Rank
200	DCPO	**37.94693**	**39.65352**	**38.99913**	**0.469898**	**2.7728**	1
	DMA	39.0108	40.97485	39.82286	0.522297	4.9436	2
	FPA	39.4735	41.99414	40.56126	0.658028	6.8894	3
	GA	85.53728	97.64482	91.76502	3.281636	141.8246	4
	DMPA	271.0386	278.7079	275.1647	1.9955	625.1303	7
	DSSA	103.3581	109.1478	106.8578	1.7055	191.5980	5
	RBGA	151.03889	172.9391	162.4814	6.3388	328.1806	6
300	DCPO	**47.84357**	**49.51733**	**48.68662**	**0.430969**	**1.7621**	1
	DMA	48.3741	49.97359	49.0335	0.433242	2.4871	2
	FPA	52.2058	54.454	53.3793	0.51708	11.5705	3
	GA	171.2452	192.2209	182.1271	5.835366	280.6721	5
	DMPA	412.6390	425.7831	421.6332	3.2214	781.2745	7
	DSSA	176.1572	187.2140	181.7783	3.0622	279.9430	4
	RBGA	240.3819	263.2672	252.38885	6.7678	427.5293	6
400	DCPO	56.64063	**58.37847**	57.64327	**0.399832**	2.7087	2
	DMA	**56.12308**	58.44733	**57.36975**	0.669549	**2.2213**	1
	FPA	63.98374	66.77027	65.53482	0.76081	16.7689	3
	GA	273.4615	291.3394	283.2349	4.839537	404.6674	5
	DMPA	562.2972	575.7477	570.4497	3.5543	916.4262	7
	DSSA	255.6438	269.0622	263.0065	3.9372	368.6245	4
	RBGA	334.3382	367.6260	353.1013	7.7292	529.1552	6
500	DCPO	**61.40847**	**64.26604**	**62.70179**	0.892082	**2.1061**	1
	DMA	63.47431	66.7778	65.18491	**0.678503**	6.1497	2
	FPA	75.0129	78.35645	77.19874	0.823387	25.7135	3
	GA	377.8307	409.8118	393.3767	8.301347	540.5903	5
	DMPA	711.7697	725.3358	720.4738	3.7864	1073.2483	7
	DSSA	341.3419	354.0146	347.5578	3.1284	465.9970	4
	RBGA	434.3002	467.3646	450.9775	8.3201	634.3897	6

Note: Values in bold denote the best results among all compared algorithms.

As shown in Fig. 9, DCPO exhibits consistently fast and stable convergence across all medium-dimensional instances. For the 200- and 300-city cases, both DCPO and FPA converge rapidly, whereas DCPO reaches lower objective values and therefore achieves better solution accuracy. In the 400-city case, DCPO and DMA show comparable convergence behavior, although DMA obtains a slightly better final solution. For the 500-city case, DCPO maintains a similar convergence rate while achieving a lower final objective value than DMA. Overall, these results demonstrate the robustness and scalability of DCPO across different problem sizes.

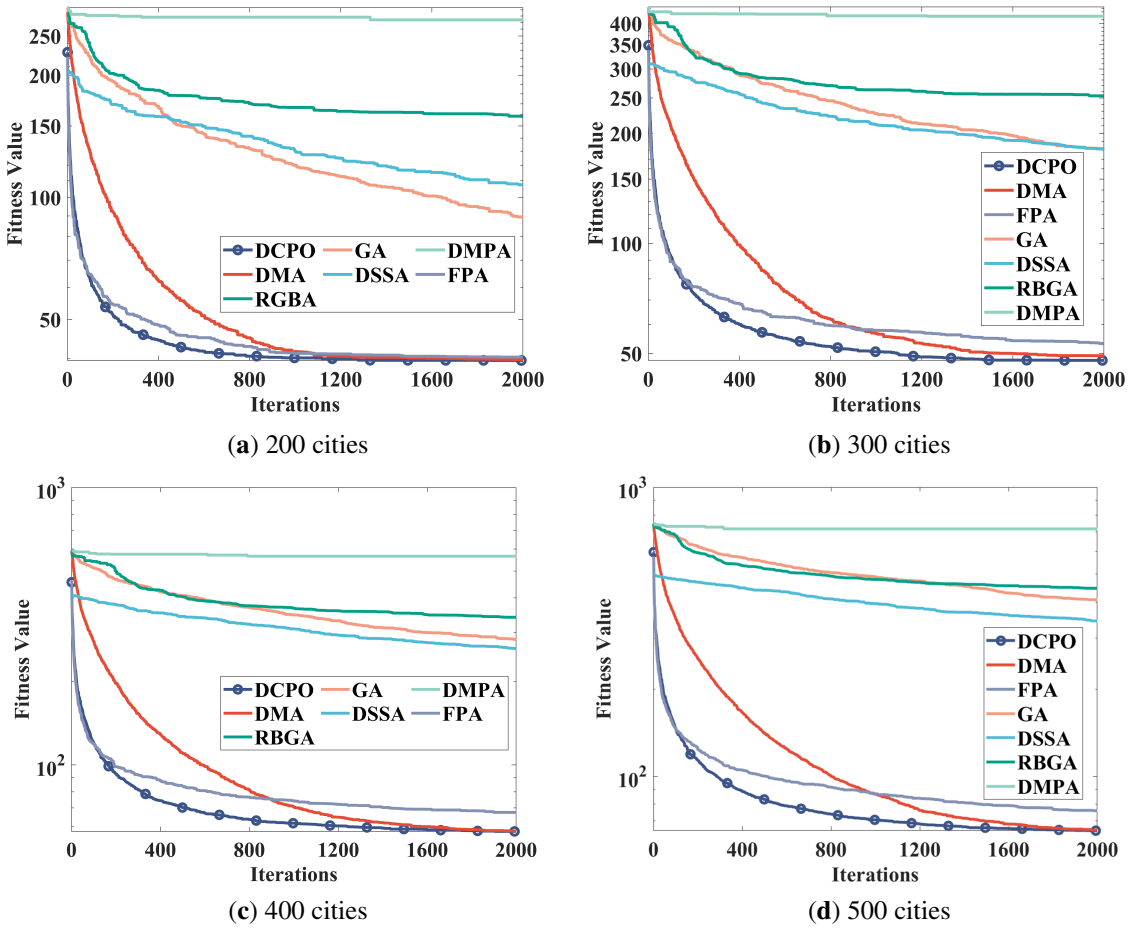
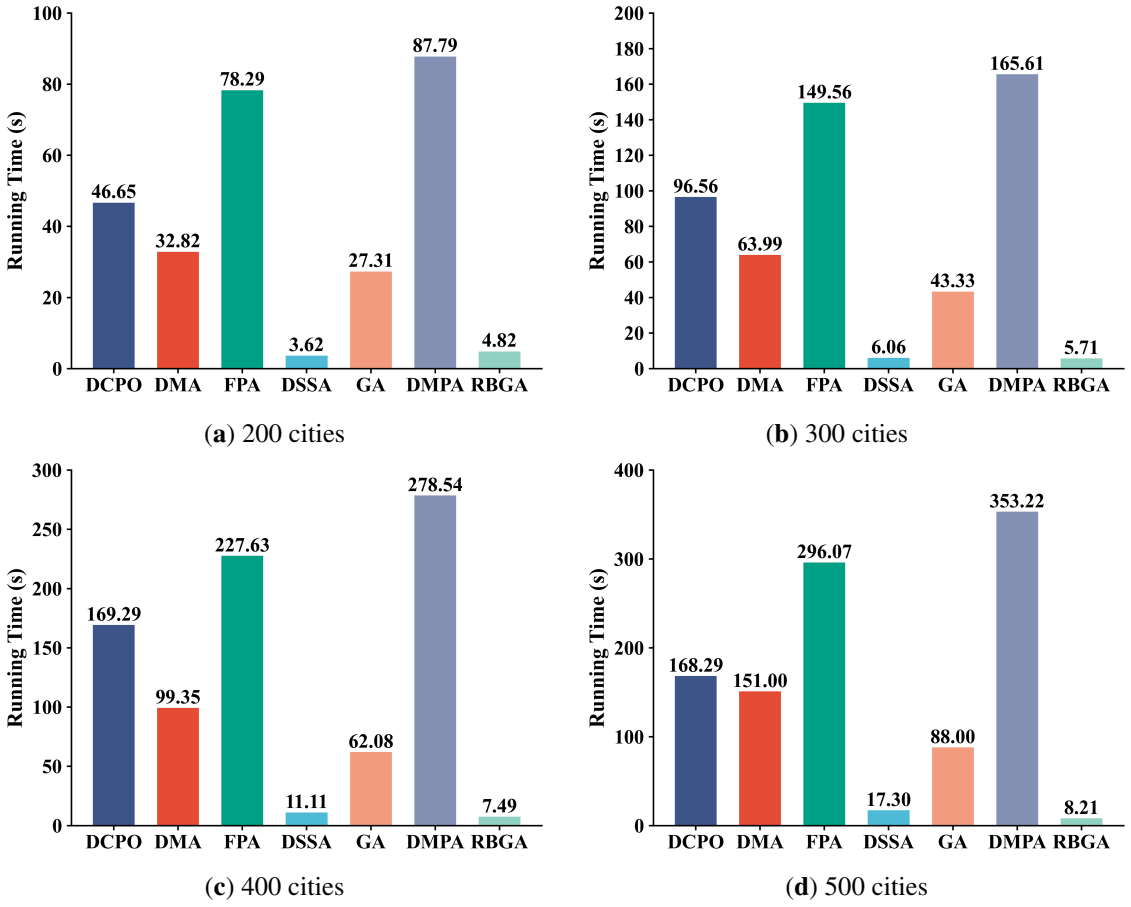


Figure 9: Convergence curves comparison in medium-dimensional SATSP scenarios.

Fig. 10 compares the average runtime of the algorithms on medium-dimensional SATSP scenarios. DSSA and RBGA consistently require the shortest runtimes, whereas DMPA and FPA incur the highest computational costs across all instances. Owing to its Hamming-distance-guided 2-opt local search, DCPO requires more runtime than DMA, GA, DSSA, and RBGA, but remains substantially faster than FPA and DMPA. Combined with the solution-quality results in Table 6, these findings indicate that DCPO achieves improved solution accuracy at a moderate additional computational cost, providing a reasonable trade-off between optimization performance and computational efficiency.

As shown in Table 7, all pairwise comparisons between DCPO and the other algorithms yield $p < 0.05$ indicating statistically significant performance differences. Table 8 further shows that DCPO achieves the

lowest Friedman mean rank of 1.25 and remains first overall, followed by DMA, FPA, DSSA, GA, RBGA, and DMPA.



(**a**) 200 cities (**b**) 300 cities

(**c**) 400 cities (**d**) 500 cities

Figure 10: Average runtime comparison in medium-dimensional SATSP scenarios.

Table 7: Wilcoxon rank-sum test results for medium-dimensional SATSP scenarios.

Cities	DMA	FPA	GA	DMPA	DSSA	RBGA
200	2.3025e−05	1.2346e−07	6.7956e−08	6.7956e−08	6.7956e−08	6.7956e−08
300	3.8500e−02	6.7956e−08	6.7956e−08	6.7956e−08	6.7956e−08	6.7956e−08
400	6.7956e−08	6.7956e−08	6.7956e−08	6.7956e−08	6.7956e−08	6.7956e−08
500	1.2346e−07	6.7956e−08	6.7956e−08	6.7956e−08	6.7956e−08	6.7956e−08

Table 8: Friedman rank test results for medium-dimensional SATSP scenarios.

Algorithms	DCPO	DMA	FPA	GA	DMPA	DSSA	RBGA
Friedman Mean Rank	1.25	1.75	3.00	4.75	7.00	4.25	6.00
Rank	1	2	3	5	7	4	6

5.2.3 Performance Evaluation in Large-Dimensional Scenarios

To further evaluate the scalability of DCPO, experiments are conducted in large-dimensional SATSP scenarios involving 600, 800, 1000, and 1200 cities. Since DMA and FPA show relatively competitive performance in the preceding experiments, this section focuses on a targeted comparison among DMA, FPA, and DCPO.

Table 9 summarizes the comparative results of the three algorithms across four large-dimensional scenarios. DCPO consistently achieves the lowest best-found, worst, and mean objective values, demonstrating its clear advantage in solution accuracy. It also obtains the lowest RPD values of 1.5526%, 1.8584%, 1.3005%, and 1.6811% for the 600-, 800-, 1000-, and 1200-city instances, respectively, indicating that its average solutions remain closest to the empirical best-known solutions. Although its standard deviation is slightly higher than that of FPA in the 800- and 1200-city instances, DCPO still maintains superior overall performance in terms of solution quality.

Table 9: Statistical performance comparison of algorithms in large-dimensional SATSP scenarios.

Cities	Algorithm	Best	Worst	Mean	Std	RPD (%)	Rank
600	DCPO	**71.23155**	**72.83894**	**72.33747**	**0.364589**	**1.5526**	1
	DMA	73.52249	78.22189	75.99045	1.260588	6.6809	2
	FPA	90.28232	94.41277	92.43644	0.969847	29.7690	3
800	DCPO	**84.87238**	**87.84861**	**86.449624**	**0.8782526**	**1.8584**	1
	DMA	92.41343	95.65977	94.04958	1.027286	10.8129	2
	FPA	113.3929	116.1305	114.5191	0.848491793	34.9309	3
1000	DCPO	**100.9383**	**103.6005**	**102.251**	**0.584602**	1.3005	1
	DMA	115.934	119.8339	117.6468	1.108796	16.5532	2
	FPA	132.2719	136.3085	134.0227	1.335686	32.7769	3
1200	DCPO	**114.132118**	**117.62145**	**116.05085**	1.002898939	**1.6811**	1
	DMA	137.485216	147.998429	144.093121	3.00816883	26.2512	2
	FPA	158.009876	159.853224	158.9830501	**0.595204637**	39.2974	3

Note: Values in bold denote the best results among all compared algorithms.

As shown in Fig. 11, DCPO exhibits the most favorable convergence behavior across the large-dimensional scenarios. Although FPA converges rapidly during the early iterations, it stabilizes at relatively higher objective values. DMA achieves final objective values closer to those of DCPO but requires more iterations to converge. In contrast, DCPO combines rapid convergence with higher solution accuracy.

Fig. 12 compares the average runtime of DCPO, DMA, and FPA on large-dimensional SATSP scenarios. The runtime of all three algorithms increases with the number of cities. DMA consistently achieves the shortest runtime, whereas FPA incurs the highest computational cost. DCPO remains between the two algorithms across all instances, reflecting the additional cost introduced by its local-refinement mechanism. Combined with the solution-quality results in Table 9, DCPO provides a reasonable trade-off between computational cost and solution accuracy in large-dimensional scenarios.

According to Table 10, all pairwise comparisons between DCPO and DMA or FPA yield $p < 0.05$ indicating statistically significant performance differences. Table 11 further shows that DCPO achieves the lowest Friedman mean rank of 1.00 and ranks first overall. Combined with the objective-value results

in Table 9, these findings confirm the superior overall performance of DCPO in the large-dimensional SATSP scenarios.

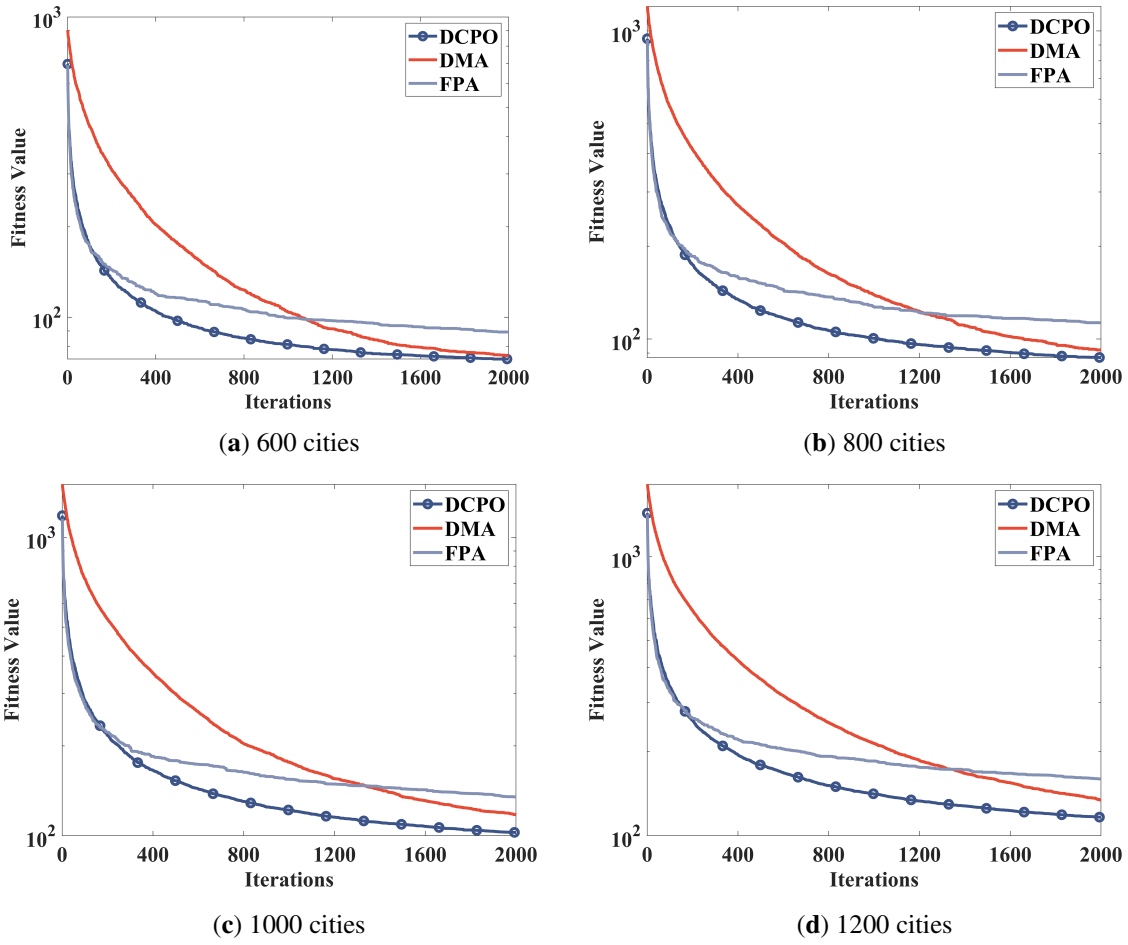


Figure 11: Convergence curves comparison in large-dimensional SATSP scenarios.

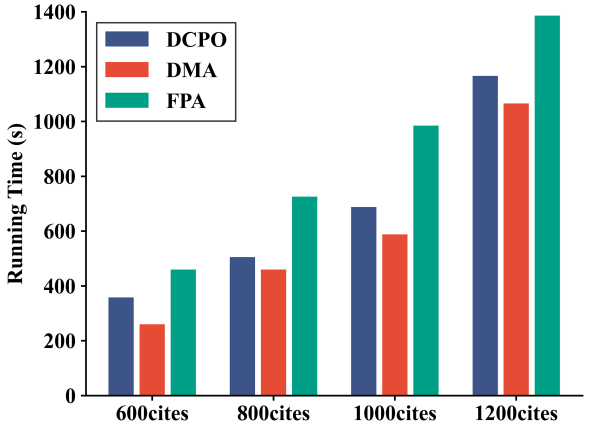


Figure 12: Average runtime comparison in large-dimensional SATSP scenarios.

Table 10: Wilcoxon rank-sum test results for large-dimensional SATSP scenarios.

Cities	DMA	FPA
600	6.7956e−08	6.7956e−08
800	6.7956e−08	6.7956e−08
1000	6.7956e−08	6.7956e−08
1200	6.7956e−08	6.7956e−08

Table 11: Friedman rank test results for large-dimensional SATSP scenarios.

Algorithms	DCPO	DMA	FPA
Friedman Mean Rank	1.00	2.00	3.00
Rank	1	2	3

5.3 Parameter Sensitivity Analysis

To evaluate the robustness of DCPO to its main parameter settings, one-factor-at-a-time sensitivity experiments were conducted on representative 25-, 200-, and 600-city SATSP scenarios. For each parameter, four candidate values were tested while all other parameters were fixed at their default settings. Each configuration was independently executed 10 times, and the best value, mean value, standard deviation, and average rank were reported. The rank is determined in ascending order according to the mean objective value for each scenario, and Avg. Rank denotes the average rank across the three scenarios. The best mean value and average rank are highlighted in bold.

Table 12 shows the sensitivity of DCPO to the defense-switching threshold. Among the tested values, T_f = 0.4 achieves the lowest mean objective value in all three scenarios and an average rank of 1.00. Although T_f = 0.6 obtains a slightly better best value in the 600-city scenario, T_f = 0.4 provides the best overall performance across the three scenarios. The small differences among neighboring values also indicate that DCPO is not highly sensitive to T_f; therefore, T_f = 0.4 is adopted as the default setting.

Table 12: Sensitivity of DCPO to the defense-switching threshold.

T_f	25-Cities Best	25-Cities Mean ± Std	200-Cities Best	200-Cities Mean ± Std	600-Cities Best	600-Cities Mean ± Std	Avg. Rank
0.2	12.5989	12.6247 ± 0.0774	37.1852	38.1227 ± 0.7504	72.2147	73.0287 ± 0.5652	2.67
0.4	**12.5989**	**12.5989 ± 0.0000**	**37.0899**	**38.0917 ± 0.7698**	71.4605	**72.6189 ±** 0.6875	**1.00**
0.6	12.5989	12.6268 ± 0.0836	37.2400	38.2798 ± 0.8685	**71.3268**	72.6558 ± 0.7499	2.67
0.8	12.5989	12.6758 ± 0.1609	37.6533	38.6555 ± 0.7310	71.7687	72.9139 ± 0.6207	3.67

Note: Bold values indicate the best result for each metric among the tested T_f settings.

Table 13 presents the sensitivity of DCPO to the crossover-selection probabilities. The configuration OX = 0.8 and PMX = 0.2 achieves the lowest mean objective values in the 25- and 600-city scenarios and obtains the best overall average rank of 1.33. Although the 0.6/0.4 configuration performs slightly better in the 200-city scenario, the differences between the two configurations is small. These results suggest that a higher probability of OX generally benefits DCPO, while retaining PMX with a probability of 0.2 provides complementary route recombination. Therefore, the 0.8/0.2 configuration is adopted because it provides the best overall performance across different problem scales.

Table 13: Sensitivity of DCPO to the crossover-selection probabilities.

OX	PMX	25-Cities Best	25-Cities Mean ± Std	200-Cities Best	200-Cities Mean ± Std	600-Cities Best	600-Cities Mean ± Std	Avg. Rank
0.2	0.8	12.5989	12.6636 ± 0.1287	37.9878	38.4487 ± 0.8631	72.2143	73.1837 ± 0.7031	4.00
0.4	0.6	12.5989	12.6316 ± 0.0762	37.8129	38.1913 ± 0.7925	71.4641	72.8229 ± 0.6522	3.00
0.6	0.4	12.5989	12.6033 ± 0.0336	**37.5276**	**38.0998 ± 0.7443**	**71.3246**	72.7658 ± 0.7229	1.67
0.8	**0.2**	**12.5989**	**12.5989 ± 0.0000**	37.6533	38.1435 ± 0.7668	71.4338	**72.3216 ± 0.6311**	**1.33**

Note: Bold values indicate the best results among the tested settings; bold parameter values denote the selected default setting.

5.4 Time Complexity of DCPO

We analyze the time complexity of DCPO. Let n denote the problem size (number of cities), N_{avg} the average population size, $T_{\max}$ the maximum number of iterations, and h the Hamming distance between the current solution and the global best solution. In the exploration phase, both the OX and PMX crossover operators require $O(n)$ time. In the exploitation phase, inversion mutation requires $O(n)$ time, whereas the swap operation itself requires $O(1)$ time. However, evaluating and greedily selecting the resulting candidate route requires recalculating the complete directed tour cost, so the overall complexity of the mutation step is $O(n)$. The local optimization phase performs h 2-opt attempts for each individual. Each attempt requires $O(n)$ time to generate and evaluate the complete directed candidate route; thus, the total complexity of this phase is $O(hn)$. Combining the complexities of these phases, the time complexity per individual per iteration is $O((h+1)n)$. The asymmetric distance matrix is precomputed once with a time complexity of $O(n^2)$ time. Consequently, using the average Hamming distance $\bar{h}$ over the search process, the practical time complexity of DCPO is $O\left(T_{\max} \cdot N_{\text{avg}} \cdot \left(\bar{h}+1\right) \cdot n\right) + O\left(n^2\right)$, where $\bar{h}$ denotes the average Hamming distance observed during the search. Since $0 \le h \le n$, the worst-case time complexity is $O\left(T_{\max} \cdot N_{\text{avg}} \cdot n^2\right) + O\left(n^2\right)$, which is dominated by $O\left(T_{\max} \cdot N_{\text{avg}} \cdot n^2\right)$.

6 Conclusions and Future Work

This study proposes a Discrete Crested Porcupine Optimizer (DCPO) for solving the Spherical Asymmetric Traveling Salesman Problem (SATSP). By introducing a permutation-based solution representation, crossover and mutation operators, and a 2-opt local search strategy, the proposed DCPO adapts the continuous search mechanism of the original CPO to discrete spherical asymmetric route optimization.

These mechanisms enhance population diversity, strengthen local exploitation, and improve the balance between global exploration and local refinement. Experimental results on 12 SATSP instances of different scales demonstrate that DCPO achieves superior solution quality and robustness compared with several representative metaheuristic algorithms, including DMA, FPA, GA, DMPA, DSSA, and RBGA. In particular, DCPO shows competitive performance in low- and medium-dimensional scenarios and maintains a clear advantage in large-dimensional scenarios, indicating its scalability for complex SATSP optimization. Although DCPO does not always achieve the shortest runtime, it provides a favorable trade-off between optimization accuracy, robustness, and computational efficiency.

Despite its effectiveness, DCPO still has room for improvement in terms of computational cost, especially when solving large-scale instances. Another limitation is that the relative contributions and complementary effects of its discrete search components have not yet been quantified through controlled ablation experiments. Future work will focus on improving the efficiency of DCPO through adaptive parameter control, more efficient local search strategies, and hybridization with other optimization algorithms. In addition, the proposed framework will be extended to real-world spherical routing and scheduling problems, such as global logistics routing, UAV or aircraft path planning, satellite routing, and mission scheduling under dynamic constraints.

Acknowledgement: Not applicable.

Funding Statement: This work was supported in part by the Joint Fund for Basic and Applied Basic Research in Guangdong Province under Grant No. 2024A1515110008 and in part by the Young Scientific and Technological Talent Cultivation Program of the Guangdong Association for Science and Technology under Grant No. SKXRC2026766.

Author Contributions: The authors confirm contribution to the paper as follows: conceptualization, Honglei Ma and Yingxuan Luo; methodology, Yingxuan Luo and Jia Chen; software, Yingxuan Luo and Jia Chen; validation, Yingxuan Luo and Jia Chen; formal analysis, Yingxuan Luo; investigation, Honglei Ma and Yingxuan Luo; data curation, Yingxuan Luo; writing—original draft preparation, Honglei Ma; writing—review and editing, Honglei Ma and Yingxuan Luo; visualization, Yingxuan Luo; supervision, Jie Li; project administration, Honglei Ma and Yingxuan Luo; funding acquisition, Jie Li. All authors reviewed and approved the final version of the manuscript.

Availability of Data and Materials: The data presented in this study are available on request from the corresponding author.

Ethics Approval: Not applicable.

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

References

1. Jünger M, Reinelt G, Rinaldi G. The traveling salesman problem. In: Handbooks in operations research and management science. Amsterdam, The Netherlands: Elsevier; 1995. p. 225–330.
2. Gomes DE, Iglésias MID, Proença AP, Lima TM, Gaspar PD. Applying a genetic algorithm to a m-TSP: case study of a decision support system for optimizing a beverage logistics vehicles routing problem. Electronics. 2021;10(18):2298. doi:10.3390/electronics10182298.
3. Zhang C, Wu Y, Ma Y, Song W, Le Z, Cao Z, et al. A review on learning to solve combinatorial optimisation problems in manufacturing. IET Collab Intell Manuf. 2023;5(1):e12072. doi:10.1049/cim2.12072.
4. Kritikos MN, Lappas PZ. Computational intelligence and combinatorial optimization problems in transportation science. In: Advances in core computer science-based technologies: papers in honor of professor Nikolaos Alexandris. Cham, Switzerland: Springer; 2020. p. 325–67.

5. Debnath D, Vanegas F, Boiteau S, Gonzalez F. An integrated geometric obstacle avoidance and genetic algorithm TSP model for UAV path planning. Drones. 2024;8(7):302. doi:10.3390/drones8070302.
6. He X, Pan QK, Gao L, Neufeld JS. An asymmetric traveling salesman problem based matheuristic algorithm for flowshop group scheduling problem. Eur J Oper Res. 2023;310(2):597–610. doi:10.1016/j.ejor.2023.03.038.
7. Di Pretoro A, Negny S, Montastruc L. Flexibility analysis in supply chain management: application to the traveling salesman problem. Comput Aided Chem Eng. 2021;50:1721–6. doi:10.1016/b978-0-323-88506-5.50267-9.
8. Eldem H, Ülker E. The application of ant colony optimization in the solution of 3D traveling salesman problem on a sphere. Eng Sci Technol Int J. 2017;20(4):1242–8. doi:10.1016/j.jestch.2017.08.005.
9. Zhou Y, Wang R, Zhao C, Luo Q, Metwally MA. Discrete greedy flower pollination algorithm for spherical traveling salesman problem. Neural Comput Appl. 2019;31(7):2155–70. doi:10.1007/s00521-017-3176-4.
10. Zhang T, Zhou Y, Zhou G, Deng W, Luo Q. Discrete Mayfly Algorithm for spherical asymmetric traveling salesman problem. Expert Syst Appl. 2023;221(1):119765. doi:10.1016/j.eswa.2023.119765.
11. Huang H, Wei Y, Zhou Y, Luo Q. Spherical vector-based artificial gorilla troops optimization for spherical asymmetric multiple traveling salesman problem. Evol Syst. 2024;15(3):965–99. doi:10.1007/s12530-023-09524-x.
12. Kumar M, Panwar K, Deep K. Discrete marine predators algorithm for symmetric travelling salesman problem. Evol Intell. 2024;17(5):3833–48. doi:10.1007/s12065-024-00960-5.
13. Osaba E, Yang XS, Diaz F, Lopez-Garcia P, Carballedo R. An improved discrete bat algorithm for symmetric and asymmetric traveling salesman problems. Eng Appl Artif Intell. 2016;48:59–71. doi:10.1016/j.engappai.2015.10.006.
14. Osaba E, Del Ser J, Sadollah A, Bilbao MN, Camacho D. A discrete water cycle algorithm for solving the symmetric and asymmetric traveling salesman problem. Appl Soft Comput. 2018;71:277–90. doi:10.1016/j.asoc.2018.06.047.
15. Ali IM, Essam D, Kasmarik K. A novel design of differential evolution for solving discrete traveling salesman problems. Swarm Evol Comput. 2020;52(6):100607. doi:10.1016/j.swevo.2019.100607.
16. Zhong Y, Wang L, Lin M, Zhang H. Discrete pigeon-inspired optimization algorithm with metropolis acceptance criterion for large-scale traveling salesman problem. Swarm Evol Comput. 2019;48(1):134–44. doi:10.1016/j.swevo.2019.04.002.
17. Nguyen QT. A new method for travelling salesman problem relied on growth optimization. Int J Intell Eng Syst. 2024;17(2):171–9. doi:10.22266/ijies2024.0430.15.
18. Xu Q, Xia K, Chu X. Discrete differentiated creative search for traveling salesman problem. Appl Soft Comput. 2025;174:112998. doi:10.1016/j.asoc.2025.112998.
19. Xu X, Shi X, Cao J, Huang W. Capacitated colored traveling salesman problem with time windows. IEEE Trans Autom Sci Eng. 2025;22:8057–68. doi:10.1109/TASE.2024.3476696.
20. Eremeev AV, Kovalenko YV. A memetic algorithm with optimal recombination for the asymmetric travelling salesman problem. Memet Comput. 2020;12(1):23–36. doi:10.1007/s12293-019-00291-4.
21. Rahman MA, Ma J. Two-stage probe-based search optimization algorithm for the traveling salesman problems. Mathematics. 2024;12(9):1340. doi:10.3390/math12091340.
22. Zhao S, Duan Q. Dynamic topology-aware linear attention network for efficient traveling salesman problem optimization. Mathematics. 2026;14(1):1–19. doi:10.3390/math14010166.
23. Bannach M, Acciarini G, Izzo D. The Keplerian traveling salesperson problem. arXiv:2605.00010. 2026.
24. Goswami K, Anekonda Veereshi G, Schmelcher P, Mukherjee R. Solving the travelling salesman problem using Bloch sphere encoding. Quantum Sci Technol. 2026;11(1):015007. doi:10.1088/2058-9565/ae1e9a.
25. Fathollahi-Fard AM, Ahmadi A, Sajadieh MS. An efficient modified red deer algorithm to solve a truck scheduling problem considering time windows and deadline for trucks' departure. In: Evolutionary computation in scheduling. Hoboken, NJ, USA: John Wiley & Sons, Inc.; 2020. p. 137–67.
26. Mojtahedi M, Fathollahi-Fard AM, Tavakkoli-Moghaddam R, Newton S. Sustainable vehicle routing problem for coordinated solid waste management. J Ind Inf Integr. 2021;23(1):100220. doi:10.1016/j.jii.2021.100220.
27. Sohrabi M, Fathollahi-Fard AM, Gromov VA, Dulebenets MA. A genetic engineering algorithm for the generalized quadratic assignment problem. Neural Comput Appl. 2025;37(18):12253–79. doi:10.1007/s00521-025-11155-z.
28. Abdel-Basset M, Mohamed R, Abouhawwash M. Crested Porcupine Optimizer: a new nature-inspired metaheuristic. Knowl Based Syst. 2024;284(1):111257. doi:10.1016/j.knosys.2023.111257.

29. Zhang Z, Han Y. Discrete sparrow search algorithm for symmetric traveling salesman problem. Appl Soft Comput. 2022;118:108469. doi:10.1016/j.asoc.2022.108469.
30. Gupta IK, Choubey A, Choubey S. Randomized bias genetic algorithm to solve traveling salesman problem. In: Proceedings of the 2017 8th International Conference on Computing, Communication and Networking Technologies (ICCCNT); 2017 Jul 3–5; Delhi, India. p. 1–6.
31. Potvin JY. Genetic algorithms for the traveling salesman problem. Ann Oper Res. 1996;63(3):337–70. doi:10.1007/BF02125403.