A Method for Solving TSP Based on Multi-scale GNN and Enhanced Ant Colony Optimization

Qiongdan Huang, Shilin Kang, Jiapeng Wang, Jiejing Han and Lulu Liu

Abstract—**In recent years, Graph Neural Networks (GNN) have shown substantial promise when combined with bio-inspired algorithms. However, current methods still struggle to solve large-scale instances of the Traveling Salesman Problems (TSP) due to limited feature extraction capabilities, inefficient utilization of global information, and suboptimal local search (LS) performance. To address these challenges, this paper proposes a Multi-Scale Temporal Fusion Ant Colony Optimization (MS-TFACO) algorithm, which integrates a Multi-Scale Temporal Feature Aggregation (MSTFA) module and a Dynamic Neural-guided Local Search (DNLS) strategy, seamlessly combining GNN with Ant Colony Optimization (ACO). By leveraging neural-learned heuristics, the proposed method reduces dependence on expert-designed features while improving solution quality and computational efficiency. The MSTFA module utilizes multi-scale convolutions and pooling operations to extract hierarchical representations and enhance interactions between local and global information. An attention mechanism is employed to further prioritize salient features, supporting more effective and holistic path planning. Building on this, the DNLS strategy dynamically predicts perturbation strengths to adaptively guide local search, enabling the algorithm to escape local optima and maintain a balance between exploration and exploitation. Experimental results demonstrate that MS-TFACO outperforms traditional ACO methods and achieves competitive performance compared to state-of-the-art Neural Combinatorial Optimization (NCO) approaches on standard TSP benchmarks.**

Index Terms—**Ant colony algorithm, Traveling salesman problems, Graph Neural Networks, Local search.**

I. Introduction

THE ACO [1] is a swarm intelligence optimization algorithm inspired by the foraging behavior of ants, initially proposed by Marco Dorigo in the 1990s. It simulates the pheromone-based communication mechanism used by ants to mark paths while searching for food, thereby enabling an iterative search for optimal solutions. During the execution of the algorithm, each "ant" deposits pheromones along its traversed path, with the pheromone concentration indicating the collective preference for that route. As iterations progress, pheromones accumulate on higher-quality paths, while those on inferior ones gradually evaporate. To enhance search efficiency, LS algorithms are often integrated with ACO. LS explores the neighborhood structure of the solution space by iteratively seeking improvements from an initial solution and terminates when no further enhancement can be achieved. By applying local optimization to the solutions generated by ACO, the convergence speed can be accelerated and the solution quality improved, thus mitigating ACO's tendency to become trapped in local optima when solving complex problems. This hybrid approach effectively guides the ant colony toward a globally optimal path and has been successfully applied to complex combinatorial optimization problems, such as the TSP.

One of the key challenges of ACO lies in the design of effective prior knowledge for its probabilistic model. Inadequate heuristic priors may lead to sluggish pheromone updates, increased computational overhead, and slow convergence. To overcome this limitation, Ye et al. proposed DeepACO [2], which integrates policy-gradient-based reinforcement learning into the ACO framework. By pretraining on specific problem instances through reinforcement learning, DeepACO enhances the efficiency of path selection. The reinforcement learning model guides the ant colony to concentrate on high-reward regions, thereby accelerating pheromone updates and significantly improving DeepACO's generalization capability to unseen problem instances. Nonetheless, DeepACO still encounters challenges when addressing larger-scale and more complex TSP, such as limited feature extraction capability, inadequate exploitation of global information, and suboptimal LS performance—all of which constrain its practical applicability.

To address these limitations, this paper proposes a neural-enhanced metaheuristic algorithm, named MS-TFACO, aimed at improving the heuristic efficiency of ACO. Within MS-TFACO, the MSTFA module and DNLS strategy effectively learn mappings from problem instances to heuristic information. The approach begins by training a neural model on multiple TSP instances to extract and encode task-relevant features. Subsequently, it refines the pheromone update mechanism during the optimization phase. Experimental results demonstrate that MS-TFACO achieves faster convergence and improved solution quality when tackling larger-scale and structurally more complex TSP.

The main contributions of this paper are as follows:

1) A novel MSTFA module is introduced within the MS-TFACO algorithm to enhance feature extraction via multi-scale convolution and to strengthen local–global information interaction through pooling operations. This design ensures stable adaptability across TSP of varying scales. Furthermore, the integration of an attention mechanism enables the model to adaptively weight features at different scales,

Manuscript received April 22, 2025; revised August 7, 2025.

This work was supported by the Key R&D Program Project of Shaanxi Province, China, Grant 2024NC-YBXM-217.

Qiongdan Huang is an associate professor at the School of Communication and Information Engineering, Xi'an University of Posts and Telecommunications, Xi'an, 710121, China. (e-mail: limitless010@163.com).

Shilin Kang is a postgraduate student at the School of Communication and Information Engineering, Xi'an University of Posts and Telecommunications, Xi'an, 710121, China (e-mail: 15686479558@163.com).

Jiapeng Wang is a postgraduate student at the School of Communication and Information Engineering, Xi'an University of Posts and Telecommunications, Xi'an, 710121, China (e-mail: 1075861687@qq.com).

Jiejing Han is a postgraduate student at the School of Communication and Information Engineering, Xi'an University of Posts and Telecommunications, Xi'an, 710121, China (e-mail: 18729432603@163.com).

Lulu Liu is a postgraduate student at the School of Communication and Information Engineering, Xi'an University of Posts and Telecommunications, Xi'an, 710121, China (e-mail: liululu0222@163.com).

thereby improving global information utilization and effectively mitigating the suboptimal decisions caused by excessive reliance on local information in traditional methods.

2) A DNLS strategy is proposed to predict perturbation intensity using a neural network and to dynamically adjust the LS policy. This facilitates escape from local optima and improves the performance of local search. Simultaneously, the strategy maintains a well-balanced trade-off between exploration and exploitation, significantly enhancing the solution quality for complex TSP instances.

II. Related Work

A. Ant Colony Optimization

ACO is a metaheuristic algorithm inspired by the foraging behavior of ants and is widely applied to optimal path-finding problems. Traditional ACO algorithms, such as the Ant Colony System [1], Max-Min Ant System [3], and Elitist Ant System [4], constitute the theoretical foundation for many state-of-the-art ACO variants. The effectiveness of these algorithms largely relies on carefully crafted heuristic designs. To alleviate the dependence on manually designed heuristics, Ye et al. [2] proposed a neural-guided ACO method, DeepACO, which replaces handcrafted heuristics with automatically acquired knowledge, thereby enabling adaptive optimization. However, DeepACO still suffers from slow convergence and suboptimal solution quality when addressing large-scale instances.

B. Neural Combinatorial Optimization in Travelling Salesman Problem

NCO is an interdisciplinary approach that employs deep learning techniques to tackle combinatorial optimization problems such as the TSP. NCO-based methods have demonstrated strong capabilities in solving TSP instances efficiently. In general, existing NCO approaches for TSP can be broadly categorized into two groups: end-to-end methods and neural-heuristic methods.

End-to-end methods in neural combinatorial optimization learn autoregressive solution construction or generate heatmaps to facilitate subsequent sampling-based decoding. Bresson et al. [5] applied the Transformer architecture to the TSP, utilizing reinforcement learning and beam search for decoding. Jin et al. [6] proposed a multi-pointer Transformer for deep reinforcement learning with controlled memory consumption, while Xin et al. [7] introduced a multi-decoder attention model to train multiple strategies and design a customized beam search, using the embedding layer to enrich information for solving the path planning problem. Innovative training techniques include those proposed by Jiang et al. [8], who enhanced TSP-solving performance by improving the model's adaptability to diverse data distributions through distributionally robust optimization. Similarly, Kim et al. [9] introduced the Sym-NCO model, which leverages the symmetry properties of combinatorial optimization problems to improve model efficiency and performance. Qiu et al. [10] developed the Dimes model, a meta-learning approach based on differentiable solvers capable of addressing a broader range of TSP instances. Xiao et al. [11] proposed NAR4TSP, which integrates reinforcement learning into the decoding process of a non-autoregressive (NAR) network, achieving superior performance in solution quality, generalization ability, and inference speed. Further advancements include the ASP method by Wang et al. [12], a general-purpose neural solver that continuously refines its model to tackle multiple optimization tasks. Cheng et al. [13] proposed a selection and optimization strategy that dynamically identifies key nodes and optimizes their sequence, thereby improving efficiency and scalability for large-scale TSP. Choo et al. [14] enhanced the practicality of neural combinatorial optimization by introducing simulated guided beam search (SimG-BS) to effectively screen candidate solutions. Pan et al. [15] developed a hierarchical TSP framework (H-TSP), which improves solution quality and efficiency through hierarchical decomposition and step-wise optimization. Sun and Yang [16] introduced Difusco, which constructs solutions through a graph diffusion process, enhancing the stability and generalization capability of TSP solutions. Wang et al. [17] proposed DEITSP, a diffusion model-based NAR solver that improves solution quality and inference efficiency via one-step denoising and a two-mode graph Transformer. Fang et al. [18] presented a novel mixed-fleet vehicle routing model under multiple distribution centers, integrating electric and fuel-powered vehicles to solve complex routing problems. Despite the substantial progress achieved by both end-to-end and neural-heuristic approaches in solving TSP, several critical challenges remain. Although end-to-end methods have improved solution quality and inference speed through techniques such as reinforcement learning and multi-decoder architectures, they often rely on single-scale modeling for feature extraction. This limits their ability to fully capture the intricate topological structures inherent in TSP instances. Moreover, these methods face scalability bottlenecks in terms of computational complexity and inference latency when dealing with large-scale TSP. Their limited capacity to exploit global information across varying data distributions and complex scenarios further constrains their generalization ability.

Neural-heuristic methods enhance performance by enabling neural networks to participate in decision-making within heuristic algorithms or to generate heatmaps that assist the heuristic process. For neural networks to contribute effectively, they must be tightly integrated within the algorithmic loop. Li et al. [19] proposed a delegated learning framework in which a neural network collaborates with a heuristic algorithm to make decisions in large-scale vehicle path planning problems. Ma et al. [20] designed an iterative solution approach using a dual collaborative Transformer architecture, which effectively addresses dynamic adjustment requirements in TSP. Wu et al. [21] introduced a learning-based method grounded in a large neighborhood search strategy for integer programming, thereby enhancing the search capability of the learning model. An improved heuristic learning strategy for solving TSP was also proposed in a separate study [22]. A recent advancement in this field is the ability to generate heatmaps in a single step to support downstream algorithms. Xin et al. [23] proposed NeurolKH, which integrates a deep learning model with the Lin-Kernighan-Helsgaun (LKH) heuristic algorithm to efficiently solve TSP. This method leverages the representational strength of neural networks along with the complementary advantages of classical heuristics to achieve high-quality

solutions. Building upon the LKH framework, Helsgaun et al. [24] extended its application to constrained TSP and vehicle routing problems, demonstrating strong performance in scenarios involving complex constraints. Fu et al. [25] showcased the generalization capability of a compact pre-trained model on large-scale TSP instances, thereby enhancing model versatility while maintaining high solving efficiency. Hudson et al. [26] proposed a novel approach that utilizes GNN to guide the LS strategy in TSP solving. Ye et al. [27] introduced Glop, a framework that addresses large-scale routing problems by combining global partitioning and local construction strategies. Further developments include the unified divide-and-conquer framework (UDC) proposed by Zheng et al. [28], specifically designed for large-scale combinatorial optimization. Wang et al. [29] explored efficient non-autoregressive solvers based on diffusion models. Xiao et al. [30] presented strategies that bridge global assessment with local selection, effectively solving TSP across varying scales. Neural-heuristic methods improve search efficiency by combining the decision-making capability of neural networks with traditional heuristic solvers. However, they remain limited by the intrinsic constraints of heuristic algorithms—particularly in their local search capabilities. In scenarios requiring complex constraint handling or dynamic adaptability, the reliance on fixed LS strategies often hampers the model's ability to escape local optima, ultimately impacting the overall solution quality.

C. MS-TFACO proposed motivation

The MS-TFACO algorithm is proposed to address the limitations of traditional ACO algorithms and NCO methods in solving large-scale TSP, particularly with regard to limited feature extraction capacity, insufficient utilization of global information, and inadequate LS capabilities. To overcome these challenges, MS-TFACO incorporates two innovative components: the MSTFA module and the DNLS strategy. The MSTFA module consists of two key designs: multi-scale feature extraction and temporal fusion attention. The multi-scale feature extraction component employs convolutional networks with varying receptive fields to capture hierarchical graph-structural features, while pooling operations are used to enhance the interaction between local and global information. This design enables the model to more comprehensively capture the structural characteristics of TSP graphs. The temporal fusion attention mechanism adaptively adjusts the weights of features across different scales through an attention-based fusion process, thereby improving the model's capability to represent overall tour structures and mitigating suboptimal decisions resulting from excessive reliance on local information. The DNLS strategy utilizes a neural network to predict perturbation intensity and adaptively adjust the scope of the LS procedure. This adaptive adjustment allows the search process to better escape local optima and maintain an effective balance between exploration and exploitation, ultimately improving search efficiency and solution quality in complex problem settings.

In comparative experiments against both ACO and NCO-based methods, MS-TFACO demonstrated significant performance advantages. Compared with traditional ACO algorithms, MS-TFACO achieved a 4.55% improvement in solution quality on the TSP1000 problem. In contrast, NCO methods typically suffer from limitations in feature extraction and LS performance, resulting in slower convergence and relatively lower solution quality. By integrating the MSTFA module and DNLS strategy, MS-TFACO significantly enhances both representational capacity and local search efficiency. Against DeepACO, a representative NCO method, MS-TFACO achieved a 0.46% improvement in solution quality on the TSP1000 problem. Overall, the experimental results confirm that MS-TFACO effectively overcomes the key limitations of traditional ACO and NCO approaches in terms of feature extraction, global information exploitation, and LS performance when solving large-scale TSP. Moreover, MS-TFACO demonstrates superior overall performance on large-scale TSP tasks, indicating its strong potential for practical application in complex combinatorial optimization scenarios.

III. Preliminary

A. Local Search

LS is a widely used optimization method for solving TSP. Its core idea is to start from an initial solution and iteratively explore its neighborhood to find a better one, continuing this process until no further improvement can be identified. LS typically yields high-quality approximate solutions within a reasonable time frame; however, it tends to get trapped in local optima. Among LS algorithms for TSP, one of the most widely adopted is 2-opt. The basic concept of 2-opt is to enhance a given tour by repeatedly removing two edges and reconnecting the resulting segments in a different way to reduce the total tour length. Specifically, for a given TSP route, 2-opt selects two points and reverses the order of the nodes between them. If the new route is shorter, the change is accepted, and the process continues until no further improvement is possible.

B. Travelling Salesman Problem

The TSP can be formulated as follows: each city corresponds to a variable X_i, and the set of all variables $X = \{X_1, X_2, \cdots, X_n\}$ defines the search space S. Each variable X_i is associated with a finite domain $D_i = \{v_1^i, \cdots, v_{|D_i|}^i\}(i = 1, \cdots, n)$, representing the cities that can be visited. The constraint set Ω enforces that each city must be visited exactly once and that the traveler must return to the starting city, thereby forming a complete Hamiltonian cycle. The objective function $f : S \to \mathbb{R}^+$ denotes the total length of the tour, and the optimization goal is to minimize $f(s)$ in order to obtain the shortest possible travel path.

C. Pheromone model and Solution construction

In ACO, the TSP is modeled using a pheromone framework. In this model, cities are represented as nodes, and paths between cities correspond to edges, forming a graph of decision variables. The path optimization problem of the TSP can be transformed into a pheromone-guided problem, where the pheromone concentration influences the path selection process.

During the solution construction phase, the pheromone intensity and heuristic information values associated with

the paths serve as guiding factors. Typically, ACO initializes pheromone values uniformly and updates these pheromone trails based on iterative results, while pheromone evaporation occurs at fixed time intervals.

Under the influence of ϕ_{ij} and ψ_{ij}, an ant constructs a solution by traversing the build graph $s = \{s_t\}_{t=1}^{n}$. If the ant is located at node i during the t th construction step ($s_{t-1} = i$) and has constructed a partial solution $s_{<t} = \{s_t\}_{t=1}^{t-1}$, then the probability of choosing node j as its next destination ($s_t = j$) is usually given by the following equation:

$$P(s_t = j|s_{<t}, \rho) = \begin{cases} \dfrac{\phi_{ij}^{\mu} \cdot \psi_{ij}^{\gamma}}{\sum\limits_{d_{ij} \in M(s_{<t})} \phi_{ij}^{\mu} \cdot \psi_{ij}^{\gamma}} & \text{if } d_{ij} \in M(s_{<t}) \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

where ρ denotes a TSP instance; $M(s_{<t})$ represents the set of selectable edges given the partial solution $s_{<t}$; μ and γ are control parameters; d_{ij} corresponds to the group component of the solution, representing the decision variable for transitioning from node i to node j; ϕ_{ij} indicates the pheromone intensity; and ψ_{ij} denotes the heuristic information value. For notational simplicity, the dependence of ϕ, ψ, d, and M on ρ is omitted. According to Eq. 1, constructing a complete solution requires m steps of graph traversal. The probability of generating a full solution can thus be factorized as:

$$P(S|\rho) = \prod_{t=1}^{m} P(s_t|s_{<t}, \rho) \tag{2}$$

where $P(S|\rho)$ represents the probability of constructing the solution S given the TSP instance ρ; $P(s_t|s_{<t}, \rho)$ denotes the probability that, at step t, the ant selects the next node s_t from the current path, with this choice depending on the previously constructed partial solution $s_{<t}$ and the TSP instance ρ; and $\prod_{t=1}^{m}$ denotes the product of the probabilities across all steps. Consequently, the probability of constructing the entire path is the concatenated multiplication of the probabilities associated with each step.

After the solution construction, the pheromone update process evaluates the solutions and adjusts the pheromone trails accordingly. By iteratively performing this process, ACO intelligently explores the solution space and eventually converges to an optimal or near-optimal solution.

IV. Methodology

This paper proposes an intelligent optimization framework, termed MS-TFACO, which integrates GNN and ACO to substantially improve both the efficiency and quality of solving complex TSP. MS-TFACO comprises a GNN backbone network, the MSTFA module, the ACO algorithm, and the DNLS strategy. The GNN backbone is based on the neural architecture introduced by Joshi et al. [31], utilizing anisotropic message passing and edge gating mechanisms to enhance the expressive capacity of the graph neural network. The MSTFA module incorporates a multi-scale feature extractor along with a temporal fusion attention mechanism, thereby improving the model's ability to extract discriminative features and exploit global information. The ACO component leverages the parameters learned by the neural network as heuristic information to guide the solution process. DNLS integrates LS with dynamic neural-guided perturbations, aiming to further enhance the effectiveness of LS.

The overall framework is depicted in Figure 1. Initially, the graph structure of the TSP instance is provided as input, and a K-nearest neighbors strategy [32] is applied to eliminate less promising solutions and accelerate the computational process. Subsequently, GNN utilize a message passing mechanism to extract features from the graph structure, capturing both local and global topological relationships between nodes and edges, thereby generating high-dimensional edge embeddings. These embeddings are then processed by the MSTFA module, which integrates a multi-scale feature extractor and a temporal fusion attention mechanism to refine and enhance the embeddings, resulting in more expressive fused edge representations. A Multilayer Perceptron (MLP) is then employed to perform nonlinear mapping on the fused embeddings, producing real-valued heuristic parameters. These parameters are subsequently converted into matrices by a heuristic converter, yielding heuristic weights that are directly used by the ACO algorithm. These weights guide the ant colony in selecting edges more effectively during the path construction phase, thereby enhancing the quality of the obtained solutions.

During the training phase, heuristic parameters are iteratively optimized by integrating reinforcement learning with the DNLS strategy. Reinforcement learning evaluates the quality of the current policy via a feedback mechanism and adjusts the heuristic parameters to improve the model's generalization capability. Concurrently, DNLS adaptively modifies the search scope through dynamic perturbations, enabling the algorithm to escape local optima and further enhance solution quality. Upon completion of training, the optimized heuristic parameters are fixed and subsequently employed as heuristic information to guide the ACO algorithm during the path construction process.

In the testing phase, ACO conducts path search on the input problem instance by combining pheromone intensities with the trained heuristic weights. As the search proceeds, pheromone is deposited and updated along the constructed paths, incrementally guiding the ant colony toward better solutions. Through multiple rounds of pheromone reinforcement, ACO efficiently iterates and explores the solution space, ultimately yielding high-quality TSP solutions. Furthermore, the DNLS strategy can optionally be incorporated during testing to further enhance solution quality, improving both robustness and convergence speed.

A. GNN module

The GNN presented in this paper is an anisotropic GNN architecture that employs an edge gating mechanism [33], which updates node and edge features at each layer through the message passing process inherent to GNN. Let x_i^l and e_{ij}^l denote the node and edge features corresponding to node i and edge ij at layer l, respectively. The features for the next layer are computed using an anisotropic message passing scheme, formulated as follows:

$$x_i^{l+1} = x_i^l + \alpha \cdot \text{BN}(U^l x_i^l + A_{j \in N_i}(\sigma(e_{ij}^l) \odot V^l x_j^l)) \tag{3}$$

$$e_{ij}^{l+1} = e_{ij}^l + \alpha \cdot \text{BN}(P^l e_{ij}^l + Q^l x_i^l + R^l x_j^l) \tag{4}$$

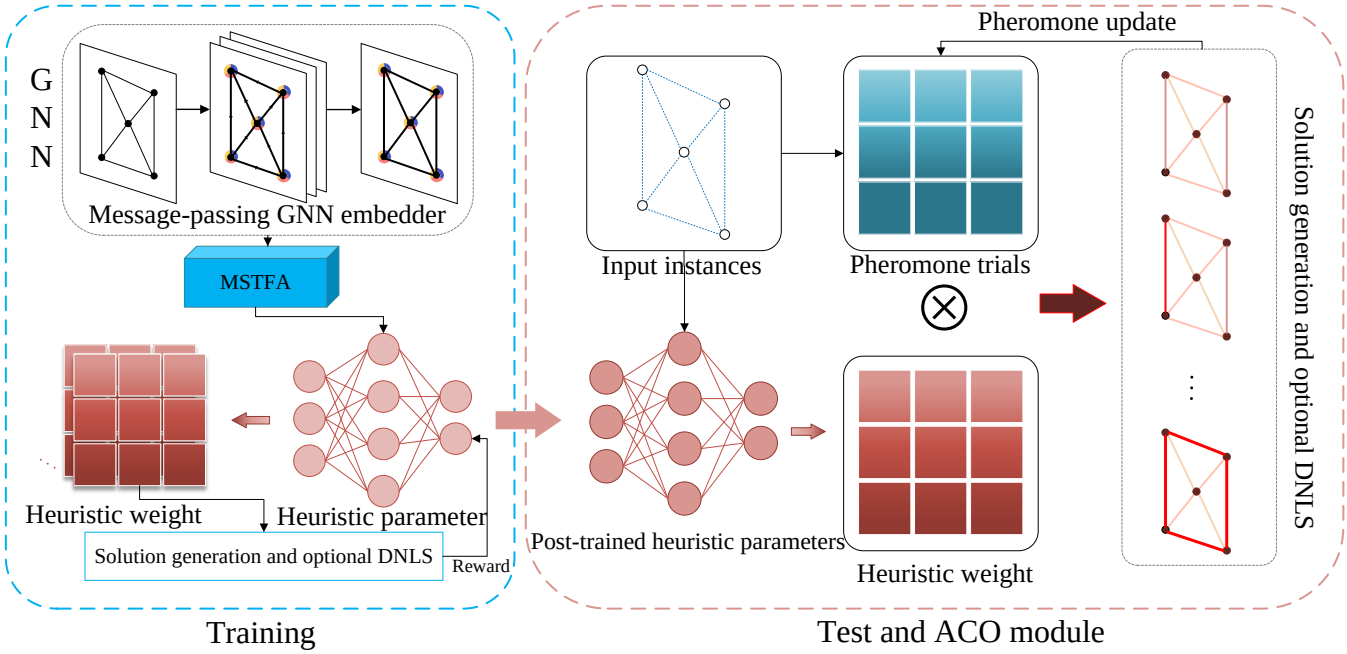

Fig. 1: Overall view of MS-TFACO

where U^l, V^l, P^l, Q^l, and R^l are the learnable parameters at layer l; α denotes the activation function (specifically, SiLU [34]); BN refers to the batch normalization operator; A represents the uniform pooling operation; σ is the sigmoid function; $\odot$ denotes the Hadamard product; and N_i represents the set of neighbors of node i. After L layers of updates, the final edge embedding is denoted as $E_{\text{final}} \in \mathbb{R}^{N_e \times d}$, where d is the dimensionality of the edge feature vector and N_e is the total number of edges in the TSP.

B. MSTFA module

Efficient feature extraction and temporal fusion are essential for effectively solving TSP. The MSTFA module is designed to capture deep structural information from TSP instances by integrating multi-scale feature extraction with an attention mechanism, thereby enhancing the global perception of both nodes and edges. The processing flow of this module is outlined as follows.

The final edge embedding, E_{final}, is obtained within the GNN module. Edge embeddings at two distinct time steps are selected and denoted as E_{final1} and E_{final2}. Subsequently, the feature matrices $E_{\text{conv1}} \in \mathbb{R}^{N_e \times d \times H \times W}$ and $E_{\text{conv2}} \in \mathbb{R}^{N_e \times d \times H \times W}$ are defined via convolution operations to extract multi-scale features. Here, H and W represent the spatial dimensions of the convolved features, which are computed as follows:

$$E_{\text{conv}i} = \text{ReLU}\Big(\text{Conv}_{3\times3}(E_{\text{final}i}) + \text{Conv}_{5\times5}(E_{\text{final}i}) + \text{Conv}_{7\times7}(E_{\text{final}i})\Big), \quad i \in \{1, 2\} \tag{5}$$

where $Conv_{k\times k}$ denotes a 2D convolution operation with a kernel size of $k \times k$. By employing 3×3, 5×5, and 7×7 convolution kernels, the model captures information from both local and broader receptive fields, thereby enhancing its capability to extract features across multiple scales. The ReLU function acts as a nonlinear activation function, introducing nonlinearity to improve the expressiveness of the resulting feature representations. The extracted features, E_{conv1} and E_{conv2}, encode multiscale information corresponding to distinct temporal stages. This design enables the model to effectively capture structural patterns at varying scales, which is essential for addressing the diverse path structures inherent in TSP.

Next, a channel attention mechanism is employed to further enhance the salient features. The corresponding weight C is computed as follows:

$$C = \text{Softmax}\Big(\text{Conv1D}\Big(\text{Cat}\Big(\text{AP}(E_{\text{conv1}}), \text{MP}(E_{\text{conv1}}), \text{AP}(E_{\text{conv2}}), \text{MP}(E_{\text{conv2}})\Big)\Big)\Big) \tag{6}$$

where cat denotes the concatenation operation; AP and MP represent average pooling and max pooling operations, respectively, which are employed to extract global information along the channel dimension. Conv1D refers to one-dimensional convolution, which is applied to capture inter-channel relationships and emphasize salient features. Softmax performs weight normalization to produce a meaningful distribution across channels. This mechanism enables the model to automatically focus on the most discriminative channel features while suppressing redundant information, thereby enhancing overall representational capability.

On this basis, a spatial attention mechanism is incorporated to enhance the model's ability to capture spatial information. The computation of the spatial attention weights S is defined as follows:

$$S = \text{Softmax}\Big(\text{Conv2D}\Big(\text{Cat}\Big(\text{AP}(E_{\text{conv1}}), \text{MP}(E_{\text{conv1}}), \text{AP}(E_{\text{conv2}}), \text{MP}(E_{\text{conv2}})\Big)\Big)\Big) \tag{7}$$

where average pooling and max pooling are employed to extract global information along the spatial dimension. Spatial weights are computed using a Conv2D operation to capture the relative importance of features across different spatial locations. The spatial attention mechanism enables the model

to focus more precisely on critical regions, such as high-weight edges or important nodes in the TSP graph, thereby enhancing its global path planning capability.

To effectively integrate multi-scale and attention information, we define the fusion weights F as:

$$F = C + S + 1 \tag{8}$$

the final fusion is characterized as:

$$E_{\text{fuse}} = F\left(E_{\text{conv1}} + E_{\text{conv2}}\right) \tag{9}$$

through the fusion weights, the model preserves fundamental information while simultaneously enhancing the effects of both channel and spatial features. The final fused features are obtained via weighted multiplication, allowing the importance of features across different spatial locations and channels to be dynamically adjusted. This approach adaptively allocates the contributions of various features, enabling the final representation to jointly incorporate multi-scale information, channel attention, and spatial attention. As a result, the model's capability for global information perception during the TSP solving process is significantly improved.

The overall architecture of the MSTFA module is illustrated in Figure 2. Following the aforementioned processing flow, the module enables deep feature learning for both nodes and edges in the TSP, and leverages multi-scale feature extraction in conjunction with temporal fusion attention mechanisms to enhance the global perception of node and edge representations. This, in turn, improves both the quality and efficiency of TSP solutions.

C. ACO module

The final fused feature E_{fuse} obtained from the MSTFA module is fed into an MLP with jump connections [35], where the extracted edge features are mapped to real-valued heuristic parameters θ. The prediction is computed according to the following formulation:

$$\theta = \sigma\left(W_{\text{MLP}} \cdot E_{\text{fuse}}\right) \tag{10}$$

where W_{MLP} denotes the learnable parameters of the MLP. In this study, we employ a three-layer MLP. The function σ represents the Sigmoid activation function, and θ denotes the output heuristic parameters. Using these parameters, we define a heuristic converter to construct a matrix representation. This converter maps the input TSP instances to their corresponding heuristic weights by leveraging the edge information of the graph, denoted as $\eta_\theta(\rho)$. The formulation is expressed as follows:

$$\eta_\theta(\rho) = \epsilon \cdot \mathbf{1}_{N\times N} + \sum_{k=0}^{E-1} \theta[k] \cdot E^{(i_k, j_k)} \tag{11}$$

where N and E denote the total numbers of nodes and edges in the TSP graph, respectively; $\theta[k]$ represents the heuristic value associated with the kth edge; (i_k, j_k) indicates the indices of the start and end nodes of the kth edge; $1_{N\times N}$ denotes an all-ones matrix; $E^{(i_k,j_k)}$ is a basis matrix with a value of 1 at position (i_k, j_k) and 0 elsewhere; and ε is a small constant added to ensure numerical stability. The resulting heuristic weights are employed to replace the manually defined heuristic information traditionally required in ACO algorithms.

$\eta_\theta(\rho)$ consists of non-negative real values $\eta_{ij;\theta}$ associated with each solution component d_{ij}, where $i \in \{1, \ldots, m\}$ and $j \in \{1, \ldots, |D_i|\}$. The MS-TFACO algorithm constructs solutions according to Eq. 2, with the introduction of $\eta_\theta(\rho)$ acting as a bias term to guide the solution construction process:

$$P_{\eta_\theta(\rho)}(S|\rho) = \prod_{t=1}^{m} P_{\eta_\theta(\rho)}(s_t \mid s_{<t}, \rho) \tag{12}$$

D. Dynamic Neural-guided Local Search

This paper extends the original NLS [2] algorithm by proposing a LS-based method, termed DNLS, which alternates between dynamic local search and neural-guided perturbations. The pseudocode of DNLS is provided in Table I. This enhancement is designed to improve the adaptability of perturbation steps, strengthen the algorithm's global search capability, and mitigate the risk of premature convergence to local optima.

TABLE I: DNLS

DNLS pseudo-code
1: Input: A solution s; an objective function f; well-learned heuristic measures $\eta_\theta(\rho)$; a local search operator LS; the initial number of perturbation moves $T_p^{(0)}$; decay parameters λ; and the maximum number of DNLS iterations T_{DNLS}; min perturbation $T_p^{(\min)}$; PS represents the perturbation strength.
2: Output: The best improved solution S^*.
3: $s = \text{LS}(s, f, +\infty)$
4: $s^* = \text{copy}(s)$
5: for $iter$ = 0 to T_{DNLS} do
6: $T_p = \max(T_p^{(\min)}, \text{round}(T_p^{(0)} \cdot e^{-\lambda \cdot iter}))$
7: $PS = \frac{1}{\eta_\theta(\rho)}$
8: $s = \text{LS}(s, \text{PS}, T_p)$
9: $s = \text{LS}(s, f, +\infty)$
10: if $f(s) < f(s^*)$ then
11: $s^* = s$
12: end for

In the original algorithm, the number of perturbation steps, T_p, is fixed. Such a static configuration can lead to either insufficient or excessive perturbations across varying problem scales or search phases, thereby constraining the algorithm's search efficiency. In the early stages of the search, larger perturbations promote broader exploration and help escape local optima, whereas smaller perturbations in the later stages conserve computational resources and refine solutions. To overcome these limitations, we introduce a minimum number of perturbation steps, $T_p^{(\min)}$, which can be adaptively tuned to prevent the total number of steps from becoming ineffective. Moreover, an attenuation parameter λ (set to 0.1 in this study) and a total number of iterations $iter$ (set to 10) are incorporated. These parameters dynamically adjust T_p, thereby enhancing the algorithm's adaptive search behavior. The adjustment formula is expressed as follows:

$$T_p = \max\left(T_p^{(\min)},\ \text{round}\left(T_p^{(0)} \cdot e^{-\lambda \cdot iter}\right)\right) \tag{13}$$

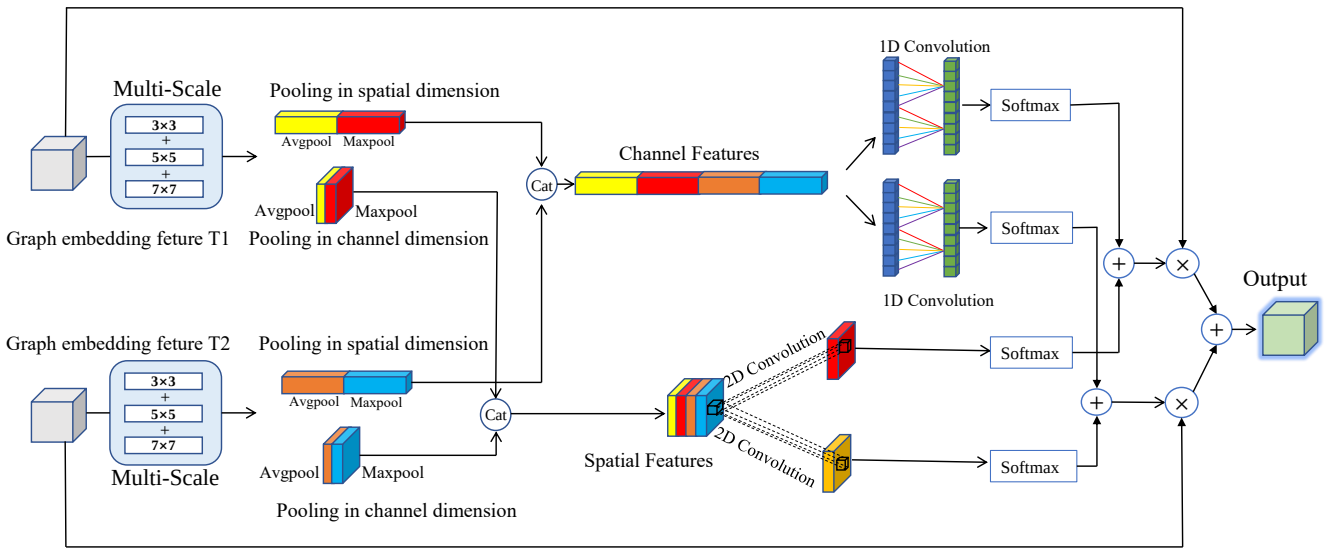

Fig. 2: MSTFA diagram

where $T_p^{(0)}$ denotes the initial number of perturbation steps, which is set to 20, and $\mathrm{round}(\cdot)$ denotes the rounding operation to the nearest integer.

The improved DNLS algorithm incorporates a dynamic perturbation strategy to better balance global search and LS. This strategy effectively improves search efficiency and enhances the algorithm's ability to escape local optima, thereby significantly boosting both solution quality and computational performance for complex TSP instances.

E. Training heuristic parameters

The heuristic learner is trained on TSP instances by learning the heuristic parameters θ, which map each instance ρ to its corresponding heuristic weights $\eta_\theta(\rho)$. The training objective is to minimize the discrepancy between the objective value of the constructed solution and that of the solution optimized by DNLS:

$$\begin{aligned}\text{minimize}\ \ L(\theta \mid \rho) =& \mathbb{E}_{s\sim P_{\eta_\theta(\rho)}(\cdot|\rho)}\left[f(s)\right. \\ &\left. + W f\left(\mathrm{DNLS}(s, f, +\infty)\right)\right]\end{aligned} \tag{14}$$

where W is a weighting coefficient used to balance the two loss terms; $f(\mathrm{DNLS}(s, f, +\infty))$ denotes the objective value of the solution optimized by the DNLS strategy; and $f(s)$ denotes the objective value of the current solution s. The first loss term directly guides the construction of an optimal solution. However, due to the inherent complexity of TSP, achieving this objective directly is often difficult. In contrast, the second loss term encourages the generation of solutions that are more amenable to further optimization by DNLS, which tends to produce high-quality results more reliably in an end-to-end framework. Nonetheless, relying solely on the second term may reduce training efficiency, as the quality of DNLS-optimized solutions typically exhibits limited variability. Therefore, combining both loss terms yields a more effective training signal and leads to improved overall performance.

Solution construction is based on the ACO framework. Equation 15 is derived from Equation 14, with the initial pheromone values set to 1 to ensure unbiased estimation. Furthermore, the REINFORCE algorithm is employed to dynamically update the learner parameters θ using a gradient estimator, as defined below:

$$\begin{aligned}\nabla L(\theta \mid \rho) =& \mathbb{E}_{s\sim P_{\eta_\theta(\rho)}(\cdot|\rho)}\big[\left(\left(f(s) - b(\rho)\right) + \right. \\ & \left. W\left(f\left(\mathrm{DNLS}(s, f, +\infty)\right) - b_{\mathrm{DNLS}}(\rho)\right)\right)\big] \\ & \cdot \nabla_\theta \log P_{\eta_\theta(\rho)}(s \mid \rho)\end{aligned} \tag{15}$$

where $b(\rho)$ denotes the average objective value of solutions generated by the current strategy, serving as a baseline to reduce the variance of the gradient estimate and improve the stability of parameter updates. Similarly, $b_{DNLS}(\rho)$ represents the average objective value of solutions obtained after DNLS optimization, acting as a benchmark to further reduce gradient variance. The term $\nabla_\theta \log P_{\eta_\theta(\rho)}(s|\rho)$ denotes the policy gradient, computed as the gradient of the strategy network $P_{\eta_\theta(\rho)}$ with respect to the parameter θ, and is used to update θ accordingly.

V. Experimentation

In this section, a comprehensive evaluation of the proposed MS-TFACO algorithm's performance in solving the TSP is conducted. The experiments include extensive comparisons with classical ACO algorithms, deep learning-based methods, and NCO approaches.

A. Experimental setup

1) Dataset and Example Generation: The experimental data were divided into two categories: synthetic and real data:

Synthesized data: TSP instances are generated uniformly and randomly in the region [0,1]². 1280 test instances are used for TSP20, TSP50 and TSP100. 128 test instances are used for TSP200, TSP500 and TSP1000.

Real data: 50 symmetric TSP instances with Euclidean distance properties were selected from the TSPLIB dataset. All instances are mapped to the [0,1]² region after normalization to ensure experimental consistency.

2) Computing platforms: Graphics: NVIDIA GeForce RTX 4090 (24GB video memory)

Processor: 16-core Intel Xeon(R) Platinum 8352V

Software environment: Python 3.8, using PyTorch 2.0.0 for model training and inference.

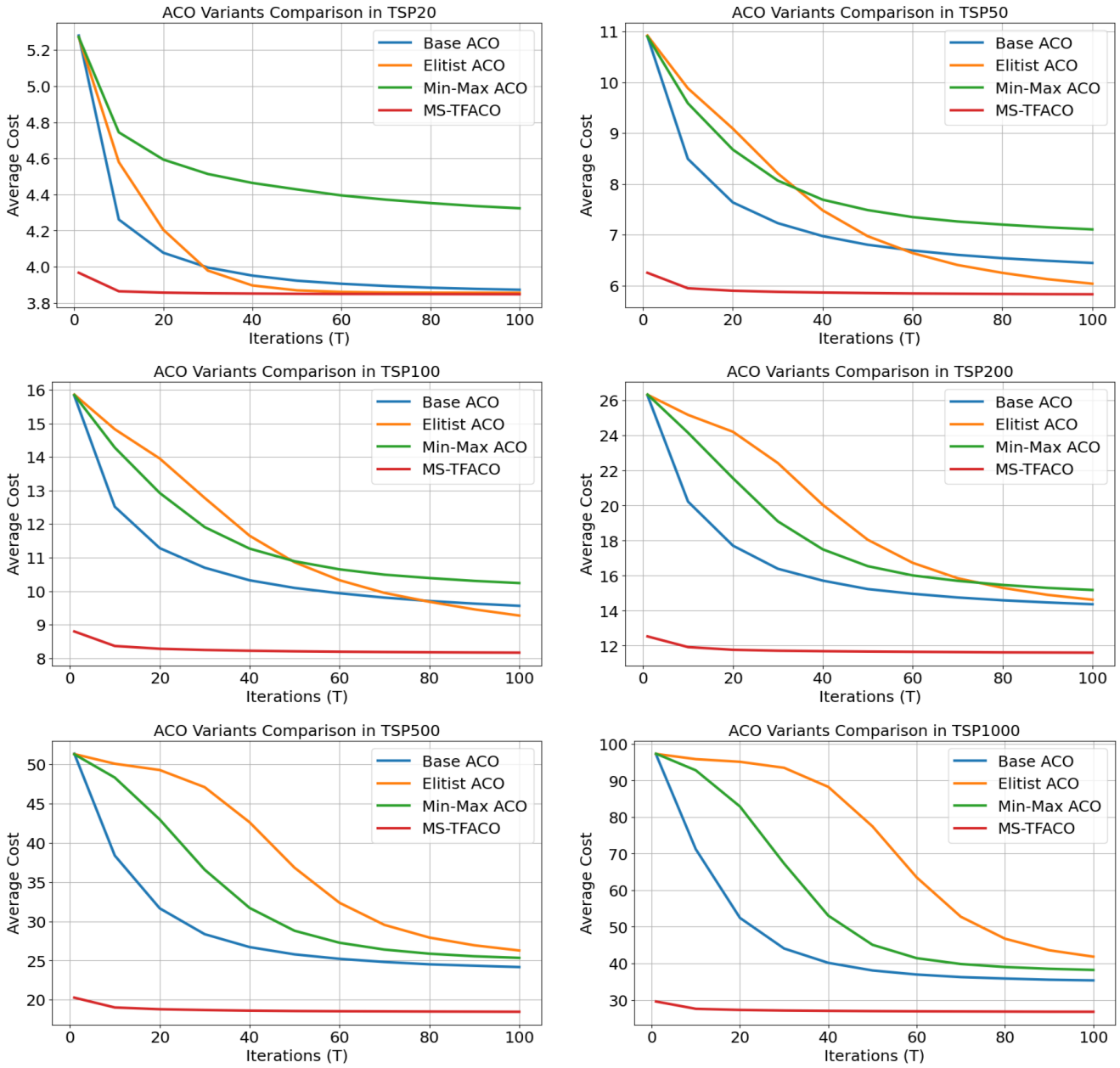

Fig. 3: MS-TFACO vs. Benchmark ACO

3) Key parameters: To improve computational efficiency and reduce the impact of suboptimal solutions, the graphs used in this experiment were sparsified by retaining only the k nearest neighbors for each node. The specific parameter settings are provided in Table II. In the experiments, the baseline heuristic is defined as the reciprocal of the edge length. The average path cost and average runtime represent the mean values obtained from three independent runs of the same model on the retained test set.

TABLE II: Graph sparsification parameter settings

A	TSP20	TSP50	TSP100	TSP200	TSP500	TSP1000
B	20	20	30	50	100	200
C	10	20	20	20	50	100

B. Algorithmic Performance Evaluation of MS-TFACO

In this section, experiments were conducted to comprehensively compare MS-TFACO without DNLS against both classical ACO methods and deep learning-based ACO approaches. The objective was to evaluate the effectiveness of the learned heuristic information.

1) Comparison with traditional ACO algorithms: In this section, three classical ACO algorithms were selected for comparison: Ant Colony System, Elite Ant System, and Max-Min Ant System. Performance comparisons between traditional ACO algorithms and MS-TFACO on TSP instances of varying scales were conducted. The experimental results are presented in Figure 3. As the number of iterations increases, the average path costs of MS-TFACO across all TSP scales remain significantly lower than those of traditional ACO algorithms and their variants. Furthermore, MS-TFACO demonstrates faster convergence. These results indicate that MS-TFACO can search for high-quality solu-

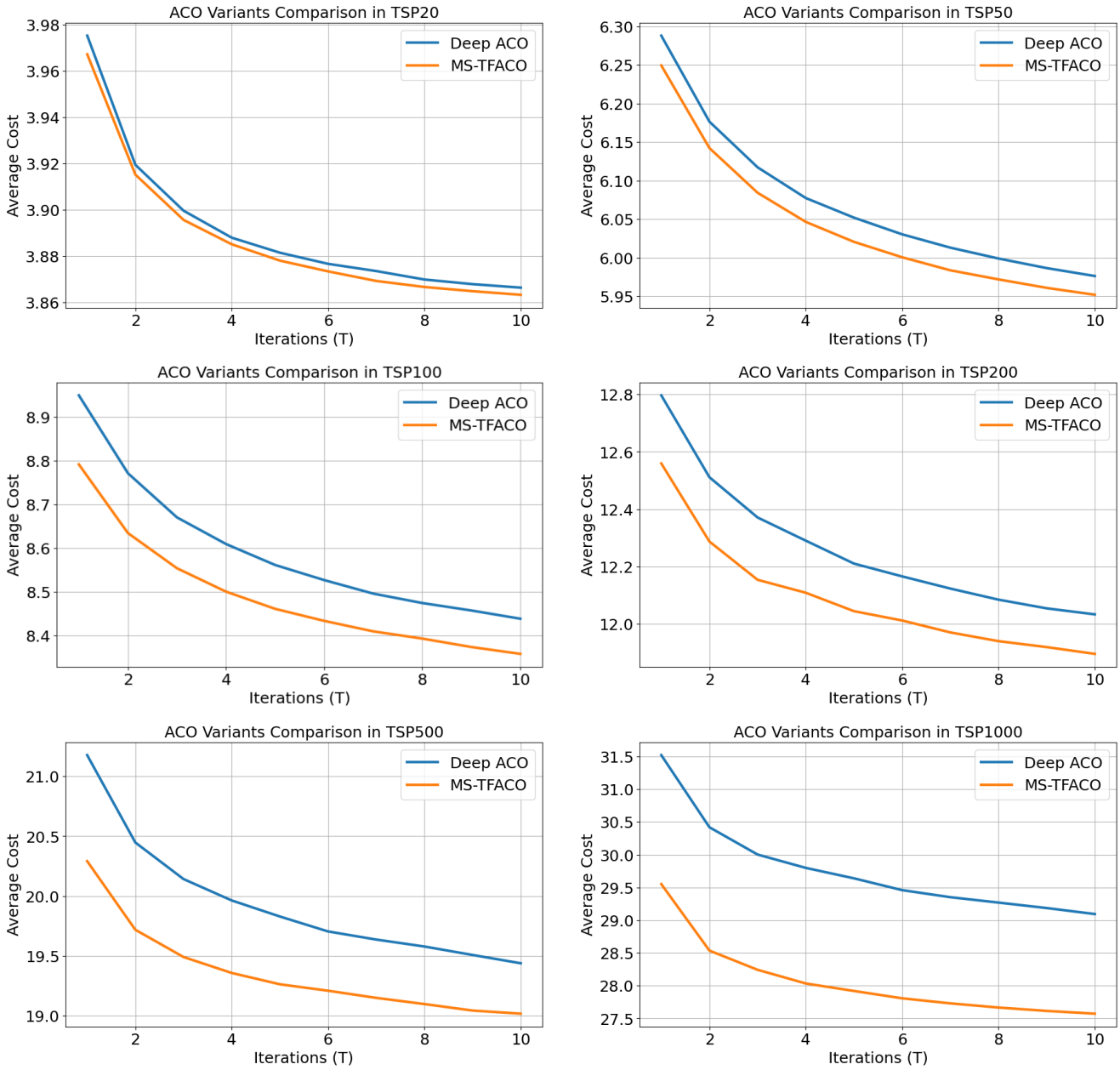

Fig. 4: MS-TFACO vs. DeepACO

tions more efficiently while maintaining strong adaptability on larger-scale TSP problems. Moreover, while traditional ACO relies on manually designed heuristic information for path construction, MS-TFACO employs MSTFA and GNN to learn more effective heuristics, enabling it to select better paths even at the early stages of the search.

2) Comparison with deep learning based ACO algorithms: In this section, MS-TFACO was compared with Deep-ACO, and the experimental results are shown in Figure 4. MS-TFACO achieves significantly lower average path costs than Deep-ACO across different TSP scales. Notably, for large-scale TSP instances, MS-TFACO converges faster and achieves lower final average path costs, demonstrating stronger optimization capability and adaptability. These results indicate that MS-TFACO is better suited for solving large-scale TSP problems and possesses higher practical value. Furthermore, owing to its superior neural network design, MS-TFACO learns more effective heuristic information, enabling it to select better paths even at the early stages of the search.

C. Algorithm performance evaluation of MS-TFACO combined with DNLS

In this section, comprehensive comparisons were conducted between MS-TFACO and existing NCO methods with the DNLS strategy enabled. The experiments evaluated their performance on both synthetic and real-world TSP datasets. Additionally, ablation studies were performed to assess the individual contributions of key components within the proposed framework.

1) Comparison with the NCO methodology: In this section, the performance of the proposed MS-TFACO framework is comprehensively evaluated on the TSP500 and TSP1000 benchmark datasets. The DNLS strategy is integrated into MS-TFACO and combined with the conventional

2-opt local search algorithm, which is well-established for improving TSP solutions by iteratively eliminating edge crossovers. Table III presents a detailed comparative analysis between MS-TFACO and several state-of-the-art NCO methods, as well as traditional heuristic algorithms. The results demonstrate that MS-TFACO consistently achieves superior solution quality and computational efficiency. Specifically, on the TSP500 dataset, MS-TFACO attains the lowest average objective value of 16.83, corresponding to a mere 1.69% optimality gap relative to the optimal solutions obtained by Concorde. This performance notably surpasses that of prominent learning-based baselines such as AM (31.42%), POMO (25.98%), and DIMES (5.80%), as well as recent GNN-based approaches like UDC (4.95%), DEITSP (3.12%), and GELD (3.47%), indicating enhanced generalization capability. Similar performance trends are observed on the larger-scale TSP1000 dataset, where MS-TFACO achieves an objective value of 23.71 with a minimal 2.51% gap, outperforming AM (49.37%), SO (11.41%), UDC (4.98%), DEITSP (3.14%), and GELD (3.02%). These results highlight the ability of MS-TFACO to generalize well to more complex and larger instances. Regarding computational efficiency, MS-TFACO exhibits competitive or superior runtime performance. For example, on TSP500, the method completes within 19 seconds, outperforming DeepACO (24s), ACO(LS) (39s), and several GNN-based methods including DIMES (25s), UDC (25s), DEITSP (34s), and GELD (32s). On TSP1000, MS-TFACO maintains scalability with a runtime of 115 seconds, which is faster than DeepACO (121s), DIMES (135s), UDC (133s), and GELD (84s), and significantly reduces computational time compared to POMO (987s) and AM (745s). Moreover, comparisons with traditional and hybrid ACO algorithms further highlight the effectiveness of MS-TFACO. Despite the use of LS and handcrafted features in ACO(LS) and GFACS, their solution quality remains inferior to MS-TFACO, underscoring the benefits of integrating dynamic neural LS with multi-scale feature aggregation. In summary, the proposed MS-TFACO framework achieves a compelling balance between solution accuracy and computational efficiency. Its superior gap minimization and reduced time complexity demonstrate its potential as a robust and scalable approach for solving large-scale combinatorial optimization problems in practical applications.

TABLE III: Comparative Results of MS-TFACO and NCO Methods

Method	TSP500			TSP1000		
	Obj.	Gap(%)	Time	Obj.	Gap(%)	Time
Concorde	16.55	-	-	23.13	-	-
LKH-3	16.55	-	-	23.13	-	-
AM[36]	21.75	31.42	187s	34.55	49.37	745s
POMO[37]	20.87	25.98	247s	33.90	46.56	987s
DIMES[10]	17.51	5.80	25s	24.75	7.00	135s
SO[13]	17.04	2.96	19s	25.77	11.41	98s
Pointerformer[6]	17.18	3.81	3s	25.10	8.51	15s
GFACS[38]	16.85	1.81	25s	23.89	3.29	135s
GLOP[27]	17.37	4.95	5s	25.17	8.82	17s
UDC[28]	16.94	2.40	20s	23.79	2.85	32s
DEITSP[29]	17.06	3.12	53s	24.14	4.36	226s
GELD[30]	17.37	4.95	36s	25.17	8.82	84s
ACO(LS)	17.46	5.50	39s	24.84	7.39	240s
DeepACO	16.85	1.81	24s	23.82	2.98	121s
Ours	**16.83**	1.69	19s	**23.71**	2.51	115s

2) Performance on the TSPLIB dataset: In this section, experiments are conducted in the TSPLIB dataset for different sizes of instances ($20 < n \leq 100$, $100 < n \leq 300$, $300 < n \leq 700$, $700 < n \leq 1500$, $1500 < n \leq 5000$), and the inference is performed using models trained on TSP50, TSP200, TSP500, TSP1000, and TSP2000, respectively. The experimental findings are presented in Table IV. MS-TFACO demonstrates superior performance in terms of maximum superiority gap across all instance sizes. In problems of very small size (20–100), MS-TFACO achieves an optimality gap of 0.87%, compared to 1.25% for ACO and 0.95% for DeepACO, showing excellent performance in small-scale problem generalization. In problems of small size (101–300), MS-TFACO achieves an optimality gap of 1.23%, compared to 1.70% for ACO and 1.30% for DeepACO, demonstrating good LS capability and generalization performance. As the problem size increases, on medium-sized (301–700) instances, the optimality gap of MS-TFACO is 2.65%, which is lower than that of ACO's 4.25% and DeepACO's 2.78%. This finding indicates that MS-TFACO maintains robust optimization capability in more complex scenarios. In problems of a large scale (701–1500), despite the substantial increase in problem complexity, MS-TFACO exhibits an optimality gap of 3.85%, which is more favorable than ACO's 7.15% and DeepACO's 4.06%. Furthermore, for very large-scale instances (1501–5000), MS-TFACO continues to outperform with an optimality gap of 5.56%, significantly lower than ACO's 12.69% and DeepACO's 6.16%. This further substantiates the efficacy, scalability, and convergence capacity of MS-TFACO in solving large-scale TSP instances within the TSPLIB dataset. $\#$ denotes the number of instances in each set.

TABLE IV: Results of the Experimental Comparison of MS-TFACO with ACO and DeepACO on the TSPLIB Dataset

TSPLIB	#	ACO	DeepACO	MS-TFACO
20-100	30	1.25%	0.95%	0.87%
101-300	30	1.70%	1.30%	1.23%
301-700	10	4.25%	2.78%	2.65%
701-1500	10	7.15%	4.06%	3.85%
1501-5000	10	12.69%	6.16%	5.56%

3) Ablation experiments with MS-TFACO: This section presents ablation experiments on the MSTFA module and DNLS strategy of MS-TFACO to analyze their contributions to the overall performance. The experimental results are shown in Table V. After removing the DNLS and MSTFA modules respectively, the solution quality significantly deteriorated, clearly demonstrating the critical roles of these modules in enhancing the final results. Specifically, upon removal of the DNLS module, the average path costs for TSP500 and TSP1000 increased from 16.83 to 18.47 and from 23.71 to 26.76, respectively, indicating that DNLS plays a key role in providing local optimization capabilities; its absence reduces solution refinement. Furthermore, after removing the MSTFA module, the values further increased to 16.93 and

24.89 for TSP500 and TSP1000, respectively, suggesting that this module significantly contributes to feature extraction and the utilization of global information.

TABLE V: Results of MS-TFACO Ablation Experiments

Method	TSP500	TSP1000
MS-TFACO	16.83	23.71
-DNLS	18.47	26.76
-MSTFA	16.93	24.89

To further elucidate the optimization effectiveness of the dynamic perturbation strategy in DNLS, this section also presents a comparative analysis of the runtime performance between the MS-TFACO algorithm and its variant without the dynamic perturbation strategy, reporting the average execution time for each instance. The experimental results, shown in Table VI, indicate a reduction in runtime of 7 seconds and 16 seconds for TSP500 and TSP1000, respectively. This improvement is mainly attributed to the dynamic perturbation mechanism, which adaptively adjusts the perturbation magnitude during iterations, effectively avoiding redundancy while ensuring global search capability.

TABLE VI: Experimental results of MS-TFACO ablation with dynamic perturbation removal strategy

Method	TSP500	TSP1000
MS-TFACO	19s	115s
-dynamic T_p	26s	131s

VI. Conclusion

In this study, we propose an intelligent optimization framework, MS-TFACO, which integrates GNN, the MSTFA module, the DNLS strategy, and the ACO algorithm. This framework is designed to overcome the limitations of conventional ACO and NCO methods in solving large-scale TSP, particularly addressing challenges related to insufficient feature extraction, limited utilization of global information, and suboptimal local search performance. The MSTFA module employs multi-scale feature extraction via convolutional networks to capture hierarchical graph structural information, thereby enhancing the model's ability to aggregate global context through pooling operations. Additionally, a temporal fusion attention mechanism adaptively weighs features across different scales, further improving the exploitation of global information. The DNLS strategy leverages a neural network to predict perturbation intensity and dynamically modulate the local search scope, enabling the algorithm to effectively escape local optima by adjusting both perturbation strength and local search range.

Experimental comparisons with traditional ACO and DeepACO methods demonstrate that, even without the DNLS strategy, MS-TFACO's incorporation of multi-scale feature extraction and temporal fusion attention facilitates more effective heuristic information learning, resulting in substantial performance improvements over these baselines. Notably, MS-TFACO exhibits enhanced adaptability and computational efficiency, especially when solving complex TSP instances. Furthermore, compared to baseline NCO approaches, the integration of the DNLS strategy significantly enhances the search's ability to avoid local optima, yielding superior solution quality and competitive performance relative to state-of-the-art NCO methods. Experiments on the TSPLIB dataset further confirm that MS-TFACO achieves smaller optimality gaps than both ACO and DeepACO. Ablation studies validate the critical contributions of the MSTFA module and DNLS strategy: omission of the MSTFA module leads to increased average path costs, underscoring its importance in feature extraction and temporal information fusion, while removal of the DNLS strategy substantially degrades local search optimization capability. These results collectively substantiate the efficacy and superiority of MS-TFACO for large-scale TSP optimization.

References

[1] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: optimization by a colony of cooperating agents," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 26, no. 1, pp. 29-41, 1996.

[2] H. Ye, J. Wang, Z. Cao, et al., "DeepACO: neural-enhanced ant systems for combinatorial optimization," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[3] T. Stützle and H. H. Hoos, "MAX–MIN ant system," *Future Generation Computer Systems*, vol. 16, no. 8, pp. 889-914, 2000.

[4] 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.

[5] X. Bresson and T. Laurent, "The Transformer Network for the Traveling Salesman Problem," arXiv, 2021.

[6] Y. Jin, Y. Ding, X. Pan, et al., "Pointerformer: Deep reinforced multi-pointer transformer for the traveling salesman problem," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, 2023, pp. 8132-8140.

[7] L. Xin, W. Song, Z. Cao, et al., "Neurolkh: Combining deep learning model with Lin-Kernighan-Helsgaun heuristic for solving the traveling salesman problem," *Advances in Neural Information Processing Systems*, vol. 34, pp. 7472-7483, 2021.

[8] Y. Jiang, Y. Wu, Z. Cao, et al., "Learning to solve routing problems via distributionally robust optimization," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, 2022, pp. 9786-9794.

[9] M. Kim, J. Park, and J. Park, "Sym-nco: Leveraging symmetricity for neural combinatorial optimization," *Advances in Neural Information Processing Systems*, vol. 35, pp. 1936-1949, 2022.

[10] R. Qiu, Z. Sun, and Y. Yang, "DIMES: A Differentiable Meta Solver for Combinatorial Optimization Problems," [Online].

[11] Y. Xiao, D. Wang, B. Li, et al., "Reinforcement learning-based nonautoregressive solver for traveling salesman problems," *IEEE Transactions on Neural Networks and Learning Systems*, 2024.

[12] C. Wang, Z. Yu, S. McAleer, et al., "ASP: Learn a universal neural solver!," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024.

[13] H. Cheng, H. Zheng, Y. Cong, et al., "Select and optimize: Learning to solve large-scale TSP instances," in *International Conference on Artificial Intelligence and Statistics*, PMLR, 2023, pp. 1219-1231.

[14] J. Choo, Y. D. Kwon, J. Kim, et al., "Simulation-guided beam search for neural combinatorial optimization," *Advances in Neural Information Processing Systems*, vol. 35, pp. 8760-8772, 2022.

[15] X. Pan, Y. Jin, Y. Ding, et al., "H-TSP: Hierarchically solving the large-scale traveling salesman problem," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, 2023, pp. 9345-9353.

[16] Z. Sun and Y. Yang, "Difusco: Graph-based diffusion solvers for combinatorial optimization," *Advances in Neural Information Processing Systems*, vol. 36, pp. 3706-3731, 2023.

[17] M. Wang, Y. Zhou, Z. Cao, et al., "An Efficient Diffusion-based Non-Autoregressive Solver for Traveling Salesman Problem," arXiv, 2025.

[18] X. Fang, J. Liu, and L. Wang, "Path optimization for mixed use of electric and fuel trucks under multiple distribution centers," *Engineering Letters*, vol. 33, no. 6, pp. 1919–1936, 2025.

[19] S. Li, Z. Yan, and C. Wu, "Learning to delegate for large-scale vehicle routing," *Advances in Neural Information Processing Systems*, vol. 34, pp. 26198-26211, 2021.

[20] Y. Ma, J. Li, Z. Cao, et al., "Learning to iteratively solve routing problems with dual-aspect collaborative transformer," *Advances in Neural Information Processing Systems*, vol. 34, pp. 11096-11107, 2021.

[21] Y. Wu, W. Song, Z. Cao, et al., "Learning large neighborhood search policy for integer programming," *Advances in Neural Information Processing Systems*, vol. 34, pp. 30075-30087, 2021.

[22] Y. Wu, W. Song, Z. Cao, et al., "Learning improvement heuristics for solving routing problems," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 9, pp. 5057-5069, 2021.

[23] L. Xin, W. Song, Z. Cao, et al., "Multi-decoder attention model with embedding glimpse for solving vehicle routing problems," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, 2021, pp. 12042-12049.

[24] K. Helsgaun, "An extension of the Lin-Kernighan-Helsgaun TSP solver for constrained traveling salesman and vehicle routing problems," Roskilde University, 2017, pp. 966-980.

[25] Z. H. Fu, K. B. Qiu, and H. Zha, "Generalize a small pre-trained model to arbitrarily large TSP instances," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, 2021, pp. 7474-7482.

[26] B. Hudson, Q. Li, M. Malencia, et al., "Graph Neural Network Guided Local Search for the Traveling Salesperson Problem," arXiv, 2022.

[27] H. Ye, J. Wang, H. Liang, et al., "GLOP: Learning global partition and local construction for solving large-scale routing problems in real-time," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, 2024, pp. 20284-20292.

[28] Z. Zheng, C. Zhou, X. Tong, M. Yuan, and Z. Wang, "UDC: A unified neural divide-and-conquer framework for large-scale combinatorial optimization problems," *Advances in Neural Information Processing Systems*, vol. 37, pp. 6081–6125, 2024.

[29] M. Wang, Y. Zhou, Z. Cao, Y. Xiao, X. Wu, W. Pang, Y. Jiang, H. Yang, P. Zhao, and Y. Li, "An efficient diffusion-based non-autoregressive solver for traveling salesman problem," *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2025.

[30] Y. Xiao, D. Wang, Z. Cao, R. Cao, X. Wu, B. Li, and Y. Zhou, "From global assessment to local selection: Efficiently solving traveling salesman problems of all sizes," arXiv:2025.

[31] C. K. Joshi, Q. Cappart, L. M. Rousseau, et al., "Learning the Travelling Salesperson Problem Requires Rethinking Generalization," *LIPIcs*, vol. 210, CP 2021, 2021, pp. 33:1-33:21.

[32] S. Lin, "An efficient heuristic algorithm for the traveling salesman problem," *Oper. Res.*, vol. 21, pp. 498-516, 1973.

[33] X. Bresson and T. Laurent, "An experimental study of neural networks for variable graphs," 2018.

[34] S. Elfwing, E. Uchibe, and K. Doya, "Sigmoid-weighted linear units for neural network function approximation in reinforcement learning," *Neural Networks*, vol. 107, pp. 3-11, 2018.

[35] K. He, X. Zhang, S. Ren, et al., "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770-778.

[36] W. Kool, H. van Hoof, and M. Welling, "Attention, learn to solve routing problems!" arXiv, 2019.

[37] Y. D. Kwon, J. Choo, B. Kim, et al., "POMO: Policy optimization with multiple optima for reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 21188-21198, 2020.

[38] M. Kim, S. Choi, H. Kim, et al., "Ant colony sampling with GFlowNets for combinatorial optimization," arXiv, 2024.